The Fourth International Workshop

on

Knowledge Discovery from Data Streams

The rapid growth in information science and technology in general and the complexity and volume of data in particular have introduced new challenges for the research community. Databases are growing incessantly and many sources produce data continuously. In most real world applications, the process generating the data is not strictly stationary. In many cases, we need to extract some sort of knowledge from this continuous stream of data. Examples include customer click streams, telephone records, large sets of web pages, multimedia data, scientific data, and sets of retail chain transactions. These sources are called data streams. Learning from data streams is an incremental task that requires incremental algorithms that take drift into account.

The goal of this workshop is to promote an interdisciplinary forum for researchers who deal with sequential learning, anytime learning, real-time learning, online learning, temporal and spatial learning, etc. from data streams. The workshop papers present significant contributions in learning decision trees, association rules, clustering, preprocessing, feature selection, etc. from data streams and related themes.

This is the fourth workshop in this series. The workshop web page is available at `http://www.machine-learning.eu/iwkdds-2006/`. There were 17 submissions. The program committee accepted 9 full papers, 4 short papers and 1 poster paper.

This year the workshop has a special emphasis on learning from sensor networks in ubiquitous environments. In that direction, the workshop includes also an invited talk, by Hillol Kargupta, one of the leading researchers in distributed mining data streams.

We would like to thank the ECML/PKDD 2006 organizers, the authors of the submitted papers, and the members of the Program Committee for all efforts to make this workshop possible.

J. Gama, R. Klinkenberg and J. Aguilar

Organization

Program Chairs

João Gama, University of Porto, Portugal
Ralf Klinkenberg, University of Dortmund, Germany
Jesús S. Aguilar-Ruiz, University of Seville, Spain

Program Committee

Michaela Black, University of Ulster, Coleraine, Northern Ireland, UK
Andre Carvalho, University of Sao Paulo, Brazil
Pedro Domingos, University of Washington, Seattle, WA, USA
Francisco Ferrer, University of Seville, Spain
Mohamed Gaber, Monash University, Victoria, Australia
João Gama, University of Porto, Portugal
Ray Hickey, University of Ulster, Coleraine, Northern Ireland, UK
Hillol Kargupta, University of Maryland, Baltimore, MD, USA
Ralf Klinkenberg, University of Dortmund, Germany
Jeremy Z. Kolter, Stanford University, CA, USA
Miroslav Kubat, University Miami, FL, USA
Mark Last, Ben-Gurion University, Israel
Mark Maloof, Georgetown University, Washington, DC, USA
S. Muthu Muthukrishnan, Rutgers University and AT&T Research, USA
Masayuki Numao, Osaka University, Japan
Pedro Pereira Rodrigues, University of Porto, Portugal
Josep Roure, Carnegie Mellon University, Pittsburgh, PA, USA
Jesús S. Aguilar-Ruiz, University of Pablo de Olavide, Spain
Bernhard Seeger, University Marburg, Germany
Elaine Parros Sousa, University of Sao Paulo, Brazil
Min Wang, IBM Watson Research Center, Hawthorne, NY, USA
Wei Wang, University of North Carolina, Chapel Hill, NC, USA
Xiaoyang Wang, University of Vermont, Burlington, VT, USA
Gerhard Widmer, University of Linz, Austria
Philip S. Yu, IBM Watson Research Center, Yorktown Heights, NY, USA

Sponsors

The workshop is supported by Kdubiq - Knowledge Discovery in Ubiquitous Environments - European Coordinated Action of FP6 and Adaptive Learning Systems II Project (POSI/EIA/55340/2004).

Table of Contents

Invited Talk

Contribution Papers

Peer-to-Peer Distributed Data Stream Mining and Monitoring

Hillol Kargupta
University of Maryland - Baltimore County, Baltimore, MD, USA

(Invited Talk)

Abstract:
Distributed data mining (DDM) deals with the problem of analyzing distributed, possibly multi-party data by paying attention to the computing, communication, storage, and human factors-related issues in a distributed environment. Unlike the conventional off-the-shelf centralized data mining products, DDM systems are based on fundamentally distributed algorithms that do not necessarily require centralization of data and other resources. DDM technology is finding increasing number of applications in many domains. Examples include data driven pervasive applications for mobile and embedded devices, grid-based large scale scientific and business data analysis, security and defense related applications involving analysis of multi-party possibly privacy-sensitive data, and peer-to-peer data stream mining in sensor and file-sharing networks. This talk will focus on peer-to-peer (P2P) distributed data stream mining and monitoring. It will first discuss the foundation of approximate and exact P2P algorithms for data analysis. Then it will present a class of P2P algorithms for eigen-analysis and clustering in details. The talk will end with a discussion on the future directions of research on P2P data mining.

Accurate Schema Matching on Streams

Szymon Jaroszewicz[1], Lenka Ivantysynova[2], Tobias Scheffer[2]

[1]National Institute of Telecommunications, Warsaw, Poland, sj@cs.umb.edu
[2]Humboldt-Universität zu Berlin, Germany
{ivantysy,scheffer}@informatik.hu-berlin.de

Abstract. We address the problem of matching imperfectly documented schemas of data streams and large databases. Instance-level schema matching algorithms identify likely correspondences between attributes by quantifying the similarity of their corresponding values. However, exact calculation of these similarities requires processing of all database records—which is infeasible for data streams. We devise a fast matching algorithm that uses only a small sample of records, and is yet guaranteed to match the most similar attributes with high probability. The method can be applied to any given (combination of) similarity metrics that can be estimated from a sample with bounded error; we apply the algorithm to several metrics. We give a rigorous proof of the method's correctness and report on experiments using large databases.

1 Introduction

In many practical situations, data mining processes are preceded by a preprocessing step in which multiple heterogeneous transaction databases are integrated into a single data warehouse. Integration requires the identification of semantic correspondences between attributes across multiple transaction database schemas [2]. In practical applications that often involve large databases, schema matching is a tedioustask, often further hindered by insufficient documentation.

Research revolves around the question how schema matching can be supported, or even automated, effectively. Schema matching algorithms generally match elements of the distinct schemas such that some similarity criterion is maximized. Schema-level matchers utilize a similarity criterion that refers to schema information such as attribute names, descriptions, or the schemas' structure. Unfortunately, the available schema information is often insufficient. Cases arise in which attribute names are opaque and clues on their semantics can only be found by inspecting the attributes' values. Instance-level schema matchers quantify the similarity of attributes by comparing properties of their values.

In order to guarantee that the produced mapping in fact maximizes the chosen similarity function, instance-level matchers need to exercise at least one entire pass over the databases. A complete pass is unreasonable for transactional databases that record high-speed data streams as they occur, for instance, in retail chains, and telecommunication and banking applications and can easily grow into the order of hundreds of terabytes. The ad-hoc solution of processing

only a small sample of transactions results in a loss of any guarantee on the optimality of the produced match. Known methods that utilize sampling do not guarantee any properties of the retrieved mappings.

We formalize the matching problem in a way that is both mathematically rigorous and closely tied to practical applications. We devise a schema matching algorithm, which is *guaranteed* to produce a "near-optimal" match; we detail the term "near-optimal" by means of probabilistic bounds on attribute similarity.

A useful similarity metric for the matching problem at hand is a base requirement for any sampling algorithm, and designing such a problem-specific metric clearly is a challenging issue in itself. Therefore, we design our solution to be generic with respect to the similarity criterion; weighted combinations of several schema-level and instance-level criteria can be applied. We review a selection of criteria, many more can be inserted into the algorithm.

The rest of this paper is organized as follows: after reviewing related work in Section 2, we formalize the problem setting in Section 3. We phrase the schema matching problem in a way that pays tribute to typical practical application scenarios, and that is sufficiently rigid to allow us to state and rigorously prove the correctness of our solutions. We detail our solution to the problem in Section 4, discuss attribute similarity metrics in Section 5 and report on experimental results on real-world data in Section 6. Section 7 concludes.

2 Related Work

Bernstein and Rahm [13] provide an overview of the schema matching problem and a taxonomy of schema matching algorithms. Matchers can be dichotomized into schema-level and instance-level methods. Schema-level matchers maximize some similarity function that refers to attribute names and other structural information (*e.g.,* [12, 6, 4]) whereas the similarity metrics employed by instance-level approaches [10] refer to the instance data; that is, to the attributes' values.

The instance-level approach can even be applied in the complete absence of useful schema-level information and it allows to even identify complex matching relations – *e.g.,* a factor-equivalence relation between an income attribute in Euros and a corresponding salary attribute in Yen [3].

The schema matching problem also plays a role in the exchange of messages in ecommerce applications [1], or the integration of databases due to merges or other changes of organizational structures. Most practical systems – such as Clio [16], SemInt [11], and LSD [5] – combine schema- and instance-level similarity metrics. Assembling and balancing the similarity metric requires experience and domain knowledge; machine learning [5] and experiments on synthetic matching problems can guide the parameter optimization [14].

Our algorithm relies on sampling and on data-dependent confidence bounds. Sampling strategies play an important role for a range of data mining tasks. They have been utilized to find the approximately most interesting association rules [7, 15], or the most significant differences between graphical models and corresponding databases [9].

3 Approximately Optimal Schema Matching

We focus on the following *schema matching problem setting*. Given are two schemas R and S over attributes $\langle r_1, \ldots, r_n \rangle$ and $\langle s_1, \ldots, s_m \rangle$, respectively. The goal is to identify semantically identical pairs of attributes (r_i, s_j).

In order to approach this problem computationally, a similarity metric $f_{R,S}(r_i, s_j)$ quantifies the similarity of attributes r_i and s_j. The similarity can exploit schema-level information such as attribute names if such information is available, and instance-level information. Instance-level similarity metrics refer to the values that occur in R and S. Naturally, instance-level information is always available—unless the databases are empty or inaccessible.

The contribution of this paper is relevant when the similarity metric $f_{R,S}(r_i, s_j)$ involves instance-level information. In order to evaluate an instance-level metric exactly, it is necessary to exercise a pass over the databases R and S; the main challenge that we address is that this is computationally infeasible for large databases and impossible for streams. The similarity criterion $f_{R,S}(r_i, s_j)$ is problem-specific and given in advance. We will only assume that it is possible to obtain an estimate of $f_{R,S}$ that lies within a bounded confidence interval.

In practice, the schema matching process is highly interactive. A user typically seeks to find out which attributes in R have counterparts in S. We phrase this problem setting as follows. *We seek to construct an algorithm that finds, for every attribute in R, a set of the k best matching candidates in S, sorted by their similarity to the attribute from R, and vice versa for all attributes in S. In addition, the algorithm has to order all attributes in R and S by their likelihood of having a counterpart in the other schema.* Based on a result of this form, a user may inspect the sorted attributes and their alleged matches and may dismiss all proposed matches occurring after some point as unlikely.

Our formulation of the problem setting builds upon ε-correct orderings of sequences. In an ε-correct ordering, values are sorted in decreasing order, but in case of near-ties between adjacent values (differences of no more than ε), any ordering is considered correct. The $\varepsilon = 0$ case corresponds to a regular ordering.

Definition 1 (ε-Correct Ordering). *A sequence of values $(x_1, \ldots, x_k)$ is in ε-correct ordering if, for all i between 1 and k and all j between 1 and $i-1$, the inequality $x_j \geq x_i - \varepsilon$ is satisfied.*

In order to formalize the *approximately optimal matching problem* we refer to stochastic approximations with confidence bounds. Intuitively, the parameters ε and δ that are involved have the following meaning. When the algorithm is restarted multiple times, then the resulting match is "ε-close" to the optimal match in a fraction of at least $1 - \delta$ of all runs.

Definition 2 (Approximately Optimal Matching). *Given schemas R and S over attributes $\langle r_1, \ldots, r_n \rangle$ and $\langle s_1, \ldots, s_m \rangle$ respectively, a desired number of matches k, approximation and confidence parameters ε and δ, find a sequence of k attributes $(s_{i_1}, \ldots, s_{i_k})$ for each r_i in R, a sequence of k attributes $(r_{j_1}, \ldots, r_{j_k})$ for each s_j in S, and permutations π^r and π^s such that, with confidence $1 - \delta$,*

1. *for all attributes* r_i*, the retrieved sequence* $(s_{i_1}, \ldots, s_{i_k})$ *contains the best matching attributes (accurately up to* ε*) and, analogously, the sequence* $(r_{j_1}, \ldots, r_{j_k})$ *contains the best matching attributes for each* s_j*; i.e., there is no offending* $s_{i'} \notin (s_{i_1}, \ldots, s_{i_k})$ *such that* $f_{R,S}(r_i, s_{i'}) > \min_{i'' \in \{i_1,\ldots,i_k\}} f_{R,S}(r_i, s_{i''}) + \varepsilon$ *and no* $r_{j'} \notin (r_{j_1}, \ldots, r_{j_k})$ *such that* $f_{R,S}(r_{j'}, s_j) > \min_{j'' \in \{j_1,\ldots,j_k\}} f_{R,S}(r_{j''}, s_j) + \varepsilon$*;*
2. *for all attributes* r_i*, the corresponding* $(s_{i_1}, \ldots, s_{i_k})$ *are ordered by their similarity to* r_i *and for all* s_j*, the corresponding* $(r_{j_1}, \ldots, r_{j_k})$ *are ordered by their similarity to* s_j*; i.e.,* $(f_{R,S}(r_i, s_{i_1}), \ldots, f_{R,S}(r_i, s_{i_k}))$ *and* $(f_{R,S}(r_{j_1}, s_j), \ldots, f_{R,S}(r_{j_k}, s_j))$ *are* ε*-correct orderings;*
3. *the permutations* π^r *and* π^s *sort the attributes of* R *and* S*, respectively, according to their likelihood of having a matching partner in the other schema; i.e., the sequences* $(\max_{j'} f_{R,S}(r_{\pi^r(1)}, s_{j'}), \ldots, \max_{j'} f_{R,S}(r_{\pi^r(n)}, s_{j'}))$ *and* $(\max_{i'} f_{R,S}(r_{i'}, s_{\pi^s(1)}), \ldots, \max_{i'} f_{R,S}(r_{i'}, s_{\pi^s(m)}))$ *are* ε*-correct orderings.*

4 Schema Matching Algorithms

We are about to present two algorithms that estimate the similarity of attributes based on samples R' and S' of records, drawn at random from R and S. They differ from each other in the way they derive the required sample size.

The first algorithm, FSM, determines a "worst-case" sample size without accessing the database or observing the stream; it only depends on the similarity metric and parameters ε and δ. The sample size suffices to assure the quality guarantee for any possible database; it may be pessimistically large for a given, particular database. The second algorithm is called progressive FSM. Its sample size depends on the database at hand; the sample-dependent bounds used within progressive FSM are technically more involved than the worst-case bounds of regular FSM. Therefore, FSM is a natural baseline for progressive FSM.

Both algorithms use *similarity confidence intervals* which bound the similarity $f_{R,S}$. A similarity confidence interval lays out a range of values such that, with a probability of at least $1-\delta$, the true similarity lies within that range. The confidence interval is wide for small samples and tightens for increasingly large samples. Similarity confidence intervals can be constructed for a wide range of instance-level similarity functions.

Definition 3 (Similarity Confidence Interval). *Let* $f_{R,S}$ *be an arbitrary similarity function. Let* R'*,* S' *be random samples from* R *and* S*; Then,* $\lfloor f_{R',S'} \rfloor_\delta$ *and* $\lceil f_{R',S'} \rceil_\delta$ *form a similarity confidence interval if, with probability at least* $1-\delta$*, there is no pair of any* $r_i \in R$ *and* $s_j \in S$ *that violates*

$$\lfloor f_{R',S'}(r_i, s_j) \rfloor_\delta \leq f_{R,S}(r_i, s_j) \leq \lceil f_{R',S'}(r_i, s_j) \rceil_\delta. \quad (1)$$

The first algorithm, "FSM", is detailed in Table 1. In Step 1, it uses a binary search to determine the smallest sample size N that suffices to guarantee that the similarity confidence intervals are tight up to ε. For a given similarity metric

Table 1. FSM: Fast Schema Matching Algorithm.

Input: Schemas R and S, respectively; desired number of candidates k; approximation and confidence parameters ε and δ.

1. **Let** N be the smallest number such that $\lceil f_{R',S'}(\cdot,\cdot)\rceil_\delta - \lfloor f_{R',S'}(\cdot,\cdot)\rfloor_\delta \leq \varepsilon$ when R' and S' contain N records. (Where we parameterize the bound with "$\cdot$", this means "for any argument".)
2. Draw N records at random from R and S, let them be R' and S'.
3. **For** all (r_i, s_j), determine $\hat{f}_{R',S'}\ (r_i, s_j) = \frac{1}{2}(\lceil f_{R',S'}(r_i, s_j)\rceil_\delta + \lfloor f_{R',S'}\ (r_i, s_j)\rfloor_\delta)$ based on the samples.
4. **For** all attributes r_i: **Let** $H^*_{r_i}$ be the k elements s_j that maximize $\hat{f}_{R',S'}(r_i, s_j)$; **For** all attributes s_j: **Let** $H^*_{s_j}$ be the k elements r_i that maximize $\hat{f}_{R',S'}(r_i, s_j)$.
5. **Let** π^r sort the attributes in R by $\max_{s_{i'} \in H^*_{r_i}} \hat{f}_{R',S'}(r_i, s_{i'})$; **let** π^s sort the attributes in S by $\max_{r_{j'} \in H^*_{s_j}} \hat{f}_{R',S'}(r_{j'}, s_j)$.
6. **Return** π^r, π^s, and for all attributes p in $R \cup S$: return the sequence of elements in H^*_p sorted by $\hat{f}_{R',S'}(p, q)$ as k best matches.

$f_{R,S}$, the corresponding confidence interval is inserted into the algorithm. (Parameterization with "$\cdot$" indicates "for any pair of attributes".) By contrast, the progressive FSM algorithm described in Table 2 starts to draw batches of records from the databases and then calculates confidence intervals in each step that depend on the sample processed so far. The progressive algorithm can terminate earlier than the sample-independent bound would have suggested, depending on characteristics of the database.

Our main result is that both, the FSM and progressive FSM algorithm solve the approximately optimal matching problem.

Theorem 1. *Let $\lceil f_{R',S'}(\cdot,\cdot)\rceil_\delta$ and $\lfloor f_{R',S'}(\cdot,\cdot)\rfloor_\delta$ be arbitrary similarity confidence bounds that satisfy Definition 3. Then both, the FSM (Table 1) and progressive FSM algorithm (Table 2) find, for any pair of schemas R and S and any input parameters $0 < \delta \leq 1$, $0 < \varepsilon$, and k, an approximately optimal matching as detailed in Definition 2.*

The proof of Theorem 1 is given in Appendix A of [8]. Copies of the implementation can be obtained for research purposes from the authors.

5 Attribute Similarity Metrics

We will now briefly sketch three exemplary similarity measures for attributes; we refer the reader to the corresponding technical report [8] for the full treatment. Various similarity measures require measuring the similarity of two probability distributions. For this purpose we will use a measure E based on Euclidean distance. Let $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$ be probability distributions. E is defined as $E(P, Q) = 2 - \sum_{i=1}^{n}(p_i - q_i)^2$. We subtract the sum of squares from 2 in order to guarantee that E has high values for similar distributions, while remaining nonnegative.

Table 2. Progressive FSM Algorithm.

Input: Schemas R and S, respectively; desired number of candidates k; approximation and confidence parameters ε and δ.

Let $R' = S' = \emptyset$. **Let** the batch size parameter be 10,000, by default. **Let** $t = 1$.

Let N be the smallest number such that $\lceil f_{R',S'}(\cdot,\cdot)\rceil_{\frac{\delta}{2}} - \lfloor f_{R',S'}(\cdot,\cdot)\rfloor_{\frac{\delta}{2}} \leq \varepsilon$ when R' and S' contain N records. **Let** $T = N/\text{batch size}$.

For $t = 1 \ldots T$ **Repeat:**

1. **Let** $\delta_t = \frac{\delta}{2t(t+1)}$.
2. Draw batches of records from R and S, add them to R' and S', respectively.
3. **For** all (r_i, s_j), calculate $\lfloor f_{R',S'}(r_i,s_j)\rfloor_{\delta_t}$, $\lceil f_{R',S'}(r_i,s_j)\rceil_{\delta_t}$, and $\hat{f}_{R',S'}(r_i,s_j) = \frac{1}{2}(\lceil f_{R',S'}(r_i,s_j)\rceil_{\delta_t} + \lfloor f_{R',S'}(r_i,s_j)\rfloor_{\delta_t})$. $(\lfloor f_{R',S'}(r_i,s_j)\rfloor_{\delta} + \lceil f_{R',S'}(r_i,s_j)\rceil_{\delta})$.
4. **If** all pairs (r_i, s_i) satisfy that $\lceil f_{R',S'}(r_i,s_j)\rceil_{\delta_t} - \lfloor f_{R',S'}(r_i,s_j)\rfloor_{\delta_t} \leq \varepsilon$ **then break.**

For all attributes r_i: **Let** $H^*_{r_i}$ be the k elements s_j that maximize $\hat{f}_{R',S'}(r_i,s_j)$; **For** all attributes s_j: **Let** $H^*_{s_j}$ be the k elements r_i that maximize $\hat{f}_{R',S'}(r_i,s_j)$.

Let π^r sort the attributes in R by $\max_{s^*_i \in H^*_{r_i}} \hat{f}_{R',S'}(r_i, s^*_i)$; **let** π^s sort the attributes in S by $\max_{r^*_j \in H^*_{s_j}} \hat{f}_{R',S'}(r^*_j, s_j)$.

Return π^r, π^s, and for all attributes p in $R \cup S$: return the sequence of elements in H^*_p sorted by $\hat{f}_{R',S'}(p,q)$ as k best matches.

Euclidean Distance. Given two attributes r and s with identical domains, a natural measure of their similarity is given by $f^E_{R,S}(r,s) = E(P_r, Q_s)$, where P_r and Q_s are probability distributions of r and s respectively. The upper and lower bounds are then

$$\lfloor f^E_{R',S'}(r,s)\rfloor_\delta = \left\{2 - \sum_{i=1}^{n_r}\left((\lceil p_{r,i}\rceil_{\frac{\delta}{M}})^2 + (\lceil q_{s,i}\rceil_{\frac{\delta}{M}})^2 - 2\lfloor p_{r,i}\rfloor_{\frac{\delta}{M}}\lfloor q_{s,i}\rfloor_{\frac{\delta}{M}}\right)\right\},$$

$$\lceil f^E_{R',S'}(r,s)\rceil_\delta = \left\{2 - \sum_{i=1}^{n_r}\left((\lfloor p_{r,i}\rfloor_{\frac{\delta}{M}})^2 + (\lfloor q_{s,i}\rfloor_{\frac{\delta}{M}})^2 - 2\lceil p_{r,i}\rceil_{\frac{\delta}{M}}\lceil q_{s,i}\rceil_{\frac{\delta}{M}}\right)\right\}.$$

Permuted Euclidean Distance. When comparing two categorical attributes, it is not clear which values correspond to each other. Therefore, all possible permutations of values in the domains of attributes have to be considered. The similarity measure is now defined as

$$f^{\pi E}_{R,S}(r,s) = \max_{\pi \in \Pi(n)} E\left(P_r, \pi Q_s\right), \tag{2}$$

where $\Pi(n)$ is the set of all permutations of $\{1,\ldots,n\}$. The following similarity confidence intervals satisfy Definition 3.

$$\lfloor f^{\pi E}_{R',S'}(r,s)\rfloor_\delta = \max_{\pi\in\Pi}\lfloor f^E_{R',S'}(r,\pi s)\rfloor_\delta; \qquad \lceil f^{\pi E}_{R',S'}(r,s)\rceil_\delta = \max_{\pi\in\Pi}\lceil f^E_{R',S'}(r,\pi s)\rceil_\delta.$$

Bigrams. This measure is based on the distribution of bigrams (*i.e.*, two character subsequences) in attribute values. We hash all possible bigrams into B

buckets; the resulting bounds are

$$\lfloor f^{Bg}_{R',S'}(r,s)\rfloor_\delta = 2 - \sum_{i=1}^{n_{r'}}\Big[(\lceil p_{r',i}\rceil_{\frac{\delta}{M}})^2 + (\lceil q_{s',i}\rceil_{\frac{\delta}{M}})^2 - 2\lfloor p_{r',i}\rfloor_{\frac{\delta}{M}}\lfloor q_{s',i}\rfloor_{\frac{\delta}{M}}\Big],$$

$$\lceil f^{Bg}_{R',S'}(r,s)\rceil_\delta = 2 - \sum_{i=1}^{n_{r'}}\Big[(\lfloor p_{r',i}\rfloor_{\frac{\delta}{M}})^2 + (\lfloor q_{s',i}\rfloor_{\frac{\delta}{M}})^2 - 2\lceil p_{r',i}\rceil_{\frac{\delta}{M}}\lceil q_{s',i}\rceil_{\frac{\delta}{M}}\Big].$$

Character Proportions. This measure compares the proportions of characters that fall into predefined subsets. We use four subsets: lowercase letters, uppercase letters, digits, and an additional set for all remaining printable characters. Vector $P^r = (P^r_l, P^r_u, P^r_d, P^r_p)$ contains the averaged proportions of records belonging too these classes. The similarity measure can now be defined as $f^c_{R,S}(r,s) = E(P^r, P^s)$. Bounds for $f^c_{R,S}$ are defined analogously to the above case.

Weighted Sum. A weighted sum of all similarity measures forms the most general case. Any weighted subset of measures can be obtained by setting weights to either zero or desired nonzero values.

$$f_{R,S}(r,s) = \frac{w_{\pi E} f^{\pi E}_{R,S}(r,s) + w_{Bg} f^{Bg}_{R,S}(r,s) + w_c f^c_{R,S}(r,s)}{w_{\pi E} + w_{Bg} + w_c} \tag{3}$$

The resulting bound for the weighted sum is simply the weighted sum of the corresponding bounds.

Combining Schema- and Instance-Level. When schema-level information is available – for instance in the form of attribute names or descriptions – it is advisable to utilize this information. The similarity function can combine schema- and instance-level components: $f_{R,S}(r,s) = w_S f^{Schema}(r,s) + w_I f^{Instance}_{R,S}(r,s)$. For instance, f^{Schema} can quantify the similarity of attributes' names and descriptions. Since f^{Schema} is independent of the database, it follows immediately that the bounds are

$$\lfloor f_{R',S'}(r,s)\rfloor = w_S f^{Schema} + w_I \lfloor f^{Instance}_{R',S'}(r,s)\rfloor, \text{ and}$$

$$\lceil f_{R',S'}(r,s)\rceil = w_S f^{Schema} + w_I \lceil f^{Instance}_{R',S'}(r,s)\rceil.$$

6 Experiments

In our experiments we want to investigate (a) whether the FSM and progressive FSM algorithms are practically applicable for large databases; we want to (b) compare the performances of FSM (using Hoeffding's and Normal bounds), progressive FSM (using Normal bounds), and a baseline instance-based matcher. The baseline matcher uses the same similarity function but processes the entire database, thus always retrieving the matches that maximize the similarity on the database. Theorem 1 *guarantees* that FSM and progressive FSM return approximately optimal matches; nevertheless, we will (c) empirically study the chance of the algorithms finding the correct matches for attributes.

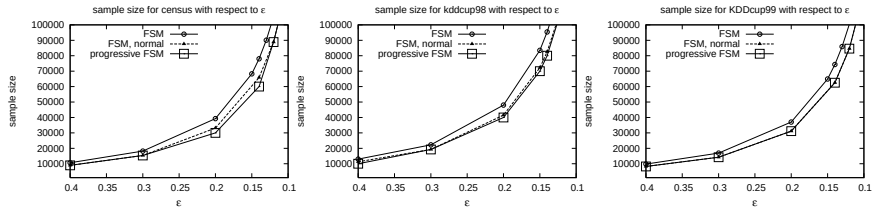


Fig. 1. Sample size against ε for schema matching algorithms.

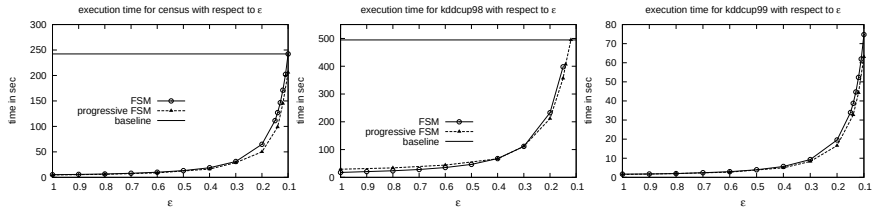


Fig. 2. Execution time against ε for schema matching algorithms.

The similarity function used for all experiments is the weighted average of all similarity functions studied in Section 5, with all weights fixed to 1. We use three publicly available databases: the KDDCup 1998 customer relationship management database [1] contains with 481 attributes and over 190,000 records; the KDDCup 1999 database [2] with nearly 5 million records and 42 attributes, 17 of which are almost always zero; and the census database [3] with 42 attributes (the majority of them contain text) and close to 300,000 records.

In order to conduct controlled experiments, we randomly split each of the databases into two parts containing half of the records. We then use the schema matching algorithms to match the two halves of the databases. When the algorithm matches an attribute with itself, this is counted as a true positive. Based on the number of attributes that are matched with itself, we determine precision, recall, and F-measure.

Figure 1 shows the number of database records that FSM and progressive FSM draw from the database and process before finding an approximately optimal match. For the census and KDD Cup 1998 data, the early stopping criterion (Step 4) of progressive FSM occasionally kicks in, reducing the number of samples on average compared to FSM. For the KDD Cup 1999 database, early stopping in Step 4 is never exercised. The reason is that due to the sparseness of the data the matching problem is very hard. For all databases, we observe that the Normal bounds reduce the required sample size over the Hoeffding bounds used by the FSM algorithm.

Figure 2 compares the execution time of FSM, progressive FSM, and the baseline algorithm that executes a pass over the entire database for varying values of ε (with $\delta = 0.1$). Again, we observe that progressive FSM is the fastest

[1] http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html
[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[3] http://kdd.ics.uci.edu/databases/census-income/census-income.html

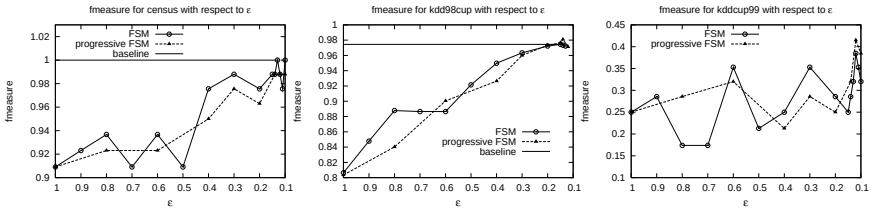


Fig. 3. F-measure against ε for schema matching algorithms.

method, followed by FSM. For the KDD Cup 1999 database, the baseline algorithm exceeds our patience.

Finally, Figure 3 compares the F-measure of FSM, progressive FSM and the baseline algorithm that processes the entire database. For the census and KDD Cup 1998 databases, the F-measures are above 0.9 and 0.8, respectively, even for $\varepsilon = 1$! For decreasing values of ε, the F-measure grows further. The KDD Cup 1999 database, by contrast, is very difficult. Since many attributes almost always assume a value of zero, most similarity values are very close and it is difficult to identify the best match; the F-measure is lower, accordingly.

Note that the low F-measure for the KDD Cup 1999 database is not in contradiction to the guarantee provided by FSM: The F-measure quantifies how frequently attributes are matched to their actually semantically equivalent partner (in our experimental setting, to themselves). By contrast, Theorem 1 guarantees that the retrieved match for each attribute is "nearly as good as the optimal match". The baseline algorithm processes the entire database in the first place and therefore obtains a constantly high F- measure for the census and KDD Cup 1998 databases. For the KDD Cup 1999 problem, the baseline algorithm exceeds our patience, no result can be provided. For the FSM and progressive FSM algorithms, execution time and sample size as well as the quality of the solution depend on the parameters ε and δ.

7 Conclusion

Our formulation of the approximately optimal schema matching problem is closely tied to practical applications and at the same time mathematically rigorous. The FSM and progressive FSM algorithms provably solve this problem; that is, for each attribute of either schema, they find the k approximately best matching partners, and approximately order them according to their similarity. Finally, the attributes in either schema are approximately sorted according to their likelihood of having a partner in the other schema.

The required sample size of the algorithms depends on parameters ε and δ, but is independent of the database size. The database can even be an infinite stream. For the progressive FSM algorithm, sample size depends on the actual databases, if there are some similar but many dissimilar potential matches, then progressive FSM can identify the similar matches faster and terminate early.

Our experiments lead to a number of conclusions. (a) FSM and progressive FSM are feasible and practically applicable for streams and very large databases.

They can be applied to a range of similarity metrics; we specified bounds for a selection of measures that can easily be extended. (b) FSM with Normal bounds and progressive FSM are equally fast for difficult matching problems with many very similar attributes. Progressive FSM is faster than FSM otherwise. The Normal bounds outperform the Hoeffding bounds. (c) While the theoretical results guarantee an approximately optimal match, we observe empirically that attributes are often actually matched to semantically equivalent attributes.

References

1. D. Aumueller, H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *Proceedings of the ACM SIGMOD Conference*, 2005.
2. P. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *Proceedings of the International Conference on Entity Relationship Modeling*, 2000.
3. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *Proceedings of the ACM SIGMOD Conference*, 2004.
4. H.-H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases*, 2002.
5. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proceedings of the ACM SIGMOD Conference*, 2001.
6. A. Doan, P. Domingos, and A. Levy. Learning source descriptions for data integration. In *Proceedings of te WebDB Workshop*, 2000.
7. C. Domingo, R. Gavalda, and Osamu Watanabe. Adaptive sampling methods for scaling up knowledge discovery algorithms. *Data Mining and Knowledge Discovery*, 6(2):131–152, 2002.
8. S. Jaroszewicz, L. Ivantysynova, and T. Scheffer. Accurate schema matching on streams. Technical report, Humboldt-Universität zu Berlin, 2006.
9. S. Jaroszewicz and T. Scheffer. Fast discovery of unexpected patterns in data, relative to a bayesian network. In *Proceedings of the ACM SIGKDD Conference*, 2005.
10. W. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proceedings of the International Conference on Very Large Databases*, 1994.
11. W. Li and C. Clifton. Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural network. *Data and Knowledge Engineering*, 33(1):49–84, 2000.
12. J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Databases*, 2001.
13. E. Rahm and P. Bernstein. A survey of approaches to automatic schema mapping. *VLDB Journal*, 10:334–350, 2001.
14. M. Sayyadian, Y. Lee, A. Doan, and Arnon Rosenthal. Tuning schema matching software using synthetic scenarios. In *Proceedings of the VLDB Conference*, 2005.
15. T. Scheffer and S. Wrobel. Finding the most interesting patterns in a database quickly by using sequential sampling. *Journal of Machine Learning Research*, 3:833–862, 2002.
16. L. Yan, R. Miller, L. Haas, and R. Fagin. Data driven understanding and refinement of schema mappings. In *Proceedings of the ACM SIGMOD Conference*, 2001.

High-Order Substate Chain Prediction Based on Massive Sensor Outputs

Nguyen Viet Phuong and Takashi Washio

I.S.I.R., Osaka University, 8-1, Mihogaoka, Ibaraki City, Osaka, 567-0047, Japan
{vphuong, washio}@ar.sanken.osaka-u.ac.jp

Abstract. Along the development of ubiquitous sensing technologies, the opportunity to have transaction time series data is increasing. We propose a novel framework named HISC modeling to predict the dynamic system behaviors based on the transaction time series which contains explosive states due to the combinatorics of massive sensors and their output values with noise. Its significant performance has been confirmed through the comparisons with high-order Markov chain models and the application to practical data analysis.

1 Introduction

Along the development of ubiquitous sensing technologies, the numbers of sensors installed in various systems such as automobiles [1] and home automation [2] are rapidly increasing in recent years. In such a system, the majority of sensors produce their signals in event driven manners to save their communication cost. Accordingly, the system output is a transaction time series which has the background of extremely high dimensional sensing variables. On the other hand, the dynamics of the subsystems in many large scale systems are not tightly coupled with the others as in the case of the small world phenomena [3]. Accordingly, the identification of each substate of the subsystems and the dynamic transitions among the substates is expected to be an efficient remedy for the dynamic system behavior prediction under massive transaction time series.

Over the last decade, sequence mining techniques have been studied to search frequent sequential patterns embedded in massive sequence data. AprioriSome/ AprioriAll [4] and WINEPI/MINEPI [5] derive all frequent subsequence patterns from transaction sequences and an event symbol sequence respectively. However, these studies remain to provide fragmentary patterns but not any synthetic model for the state prediction. In statistics, Hidden Markov Modeling (HMM) and HIgh-order Markov Chain (HIMC) modeling have been extensively studied [6]. Especially, the HIMC modeling has the modeling ability nearly equivalent to that of HMM while maintaining efficient and stable model estimation.

Though various HIMC modeling approaches such as AR and ARMA modeling have been studied for a continuous number field [6], only a limited number of studies have been reported for the symbolic time series data. This is due to the combinatorics over the past and the present discrete states. To alleviate this difficulty, Mixture Transition Distribution (MTD) modeling introduced

an approximation to take a linear combination of the 1st order Markov chain probabilities over the past states within the model order [7]. This approximation suppresses the modeling complexity to linear dependency on the model order while reducing the modeling accuracy. The Maximum Log-Likelihood method is used to estimate the MTD model. Variable Length Markov Chain modeling (VLMC) uses more flexible approximation by adapting the model order to each key subsequence of the states represented in a probabilistic suffix tree (PST) [8]. A key subsequence is the subsequence appearing in the time series more frequently than a given minimum support and having a significant confidence to predict its next state. This provides a compactly well-approximated model. However, to our best knowledge, almost all modeling approaches for the time series data at present assume each sample represented by a limited dimensional state vector or a limited state symbol but not the aforementioned state transaction. The practical limit of states and variables which can be introduced into the models is usually less than a few tens.

Upon these considerations, we propose a novel framework named "*HIgh-order Substate Chain* (HISC) *modeling*" to predict dynamic system behaviors from a time series of transactions containing massive sensing states. HISC represents a probabilistic and dynamic relation between a finite number of past major substates and present major substates. In the current research, the value of each sensor output is assumed to be categorical without loss of generality as well as the ordinary Markov modeling. Appropriate signal discretization can be combined with our framework. Furthermore, these modeling and prediction functions are implemented into a tool named "HISCMiner". Significant performance of our proposing framework has been confirmed though the comparisons with MTD and VLMC approaches and the application to practical data analysis.

2 Proposing Framework

2.1 HIgh-order Substate Chain (HISC) model

We consider the following time series data D of transactions containing items.

$$D = X_1 ... X_n$$

where $X_t = \{item_1^t, ..., item_{m_t}^t\}$. Under the setting of a large scale system having massive sensors, each item corresponds to a pair of a senor name and its sensing output value. As many sensors output their values in event-driven manners, each transaction usually contains only a subset of all sensor outputs. First, we consider to apply the standard HIgh-order Markov Chain (HIMC) model to D. Let M be a finite set of state symbols. The HIMC model is given by the following state transition probability.

$$p_{\{s_1,...,s_\ell\},s_0}(t) = P(X_t = s_0 | X_{t-1} = s_1, ..., X_{t-\ell} = s_\ell)$$

where ℓ is the model order and $s_0, ..., s_\ell \in M$. A natural application of this HIMC model to D is done by labeling each state s_i by a transaction. However,

as mentioned earlier, the number of the states, M, labeled by the transactions is explosively large due to the combinatorics of the sensors, the variety of their output values and their noise. The transitions among some major states grabbing the system dynamics may be hardly identified under this modeling condition.

To overcome this difficulty, we propose a novel framework for the HISC modeling, where each transition rule is limited to a frequent combination of substates and a transition among them. Let $S = S_1...S_\ell$ be a subsequence consisting of itemsets $\{S_1, ..., S_\ell\}$. Then a counter of S is defined as

$$\chi_S^t = \begin{cases} 1 \text{ if } S_1 \subseteq X_t, ..., S_\ell \subseteq X_{t-\ell+1} \\ 0 \text{ otherwise} \end{cases}.$$

Note that $S_i \subseteq X_{t-i+1}$ always holds if $S_i = \phi$. From the total number of occurrence of S in D: $\chi_S = \sum_t \chi_S^t$ and the total number of transaction subsequences of length ℓ in D: $L_\ell = n - (\ell - 1)$, the support of S in D and the confidence of an itemset S_0 to appear right after S are defined as

$$sup(S) = \frac{\chi_S}{L_\ell}, \quad conf(S_0|S) = \frac{\chi_{S_0 S}}{\chi_S}.$$

Given a minimum support $minsup$, the HISC model is given by the following transition probability under the assumption that the probability is time invariant.

$$\begin{array}{l} conf(S_0|S) \simeq \\ P(S_0 \subseteq X_t | S_1 \subseteq X_{t-1}, ..., S_\ell \subseteq X_{t-\ell}) \ \text{ s.t. } sup(S_0 S) \geq minsup. \end{array} \tag{1}$$

Because the sequence $S_0 S_1 ... S_\ell$ is frequent, its subsequence $S'_0 ... S'_{\ell'}$ ($\subseteq S_0 S_1 ... S_\ell$) is also frequent. Here, $S'_0 ... S'_{\ell'} \subseteq S_0 ... S_\ell$ stands for the relation $S'_0 \subseteq S_i \wedge ... \wedge S'_{\ell'} \subseteq S_{i+\ell'}$ ($\ell' = 0, ..., \ell; i = 0, ..., \ell - \ell'$). Thus the transition probability $P(S'_0 \subseteq X_t | S'_1 \subseteq X_{t-1}, ..., S'_{\ell'} \subseteq X_{t-\ell'})$ is included in the HISC model. This fact indicates that a HISC model contains many transition rules and their probabilities which are mutually dependent. Accordingly, the HISC model is not a Markov model, and consists of the transitions among many dependent substates.

2.2 HISC modeling approach

According to Eq. (1), the HISC modeling requires to derive a complete set of the frequent subsequences of a length $\ell + 1$, *i.e.*, $FS^\ell = \{S_0 S | S_0 S \subseteq D, sup(S_0 S) \geq minsup\}$, together with their confidence values $conf(S_0|S)$. To obtain this information, two step processes are applied. The first step is data preprocessing to cut out all subsequences from D by using a moving window of the width $\ell + 1$ for the model order ℓ. This transforms D to the following set of transactions concatenated over the moving window.

$$D^\ell = \{Y_t^\ell | t = \ell + 1, ..., n\},$$

where

$$Y_t^\ell = X_t \cup ... \cup X_{t-\ell} = \{item_1^t, ..., item_{m_t}^t, ..., item_1^{t-\ell}, ..., item_{m_{t-\ell}}^{t-\ell}\}.$$

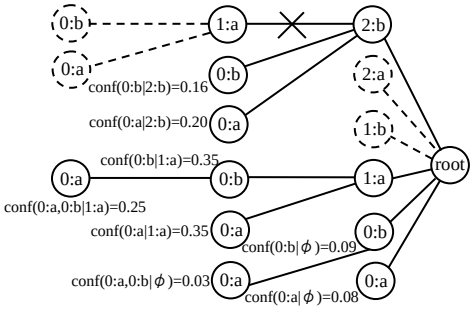


Fig. 1. HISC model represented by a trie.

Moreover, each $item^{t-i}$ in a transaction is prefixed by its time lag i as $i : item^{t-i}$, and the items in every transaction are sorted in descending order, *i.e.*, every Y_t^ℓ is reversed in order. Then each transaction is further transformed in D^ℓ as

$$Y_t^\ell = \{\ell : item_{m_{t-\ell}}^{t-\ell}, ..., \ell : item_1^{t-\ell}, ..., 0 : item_{m_t}^t, ..., 0 : item_1^t\}.$$

The second step is to apply an altered basket analysis to D^ℓ to derive $FS^\ell = \{S_0S | S_0S \subseteq D, sup(S_0S) \geq minsup\}$ [9]. In our implementation, the Apriori-based levelwise search of the frequent itemsets is applied by using a trie data structure as depicted in Fig. 1 which is an example of the 2nd order model. In the trie, each node is labeled by an item and sorted in descending order from the left (from the top in Fig. 1). The node corresponds to a candidate frequent itemset containing the labeling items from the root to the node. The nodes of solid lines are frequent, and these of the dashed lines are not frequent. For instance, the solid node(0:a) at the left most in Fig. 1 represents a frequent subsequence {0:a,0:b,1:a}. Because the objective frequent itemsets are limited in the form "S_0S," the nodes which are not in the level of the time lag 0 and does not have any further children are pruned for the efficient memory usage. node(1:a) at the top line in Fig. 1 is pruned since it does not have any frequent child of the time lag 0. Every node labeled by an item of the time lag 0 in the trie stands for a transition rule from S to S_0, and its confidence $conf(S_0|S)$ is stored at the node as shown in the figure. The trie obtained by the Apriori algorithm adapted to our HISC modeling includes all relations of Eq. (1), *i.e.*, the substate transition rules having their support more than *minsup* and their transition probabilities. Hence, this pruned trie is a compact representation of the HISC model.

2.3 Prediction method

As mentioned earlier, the HISC model consists of the transitions rules and their probabilities among many dependent substates. Accordingly, multiple transition rules which are mutually dependent are applicable to compute the next substates and their probabilities under a given transaction subsequence. For example, if we apply the HISC model in Fig. 1 to a transaction subsequence {1:a,1:b,1:c}{2:a,2:c}, the following four transition rules are applicable.

$conf(\{0{:}a\}\{0{:}b\}|\{1{:}a\}) = P(\{0{:}a\}\{0{:}b\} \subseteq X_t|\{1{:}a\} \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.25$
$conf(\{0{:}b\}|\{1{:}a\}) = P(\{0{:}b\} \subseteq X_t|\{1{:}a\} \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.35$
$conf(\{0{:}a\}|\{1{:}a\}) = P(\{0{:}a\} \subseteq X_t|\{1{:}a\} \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.35$
$conf(\{0{:}a\}\{0{:}b\}|\phi) = P(\{0{:}a\}\{0{:}b\} \subseteq X_t|\phi \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.03$
$conf(\{0{:}b\}|\phi) = P(\{0{:}b\} \subseteq X_t|\phi \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.09$
$conf(\{0{:}a\}|\phi) = P(\{0{:}a\} \subseteq X_t|\phi \subseteq \{1{:}a,1{:}b,1{:}c\}, \phi \subseteq \{2{:}a,2{:}c\}) = 0.08$

To compute the next substates, we have to synthesize the consequences of the multiple rules under some criteria. The criteria and the way of the synthesis may be up to the objective of the model use. In this paper, we investigate the criteria and the computational procedure for the prediction of the system substate. The following two criteria are considered to be obviously important.

(A) The system substate must be specifically computed as much as it can.
(B) The most probable substate must be computed.

To predict the next substate of a given transaction subsequence $D_s(\subseteq D)$, candidate transition rules having the maximal itemset predictions S_0 from S, where S matches to D_s from the beginning, are selected according to the criterion (A). More formally, a set of the candidate transition rules to predict the next substate of D_s is as follows.

$$CTR(D_s) = \{S_0S \mid S_0S \in FS^\ell \wedge S \subseteq_{ini} Ds \text{ and } \nexists S_0^*S^*, S_0 \subset S_0^* \text{ s.t. } S_0^*S^* \in FS^\ell \wedge S^* \subseteq_{ini} Ds\}$$

Here, $S' \subseteq_{ini} S$ stands for the relation $S'_1 \subseteq S_1 \wedge ... \wedge S'_{\ell'} \subseteq S_{\ell'}$ $(\ell' \leq \ell)$. Then the transition rule having the maximum confidence among the candidates in $CTR(D_s)$ as

$$S_0S = argmax_{S_0S \in CTR(D_s)} conf(S_0|S)$$

is used for the prediction according to the criterion (B). Hence, the next substate of D_s is predicted as unique S_0 satisfying these conditions. In the above example, given D_s ={1:a,1:b,1:c}{2:a,2:c}, $CTR(D_s)$ = {{0:a,0:b}{1:a}, {0:a,0:b}} is derived since {0:a,0:b} is the maximal frequent itemset under D_s. Then, the rule having the highest confidence {0:a,0:b}{1:a} is chosen, and {0:a,0:b} is predicted for the next substate.

3 Performance Evaluation

The aforementioned functions of HISC modeling and prediction are implemented into a tool named HISCMiner by using C++ language. In this section, the performance of HISCMiner is assessed in experimental comparisons with some conventional HIMC approaches by using synthetic data. Furthermore, its practicality is demonstrated through an experimental application to a real world data. All experiments have been conducted by using a personal computer with 3.0GHz Pentium 4 CPU and 2GB RAM.

3.1 Experimental setting using synthetic data

We constructed a synthetic data generation program. Let I be a set of all items which appear in the synthetic data. Then the program generates major substates S by randomly choosing $|S|$ items from I where $|S|$ is determined by a Gaussian random value having the average $\overline{|S|}$ and the unit variance. Let SS be a set of generated S. Let ℓ be a model order, R a state transition rule and RS the set of R. R is randomly generated by following the form bellow where each S_i is a substate in SS and c a number of the candidate next substates.

$$R = \{S_0^1, ..., S_0^c\}S_1...S_\ell. \tag{2}$$

Starting from an initial transaction sequence $X_1...X_\ell$, R whose conditional part satisfies $S_1...S_\ell \subseteq_{ini} X_1...X_\ell$ is used to derive a next substate S_0. The next substate is chosen under probability $P_R(S_0^i)$ of R where $\sum_{i=1}^{c} P_R(S_0^i) = 1$. Then $X_{\ell+1}$ which is obtained by $X_{\ell+1} = S_0 \cup N$ where N is a noise itemset randomly chosen from I. $|N|$ is determined by a Gaussian random value having the average $\overline{|N|}$ and the unit variance. This process is repeated by incrementing all time indices to generate a transaction time series $D = \{X_t | t = 1, ..., n\}$. By default, the parameter values of $|I| = 5000$, $|SS|$=1000, $\overline{|S|} = 3$, $|RS| = 10$, $\ell = 3$, $c = 3$, $\overline{|N|} = 7$ and $n = 10000$ are used. Thus, this is a Markov chain process embedded in very noisy transaction time series having a large number of background items and substates.

Because our HISCMiner estimates the substates but not the total states, the ordinary definition of estimation error is not applicable to the performance evaluation. Given a true substate S_t generated in the data synthesis and the substate S_e estimated by the HISCMiner, they may not perfectly match as $S_t \neq S_e$ but overlap as $S_t \cap S_e \neq \phi$. A simple way to evaluate the error between these two substates is to measure the ratio of their different part over their join as $|S_t \cup S_e - S_t \cap S_e|/|S_t \cup S_e|$. However, if S_t and/or S_e do not appear very frequently, their error does not strongly affect the performance of the HISCMiner. Accordingly, a better way to evaluate their error is to take into account the frequency of their appearance in D. This can be achieved by using the following information of each item in a substate, $Info(item)$, instead of the cardinality of the substate.

$$Info(item) = -\log_2(p(item)),$$

where $p(item)$ is the empirical probability of the $item$ in D. Then the substate error between S_t and S_e is formulated as

$$E_S(S_t, S_e) = \frac{\sum\limits_{item \in S_t \cup S_e} Info(item) - \sum\limits_{item \in S_t \cap S_e} Info(item)}{\sum\limits_{item \in S_t \cup S_e} Info(item)}.$$

When a time series of S_t having a length n and the time series of their estimations S_e are given, their average substate error is defined as

$$\overline{E_S(S_t, S_e)} = \frac{\sum\limits_{i=1}^{n} E_S(S_t, S_e)(i)}{n}. \tag{3}$$

Table 1. Comparison with MTD and VLMC

Data set	HISC	MTD	VLMC
#3-3-0-5000	0.321	0.383	0.341
#3-3-2-5000	0.279	1.000	0.858
#3-3-4-5000	0.309	1.000	0.886
#3-3-2-5000	0.279	1.000	0.858
#3-3-2-100	0.350	1.000	0.837
#3-3-2-50	0.397	1.000	0.817
#3-3-2-10	0.653	0.933	0.598

Table 2. Parameters for Pioneer-1

Setting	Move Data	Turn Data
	minsup ; *disc*	*minsup* ; *disc*
#1	0.0400 ; 5	0.0250 ; 5
#2	0.0180 ; 10	0.0210 ; 10
#3	0.0075 ; 30	0.0150 ; 30
#4	0.0075 ; 50	0.0150 ; 50
#5	0.0075 ; 100	0.0150 ; 100
#6	0.0075 ; 300	0.0150 ; 300
#7	0.0085 ; 30	0.0300 ; 30
#8	0.0095 ; 30	0.0450 ; 30
#9	0.0110 ; 30	0.0600 ; 30

3.2 Comparison with conventional methods

HIMCMiner was compared with MTD [7] and VLMC [8] in terms of the model based prediction accuracy by using some synthetic datasets. We obtained the MATLAB code of MTD modeling from a download site [11] and the binary code of VLMC from its authors [8]. The transaction time series of the synthetic data is divided into two parts by 1:1. The former is used for the training and the rest for the testing. Because MTS and VLMC accept only symbolic sequences but not transaction sequences, each transaction state containing a substate and a noise itemset is mapped to a unique character in the preprocessing for VLMC and MTD. 6 datasets having different simulation parameters of the model order ℓ, the average major substate size $\overline{|S|}$, the average noise itemset size $\overline{|N|}$ and the item space size $|I|$ are generated. They are labeled as # ℓ-$\overline{|S|}$-$\overline{|N|}$-$|I|$. In the transition probability distribution $P_R(S_0^i)$ $(i = 1, ..., c)$ of the rule Eq. (2), one candidate next substate is set to be 80%, and the others are set to share the rest 20%. In the application of these three modeling methods, their model order is set to be ℓ used in the data synthesis. *minsup* for HISCMiner and VLMC was set at sufficiently low 3% to search all major substates in the datasets.

Table 1 shows the result of $\overline{E_S(S_t, S_e)}$ for the three methods. In the case of #3-3-0-5000 where no noise is contained, the errors of the three methods are nearly same. However, under the larger noise such as #3-3-2-5000 and #3-3-4-5000, MTD and VLMC almost fail to predict the substates while HISCMiner is highly robust against the noise. In case of the noisy dataset, the test data for MTD and VLMC contains many states which do not appear in the training data due to the nature of the data preprocessing. This affects the accuracy of MTD and VLMC significantly. On the other hand, when the item space I is small as #3-3-2-10, the errors of MTD and VLMC decrease since the number of states in both test and training data decreases. However, the error of HISCMiner increases under this condition, because the noise items generated from the small item space accidentally form spurious substates, and these substates disturb the prediction of HISCMiner. In this regard, HISCMiner is suitable to the data having the background of the large item space.

The computation time required by HISCMiner to model the data under #3-3-7-5000 setting was 9min20sec. When the model order ℓ is increased to 4, it becomes to 18min21sec. When the size of the substate $|S|$ is 4, it is 28min29sec. It remains at 13min45sec under the size of the noise item $|N| = 14$. In theory, the computation time exponentially increases along these parameters. However, the sensitivity to $|N|$ is relatively low, because the random noise does not significantly generate any major substates. For the size of the data $|D|$, the computational complexity is $O(|D|)$ since the Apriori is the main algorithm in this framework. In comparison with the other methods, the computation times under #3-3-0-5000 setting were 6sec, 0.1sec and 245min17sec by HISCMiner, VLMC and MTD respectively. Thus, the HISCMiner is practically sufficient comparing with the other methods.

3.3 Experiment on real world data

The practical performance of our HISCMiner has been assessed through the application to the Move and the Turn datasets of the Pioneer-1 Mobile Robot monitored by 36 sensors and provided at UCI KDD archive [12]. The Move dataset contains an experimental time series data of 6130 time steps where the Pioneer-1 moved on a straight line. The Turn dataset contains a time series data of 2325 time steps where the Pioneer-1 turned in a place. Each row of these data represents many sensor outputs at a time step where some outputs are missing due to their data driven operations. Hence these are transaction time series. Because many sensor outputs are numeric values, the $[s_{min}, s_{max}]$ output range of each numeric sensor s in the data is discretized into *disc* levels as $[l_i^s, u_i^s]$ $(i = 1, ..., disc, l_1^s = s_{min}, u_{disc}^s = s_{max})$ in equi-depth manner, and it is represented as an item $< s : [l_i^s, u_i^s] >$. The first 3/4 time series is used for the training and the rest for the testing.

Similarly to the evaluation on the synthetic data, the ordinary error index is not applicable to measure the performance of HISCMiner. Thus, we use *Precision* and *Recall* under our own error definition. Given a discretized transaction time series $D = X_1...X_n$ and its substate time series $S = S_1...S_n$ predicted by HISCMiner. For two sensor outputs $< s : v_t > \in X_t \in D$ and $< s' : v'_t > \in S_t \in S$ where v is a (discretized) sensor output value, their error $e(< s : v_t >, < s' : v'_t >)$ is defined as follows. When they are numeric and $s = s'$,

$$e(< s : v_t >, < s' : v'_t >) = \left(\frac{x_t - (u_i^s + l_i^s)/2}{s_{max}^s - s_{min}^s} \right)^2,$$

where x_t is the original numeric output of s and $< s' : v'_t > = < s : [l_i^s, u_i^s]_t > \in S_t$. When they are categorical, $s = s'$ and $v_t = v'_t$, $e(< s : v_t >, < s' : v'_t >) = 0$. When they are categorical, $s = s'$ but $v_t \neq v'_t$, $e(< s : v_t >, < s' : v'_t >) = 1$. Otherwise, $e(< s : v_t >, < s' : v'_t >) = 0$. Then, the *Precision* and the *Recall* to take into account the quantitative prediction errors at a time step t is given by

$$Precision(X_t, S_t) = 1 - \sqrt{\frac{\sum_{<s:v_t> \in X_t, <s':v'_t> \in S_t} e(< s : v_t >, < s' : v'_t >)}{|S_t|}},$$

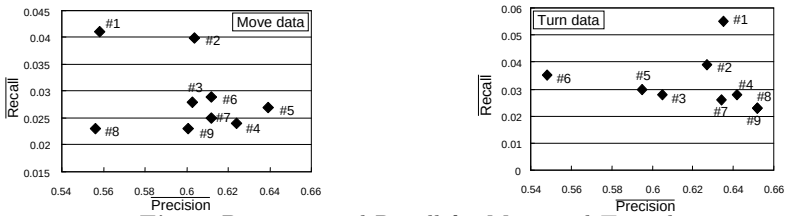


Fig. 2. Precision and Recall for Move and Turn data.

$$Recall(X_t, S_t) = 1 - \sqrt{\frac{\sum_{<s:v_t>\in X_t, <s':v'_t>\in S_t} e(< s : v_t >, < s' : v'_t >)}{|X_t|}}.$$

These are averaged over $t = 1, ..., n$ as $\overline{Precision(X_t, S_t)}$ and $\overline{Recall(X_t, S_t)}$.

For each combination of *minsup* and *disc* shown in Table 2, we conducted the experiments of modeling and prediction. When the small *disc* is used, the number of the frequent substate subsequences is large since many substate transitions are summed up into the small number of the transitions. To save the memory usage of the trie during the search of HISCMiner under the condition, the relatively high *minsup* values were used. Fig. 2 shows the averaged Precision-Recall plot for the two datasets. The points distant from the origin show better performance. Since each real transaction X_t contains many noise items, the Recall is quite small comparing with the Precision. In addition, since the Move and Turn datasets are sampled from very nonstationary operations of the Pioneer-1 Robot, the characteristics of the former 3/4 training data and these of the rest data have some discrepancy. This effect is considered to reduce the prediction precision to some extent. Roughly speaking, the low granularity of the discretization such as the case #1 provides high recall, since the items in S_t easily match to the items in X_t under the less item space I. On the other hand, the moderate granularity of the discretization such as #4 and #7 derives higher precision. $disc = 30 \sim 50$ seems to efficiently capture the substate change of the robot.

4 Discussion

Based on the consequences obtained in the former section, HISCMiner is considered to have a superior ability to model the system dynamics observed by the transaction time series. However, the interpretability of the model is another very important issue. Because the performance of the model interpretability is a quite subjective matter, we limit our discussion to just demonstrate an example on the model interpretation. In the aforementioned example of Pioneer-1 Mobile Robot, most of the substate transition rules contained in the trie of the HISC model trivially represent the static situations where the robot keeps its stable states. However, some substate transition rules clearly indicates the significant behaviors of the robot. For example, the following substate transition rule captured from the Move data shows the specific action of the robot.

$\{0 :< S : [193, 251] >, 0 :< RV : [-88.7, 25.3] >, 0 :< LV : [-86.1, 25.3] >, 0 :< TV : [-87 : 4, 21.5] >\}$
$\{1 :< S : [193, 251] >\}\{2 :< S : [193, 251] >\},$

where S, RV, LV and TV is abbreviation of SONAR-3, R-WHEEL-VEL, L-WHEEL-VEL, and TRANS-VEL, respectively. SONAR-3 is the forward depth

sensed by a sonar. R-WHEEL-VEL and L-WHEEL-VEL are the velocities of right and left wheels. TRANS-VEL is the translational velocity of the robot. This rule indicates that, if SONAR-3 sensed some obstacle within a [193-251]mm distance in the previous two time steps, the velocity of right wheel, left wheel and the translational velocity become within [-88.7,25.3]mm/sec, [-86.1,25.3]mm/sec and [-87:4,21.5]mm/sec. As the maximum moving back velocity of this robot is -300mm/sec, this rule shows that the robot almost stops and/or moves back slowly when it is close to an object in the front.

5 Conclusions

In this paper, we proposed a novel framework called HIgh-order Substate Chain (HISC) modeling and its tool named HISCMiner for the modeling and prediction. The main advantage of our HISCMiner is to derive dynamics of a large size system or a large size sensing information in tractable manner. This issue has not been addressed in the past study of data mining and machine learning. Another advantage of our framework is to obtain interpretable rules on the system dynamics which helps to obtain the knowledge on the objective system. An important issue remained in the HISC modeling is the online modeling and its updating for data streams. This issue can be explored by using some frequent pattern mining technique for data streams [10]. The importance of the proposed framework is expected to increase along the development of the ubiquitous sensing technologies.

References

1. Sun, Z., Miller, R., Bebis, G., DiMeo, D.: A real-time precrash vehicle detection system. Proc. of the IEEE Workshop on Applications of Computer Vision. (2002)
2. Beaudin, J., Intille, S., Tapia, E.M.: Lessons learned using ubiquitous sensors for data collection in real homes. Extended Abstracts of the 6th Conf. on Human Factors in Computing Systems. (2004)
3. Albert, R.Z., Barabasi, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics, 74(1). (2002)
4. Agrawal, R., Srikant, R.: Mining sequential patterns. Proc. of the 11th Int. Conf. on Data Engineering. (1995)
5. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1(3). (1997)
6. MacDonald, I.L., Zucchini, W.: Hidden Markov and Other Models for Discrete-valued Time series. Chapman & Hall/CRC, New York. (1997)
7. Berchtold, A., Raftery, A.E.: The mixture transition distribution model for high-order markov chains and non-gaussian time series. Statistical Science, 17(3). (2002)
8. Bejerano, G., Yona, G.: Variations on probabilistic suffix trees - a new tool for statistical modeling and prediction of protein families. Bioinformatics, 17(1). (2001)
9. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proc. of 20th Int. Conf. on Very Large Data Bases. (1994)
10. Giannella, C., Han, J., Pei, J., Yan, X., and Yu, P.S.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Next Generation Data Mining, MIT Press. (2003)
11. CMU: (http://lib.stat.cmu.edu/general)
12. UCI-KDD-Data-Repository: (http://kdd.ics.uci.edu/databases/pioneer/)

Online Prediction of Clustered Streams

Pedro Pereira Rodrigues and João Gama
{prodrigues,jgama}@liacc.up.pt

LIACC-NIAAD - University of Porto
Rua de Ceuta, 118 - 6 andar
4050-190 Porto, Portugal

Abstract. This paper presents a real-time system for online prediction in large sensor networks, with two components. The goal of our system is to predict the value of each sensor with a given horizon. Due to the large number of sensors, we first cluster them using an online clustering system able to aggregate streaming sensor variables with high correlation. Afterwards, associated with each cluster, we train a neural-network-based predictive model. The clustering system uses a top-down divisive strategy, with the leaves being the resulting clusters, which may be aggregated if the system detects changes in the actual correlation structure. The neural networks are continuously trained with data from the corresponding streams. The system operates online, with incremental monitoring of clusters' diameters and incremental neural network training. Whenever a cluster is split the offspring clusters inherit the parent neural network, starting to fit a different copy. Experiments in the context of online prediction of large electrical sensor networks support our approach, presenting capacity to learn predictive models from clusters' centroids and improving the quality with online training.

1 Introduction

In recent real-world applications, data flows continuously from a *data stream* at high speed, producing examples over time, usually one at a time. Traditional models cannot adapt to the high speed arrival of new examples [3]. This way, algorithms have been developed that aim to process data in *real-time*. These algorithms should be capable of, at each given moment, supply a compact data description and process each example in constant time and memory [1]. Most of the work in incremental clustering of data streams has been concentrated on example clustering rather than variable clustering. Moreover, variable clustering is usually considered a batch offline procedure. Thus, incremental variable clustering isn't too surveyed yet, so we may find a lot of possibilities for contributions in this area of research. Online learning is one of the most powerful and commonly used techniques for training large layered networks and has been used successfully in many real-world applications [13].
The main objective of this work is to present a real-time system for online prediction in large sensor networks, where each variable is a time series and each new example that is fed to the system is the value of an observation of all time

series in a particular time step. There are two components in our system. The first one is an online clustering algorithm able to aggregate variables that exhibit high correlation in the previous recent period. The second component is a set of neural networks, each one being associated with one cluster.
In the next section, a review over incremental clustering analysis for data streams and online learning with neural networks is performed. Section 3 presents the complete system, extending the clustering system with predictive models at the leaves. Experimental evaluation on artificial and real data is presented on section 4, supporting the quality of the system. We finalize the exposition with section 5, where concluding remarks and future work are presented.

2 Related Work

Incremental clustering methods have been developed, which can cope with data stream analysis. An overview of clustering analysis can be found in [10]. Apart from other good examples, *COBWEB* [4], *BIRCH* [15], and *CURE* [6] are well-known incremental methods. Unfortunately, incremental methods to perform clustering on variables are hard to find. A recent development is *ODAC* [12], which we will use on our system, and is detailedly described in section 3.
The objective application of our system is to build predictive models for a large number of load demand sensors, in order to perform online load forecasting. Load forecasting has become in recent years one of the major areas of research in electrical engineering, and several data mining techniques have been applied to fulfill this quest, from simpler time series analysis, like ARIMA models [2], to more complex approaches such as neural networks. Moreover, neural networks have been widely used in the scope of time series forecasting. In [7] the authors present a review of systems and applications of neural networks to load forecasting, both hourly load and profile prediction.
Concerning short-term forecasting, most neural network approaches consider previously observed values in recent time (hours, days, weeks) very important to the forecasting procedure [7,11], acknowledging the cyclic behavior of load demand, detecting daily, weekly and even yearly periods. This has in fact been the basic structure to the input vectors of neural networks in load forecasting. In some approaches, the temperature values are also considered as relevant features for the prediction [14], as they are highly correlated to power consumption.
Unfortunately, load forecast has always been based in batch procedures, with offline training and adaptation of neural networks. Nowadays, the burst of data sensors in electrical networks turned it nearly impossible to consistently adapt the network with all the data available. Thus, online methods that treat electrical data as streams of time series have become even more relevant to the field.
Another reason to consider online learning is that, with the advent of data streams, the assumption that examples are generated at random according to some stationary probability distribution is being disregarded. This way, new methods are being proposed to deal with changes on the concept of the distribution producing the examples, that is, *concept drift* [5].

3 System Description

The goal of the system is to gather a predictive model for all the variables in the system with a horizon forecasting. Data from sensors arrives in real time. Suppose that at time stamp t_i we receive an example. The goal of the system is to predict the value of each variable in time stamp t_{i+k}.
Given the expected high dimensionality of the problem, a design which would gather one model per variable is not possible. Thus, the system fits one predictive model per cluster, leaves of the clustering structure gathered online with a hierarchical clustering algorithm, considering that clustered variables should behave with high correlation. The system predicts all the variables independently.
The ODAC *(Online Divisive-Agglomerative Clustering)* system is a variable clustering algorithm that constructs a hierarchical tree-shaped structure of clusters using a top-down strategy. The leaves are the resulting clusters, with a set of variables at each leaf. The union of the leaves is the complete set of variables. The intersection of leaves is the empty set. The system encloses an incremental distance measure and executes procedures for expansion and aggregation of the tree-based structure, based on the diameters of the clusters. The main setting of our system is the monitoring of existing clusters' diameters. In a divisive hierarchical structure of clusters, considering stationary data streams, the overall intra-cluster dissimilarity should decrease with each split. For each existing cluster, the system finds the two variables defining the diameter of that cluster. If a given heuristic condition is met on this diameter, the system splits the cluster and assigns each of the chosen variables to one of the new clusters, becoming this the *pivot* variable for that cluster. Afterwards, all remaining variables on the old cluster are assigned to the new cluster which has the closest pivot. New leaves start new statistics, assuming that only forthcoming information will be useful to decide whether or not this cluster should be split. This feature increases the system's ability to cope with changing concepts as, later on, a test is performed such that if the diameters of the children leaves approach the parent's diameter, then the previously taken decision may no longer reflect the structure of data, so the system re-aggregates the leaves on the parent node, restarting statistics. Along with the clustering structure definition, a set of predictive models is kept at the leaves, one per leaf, to allow online learning and real time predictions of all the variables. Overall, the system produces real-time predictions of all variables, with online training of the neural networks and online monitoring of the clustering structure.

3.1 Growing the Hierarchy

The main procedure of the ODAC system is to grow a tree-shaped structure that represents the hierarchy of the clusters present in the data. It uses Pearson's correlation coefficient between time series as *similarity* measure. The *sufficient statistics* needed to compute the correlation are easily updated at each time step. In ODAC, the dissimilarity between variables a and b is given by an appropriate

metric, the *Rooted Normalized One-Minus-Correlation* given by

$$rnomc(a,b) = \sqrt{\frac{1 - corr(a,b)}{2}} \tag{1}$$

with range $[0,1]$. We consider the cluster's *diameter* to be the highest dissimilarity between two time series belonging to the same cluster, or the variable variance in the case of clusters with single variables.
In the ODAC system, each example is processed only once. The system incrementally updates, at each new example arrival, the sufficient statistics needed to compute the dissimilarity matrix, enabling its application to clustering of data streams. One problem that usually arises with this sort of models is the definition of a minimum number of observations necessary to assure convergence. The chosen approach is to apply techniques based on the Hoeffding bound to solve this problem. The Hoeffding bound has the advantage of being independent of the probability distribution generating the observations [8], stating that after n independent observations of a real-valued random variable r with range R, and with confidence $1-\delta$, the true mean of r is at least $\overline{r} - \epsilon$, where $\overline{r}$ is the observed mean of the samples and

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{2}$$

As each leaf is fed with a different number of examples, each leaf C_k will possess a different value for ϵ, designated ϵ_k. Let $d(a,b)$ be the heuristic measure used to choose the pair of time series representing the diameter (real value of the distance measure), and $D_k = \{(x_i, x_j) \mid x_i, x_j \in C_k, i < j\}$ be the set of pairs of variables included in a specific leaf C_k. After seeing n samples at the leaf, let $(x_1, y_1) \in \{(x,y) \in D_k \mid d(x,y) \geq d(a,b), \forall (a,b) \in D_k\}$ be the pair of variables with maximum dissimilarity within the cluster C_k, $D'_k = D_k \backslash \{(x_1, y_1)\}$ and $(x_2, y_2) \in \{(x,y) \in D'_k \mid d(x,y) \geq d(a,b), \forall (a,b) \in D'_k\}$, $d_1 = d(x_1, y_1)$ and $d_2 = d(x_2, y_2)$. Let $\Delta d = d_1 - d_2$ be a new random variable, the difference between the observed values. Applying the Hoeffding bound to Δd, if $\Delta d > \epsilon_k$, we can confidently say that, with probability $1-\delta$, the difference between d_1 and d_2 is larger than zero, and select (x_1, y_1) as the pair of variables representing the diameter of the cluster. That is,

$$d_1 - d_2 > \epsilon_k \Rightarrow diam(C_k) = d_1 \tag{3}$$

With this rule, the ODAC system will only split a cluster when the true diameter of the cluster is known with statistical confidence given by the Hoeffding bound. This rule triggers the moment when the leaf has been fed with enough examples to support the decision.
A time series is obviously not a random variable, so we have decided to model the time series first-order differences in order to reduce the negative effect of autocorrelation on the Hoeffding bound, preventing larger errors. Moreover, the missing values can be easily treated with a zero value, considering that, when unknown, the time series is constant.

To distinguish between the cases where the cluster has many variables nearly equidistant and the cases where there are two or more highly dissimilar variables, we introduce a parameter to the system, τ. At any time, if $\tau > \epsilon_k$ the system applies the tests, assuming the leaf has been fed with enough examples, hence it should consider the highest distance to be the real diameter.To prevent the hierarchy from growing unnecessarily, we define a third criterion that has to be met to perform the splitting. The splitting criterion should reflect some relation among the distances between variables of the cluster. Given this fact, we can impose a cluster to be split if it includes a high difference between $(d_1 - \overline{d})$ and $(\overline{d} - d_0)$, where d_0 stands for the minimum distance between variables belonging to the cluster and $\overline{d}$ is the average of all distances in the cluster. In our approach, we relate the expression with the global difference $d_1 - d_0$. Our heuristic is the following: for a given cluster C_k, we choose to split this leaf if:

$$(d_1 - d_0) \left| d_1 + d_0 - 2\overline{d} \right| > \epsilon_k \tag{4}$$

When a split point is reported, the pivots are variables x_1 and x_2 where $d_1 = d(x_1, x_2)$, and the system assigns each of the remaining variables of the old cluster to the cluster which has the closest pivot.

3.2 Adaptation of the Clustering Structure to New Concepts

Considering stationary data streams, the clusters' diameters should decrease with each split. If diameters increase, then probably the structure hass changed, and a new concept is arising. The heuristic that is adopted in this work is the analysis of diameters. This way, no computation is needed between the variables of the two siblings. For each given leaf C_k, we shall test the diameters of C_k, C_k's sibling (C_s) and C_k's parent (C_j), assuming that the sum of the children diameters should not be as large as two times the diameter of the parent. We define a new random variable $\Delta a = 2 \cdot diam(C_j) - (diam(C_k) + diam(C_s))$. Applying the Hoeffding bound to this random variable, if $\Delta a > \epsilon_j$ then the condition is met, so the splitting decision is still a good approach. Given this, we choose to aggregate on C_j if

$$2 \cdot diam(C_j) - (diam(C_k) + diam(C_s)) < \epsilon_j \tag{5}$$

supported by the confidence given by the parent's consumed data. The resulting leaf starts new computations and a concept drift is detected.

3.3 Online Predictions

At each time stamp t_i there are two actions: one is to make a prediction for time stamp t_{i+k}; the other is to back-propagate the error between the prediction done at time stamp t_{i-k} and the value observed at current time stamp. The prediction is made with only the available data at the current time. Nevertheless, we decided to first back-propagate the error, in order to improve predictions

as soon as possible. At time stamp t_{i-k} we did a prediction for the current time t_i. To back-propagate the error, we first compute the difference between the observed values in time t_i and t_{i-1} and propagate the error only one time through the network, achieving an online training of the predictive model. To make a prediction at time t_{i+k}, the system uses previous real data as input to the predictive model. The clustering structure is used here as missing data handler. All the variables in the same cluster are highly correlated, so, they have similar gradient. Since we model the first-order differences for the clustering procedure, when a given time series expresses missing data, this data is replaced by applying to its last known value the average variation of similar time series.
From the user's point of view, there is no need to predict time series that do not behave accordingly to what is expected. Given this, our system only starts fitting a predictive model if and when a cluster expresses good intra-cluster correlation. If it does not, the previously seen value is used as prediction to that particular time series. Let μ_k be the mean intra-cluster correlation of cluster k and σ_k the standard deviation of those correlations. The heuristic used to detect good clusters is to fit a predictive model if

$$\mu_k - \sigma_k > \epsilon_k \tag{6}$$

with ϵ_k given by the Hoeffding bound for that leaf. This heuristic supports the notion that most of the variables in the cluster express positive correlation with each other. In case of clusters with single variables, the variance of the time series is used to compare with the Hoeffding bound, hence, not building predictive models for irrelevant time series.
The predictive models are feed-forward neural networks, with ten inputs, four hidden neurons with *tahn* activation function and one *linear* output, with the input vector to the neural network of time series t at time stamp k being the previously values of t at time stamp k minus $\{1, 2, 3, 4\}$ hours and k minus $\{7, 14\}$ days. We use the iRprop training algorithm [9] to learn the neural networks, reducing the parameter sensitivity of our system. The modularity of our design allows the predictive models to be, not only dynamic and easily changed, but also heterogeneous along different clusters. However, for simplicity and first evaluation of the system, we will only consider homogeneous models for the next hour load forecasting.
To enable ODAC with predictive capacity, it was mandatory to consider *buffering* the input, so the networks could ask for past values of respective variables and the system could store future predictions until real values appear in the data stream. Figure 1 sketches the overview of the buffering procedure executed every time a single example arrives at the system. The size of the buffer is directly proportional to the granularity of the incoming data, since it must include data from t_i minus two weeks till t_i plus one hour. As often considered [11,14,7], four cyclic variables were also included, for hourly and weekly periods (*sin* and *cos*). Every time a cluster is split, the offspring clusters inherit the parent's network, starting to model a different network separately. This way, a specification of the model is considered along the specification of the clustering structure. When an

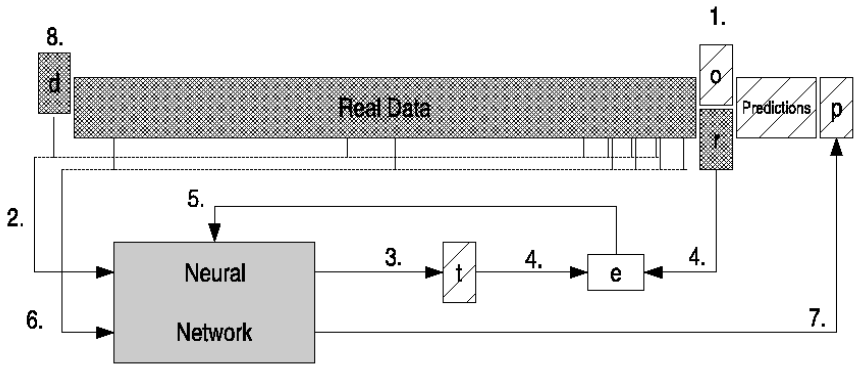


Fig. 1. Buffered Online Predictions: 1. new real data arrives (r) at time stamp i, substituting previously made prediction (o); 2. define the input vector to predict time stamp i; 3. execute prediction (t) for time stamp i; 4. compute error using predicted (t) and real (r) values; 5. back-propagate the error one single time; 6. define input vector to predict time stamp i plus one hour; 7. execute prediction of next hour (p); 8. discard oldest real data (d).

aggregation of clusters occurs, due to clustering concept drift, the new cluster starts a new neural network defined as a weighted merge of the descendants networks. Another possibility is to reset the internal weights to a previously learned set of weights, which may have been proved to be robust and general enough to start the training of one particular set of variables.

4 Experimental Evaluation

Experimental evaluation of the system is two-folded. First, the clustering strategy applied in the system must be validated. This was already performed in [12], using real data from medical sensors and electrical network power demand data. Afterwords, preliminary evaluation of the predictive strategy is performed and results are presented. Using electrical demand sensor data, the clustering the system resulted in a large tree, supporting the data streams requirements of speed and memory, identifying quality clusters with good intra-cluster correlation. Figure 2 plots the structure gathered in this experiment. Overall, experimental results show that the system possesses competitive performance when compared with batch clustering analysis, evolving and adapting in the presence of concept drift.

4.1 Predictive Task

There are two major concepts our system is supposed to address. On one hand, considering the expected high correlation between time series of the same cluster, the system should be able to fit a predictive model that represents the whole

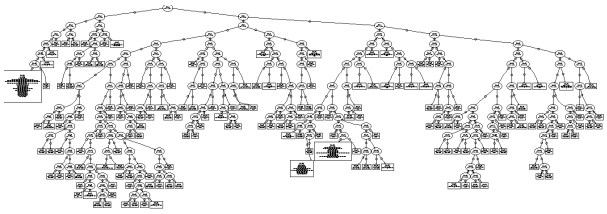


Fig. 2. ODAC Clustering Structure

cluster, training with the cluster's centroid. On the other hand, it is expectable that online learning should produce better adaptation to new examples, comparing to predictive models trained with past examples and no adaptation to current data. These are the aims of the following experiences.

Clusters as Representatives For this first experience, we try to discover if our system can build predictive models for each cluster with a certain degree of quality. We use 482 variables of the entire electrical demand data set and build the clustering structure with several months of real data. For each cluster which possess good intra-cluster correlation, the system learns a predictive model using the centroid of the corresponding time series for the whole past data and tests them in the following week, for each variable individually. We evaluate predictions using the following well-known cost function:

$$MAPE = \sum_{i=1}^{n} \frac{|(\hat{y}_i - y_i)/y_i|}{n} \tag{7}$$

where y_i is the real value of variable y at time stamp i, and $\hat{y}_i$ is the corresponding predicted value. Preliminary results on electrical power demand current data support some of the motivations for our work. Fitted models resulted in predictions with MAPE evaluation values under 10%. Figure 3 presents an example of predictions made by a neural network trained with the centroid of the cluster.

Online Learning In these experiences, we are interested on inspecting the advantages of online learning, comparing online training results with the predictions made by batch models. In Figures 3 and 4, plots of the resulting predictions and the evolution of the MAPE error are shown, for two example time series from different clusters. In Figure 3, the incremental neural network starts giving poorer predictions compared with the static network, as this still maintains the underlying dynamics of the time series. However, after some time, the incremental learning combines its efficiency with accuracy, diminishing the error below the static model. This behavior becomes even clearer in Figure 4, where the incremental system maintains an error descent through time, outperforming the static predictions. After the first week, for all non-null variables, the average improvement achieved by online training is about 5%.

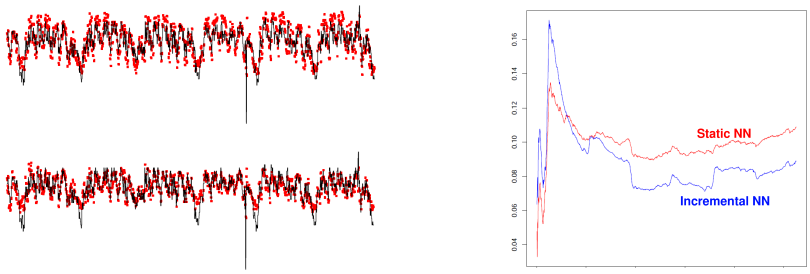


Fig. 3. Real values (line) and system's predictions (squares) using a static (top) and online trained (bottom) neural network. On the right side we can stress the quality of online training, comparing the MAPE evolution of both networks on the same series.

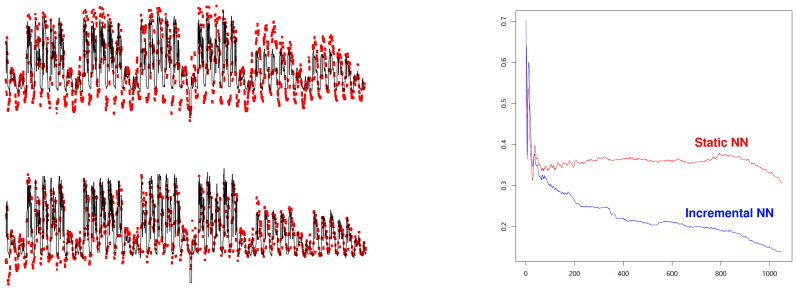


Fig. 4. Plot of another example time series. The MAPE evolution On the right side clearly shows the gain in performance achieved with online training of the predictive models, since the static network fails to adapt to changes in the future.

5 Conclusions

This paper introduces a system that gathers a predictive model for a large number of data variables with an horizon forecasting, incrementally constructing a hierarchy of clusters and fitting a predictive model for each leaf. The main setting of the clustering system is the monitoring of existing clusters' diameters. The main setting of the predictive strategy is the buffered online prediction of each individual variable, based on a neural network trained with clustered variables. The examples are processed as they arrive, using a single scan over the data. Experimental results show that the system is able to fit predictive models using the centroids of the cluster they are associated to. Moreover, applying incremental learning, using the online strategy developed in this work, seems to outperform predictions made with static predictive models. Future work will focus on the definition of global evaluation strategy and the inspection of other online learning algorithms or architectures such as recurrent neural networks. We believe further work on the learning task will improve the quality of neural network basic accuracy.

Acknowledgments

The authors wish to thank the Plurianual support attributed to LIACC, and the participation of projects RETINAE (PRIME/IDEIA/70/00078) and ALES II (POSI/EIA/55340/2004).

References

1. Daniel Barbará. Requirements for Clustering Data Streams. *SIGKDD Explorations*, 3(2):23–27, 2002.
2. George Box and Gwilym Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.
3. Pedro Domingos and Geoff Hulten. Mining High-speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2000.
4. Douglas H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2(2):139–172, 1987.
5. Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with Drift Detection. In *Advances in Artificial Intelligence - SBIA 2004*, (LNCS 3171), pages 286–295. Springer Verlag, 2004.
6. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 73–84. ACM Press, 1998.
7. H. S. Hippert, C. E. Pedreira, and R. C. Souza. Neural Networks for Short-Term Load Forecasting: A Review and Evaluation. *IEEE Transactions on Power Systems*, 16(1):44–55, 2001.
8. W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
9. Christian Igel and Michael Hüsken. Improving the Rprop Learning Algorithm. In *Proceedings of the Second International ICSC Symposium on Neural Computation*, pages 115–121, Berlin, 2000. ICSC Academic Press.
10. A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323, 1999.
11. Pedro Rodrigues and João Gama. Forecast adaptation to charge transfers. *WSEAS Transactions on Circuits and Systems*, 3(8):1693–1699, 2004.
12. Pedro Pereira Rodrigues, João Gama, and João Pedro Pedroso. ODAC: Hierarchical Clustering of Time Series Data Streams. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 499–503, Bethesda, Maryland, 2006. SIAM.
13. David Saad, editor. *On-line Learning in Neural Networks*. Cambridge University Press, 1998.
14. P. J. Santos, A. G. Martins, and A. J. Pires. Designing the input vector to ann-based models for short-term load forecast in electricity distribution systems. Technical Report 4, INESC Coimbra, 2005.
15. Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM Press, 1996.

Model Averaging via Penalized Regression for Tracking Concept Drift

Kyupil Yeon[1], Moon Sup Song[1], Yongdai Kim[1] and Cheolwoo Park[2]

[1] Department of Statistics, Seoul University, Seoul, 151-742, South Korea
[2] Department of Statistics, University of Georgia, Athens, GA 30602-1952, USA

Abstract. We propose a new model averaging algorithm for tracking concept drift in data streams. The final predictive ensemble model takes the form of a weighted average of base models. The combining weights are determined by a ridge regression with the constraints such that the weights are nonnegative and sum to one. The proposed algorithm is motivated from a new measure of concept drift, which is defined as the angle between the estimated and the optimal weight vector obtained under no concept drift. It is shown that the ridge tuning parameter plays a crucial role of forcing the proposed algorithm to adapt to concept drift. Main findings are (i) the algorithm can achieve the optimal weights in the case of no concept drift if the tuning parameter is sufficiently large, and (ii) the angle is monotonically increasing as the tuning parameter decreases. These imply that if the tuning parameter is well-controlled, the algorithm can produce weights which reflect the degree of concept drift measured by the angle. Under various simulation settings, it is shown that the proposed algorithm can track the concept drift better than other existing ensemble methods.

1 Introduction

In this paper, we are interested in supervised learning problems under concept drift in a stream of data batches. Let $\{D_j, j = 1, 2, \cdots\}$ be a sequence of data batches, each consisting of input-output pairs. Suppose that observations in D_j are random samples from unknown distribution $F_j(\mathbf{x}, y)$, where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^p$ is an input vector and $y \in \mathcal{Y}$ is a response. The objective of learning is to construct a predictive model for future instances based on the data sets $D_1, \ldots, D_m$ at each time point m. One problematic issue in this setting of sequential data stream is *concept drift* which states that the underlying target concept changes over time abruptly or gradually. In a probabilistic sense, it can be characterized as the change of underlying distributions such that $F_j(\mathbf{x}, y) \neq F_{j+1}(\mathbf{x}, y)$.

The concept drift phenomenon, which emerges naturally in a data stream, makes the learning process complex because the predictive model constructed on the past data is no longer consistent with the new examples. Therefore, in order to cope with a drifting concept, a learning algorithm should be equipped with a mechanism to adapt to the concept drift. For example, a learning algorithm should adapt quickly to concept drift whatever the types of changes are like,

discriminate a real concept drift from noises, be able to cope with recurring concepts, be simple and fast in order to deal with continuous data streams.

We suggest an ensemble algorithm which can achieve these goals. The final predictive model produced by the proposed algorithm is the form of weighted average of base models which are independently constructed from each previous data batch. The model averaging process is performed via a penalized regression which is designed to be able to track concept drift effectively. The organization of the paper is as follows. The next section reviews some literatures on concept drift tracking algorithms. Section 3 describes the proposed algorithm with its motivation and properties. In Sect. 4, we show some simulation results with artificial data sets. Finally, we conclude in Sect. 5.

2 Learning under Concept Drift

Concept drift tracking algorithms can be roughly divided into two categories. One is a single learner based tracker which aims to select previous examples or data batches most relevant to learning the current concept. We refer to it as *data combining approach* in this paper. The other is an ensemble approach which is mainly related to formulating and restructuring an ensemble of several base learners. In this approach, the most important thing is how to combine base models to cope with various types of concept drift. Therefore, we refer to it as *model-combining approach*.

2.1 Data combining approach

A conventional way of coping with concept drift problem is to use a time window of fixed size over data streams. In other words, the most recent M data batches are used to construct a predictive model. However, there is a dilemma in this approach. A large size of the time window is preferable when there is no concept drift, but cannot adapt quickly to concept drift. On the contrary, a small size of the time window can track a new concept quickly, but is not preferable when the concept is stable or recurrent. Hence the optimal size of the time window cannot be set in general unless the type and degree of concept drift are known in advance.

To adaptively determine the size of time window, Widmer and Kubat (1996)'s FLORA systems use a WAH (Window Adjustment Heuristic) approach. Klinkenberg and Joachims (2000) proposed an algorithm for tracking concept drift with SVM (Support Vector Machine) in classification problem. Indicating some problems in fixed window size and heuristic adjustment approaches, they suggested an algorithm to adjust adaptively the size of time window over data batches. Assuming a non-recurring concept drift, which means that newer examples are always more important than older ones, they selected the window size so that the estimated generalization error on the most recent data batch is minimized. It is important to note that $\xi\alpha$-estimates (Joachims, 2000), a special property of SVM, is utilized to get an estimate of the generalization error efficiently, which

makes the algorithm not heuristic but theoretically well-founded. In a sense, however, it restricts the algorithm to use only SVM as a learner.

A time window can be generalized to contain inconsecutive data batches selectively. This scheme is more relevant especially in the case of recurrent concepts. Klinkenberg (2004) suggested a local batch selection scheme which is performed by utilizing $\xi\alpha$-estimates for SVM. He also considered global/local batch weighting schemes and compared various strategies of selection and weighting.

2.2 Model combining approach

Model-combining approach is an ensemble strategy for learning in changing environments. In a static setting, ensemble methods such as bagging (Breiman, 1996), boosting (Freund and Schapire, 1997) and stacking (Wolpert, 1992) are generally known to produce a better ensemble prediction model than one best model. In the wake of success in a static setting, ensemble methods have been effectively applied to a concept drift setting. Compared with data combining approaches, the ensemble method may be more suitable to data streams in that it does not need to keep all the previous data sets, instead it may just retain the base models.

Kuncheva (2004) grouped ensemble methods for tracking concept drift into five categories and summarized each representative algorithm. Street and Kim (2001) proposed the streaming ensemble algorithm (SEA) which applies a simple unweighted voting like bagging to combine M base models into the final prediction model. Therefore, the final ensemble model for predicting future instances is given by

$$\hat{f}_E(\mathbf{x}) = \frac{1}{M}\sum_{j=1}^{M} \hat{f}_j^{(m)}(\mathbf{x}),$$

where $\{\hat{f}_1^{(m)}(\cdot), \cdots, \hat{f}_M^{(m)}(\cdot)\}$ denote M ensemble members formulated at the current time point m. When a new data batch D_{m+1} comes in, if the ensemble size falls short of M then the new base model $\hat{f}_{m+1}(\cdot)$ is added into a new ensemble, otherwise, the ensemble is updated by replacing the least useful member with $\hat{f}_{m+1}(\cdot)$ which is assumed to reflex the current concept well. The substitution is based on a quality measure which measures the usefulness of each member of $\{\hat{f}_1^{(m)}(\cdot), \cdots, \hat{f}_M^{(m)}(\cdot)\}$ and the new base model $\hat{f}_{m+1}(\cdot)$. The quality measure they suggested gives more score to the classifiers which correctly classify points on which the ensemble is nearly undecided. In this way the contribution of each base model to the ensemble's performance is evaluated and then the least useful base model is replaced with the new one.

Another interesting way of averaging base models is a weighted average. Wang et al. (2003) suggested an accuracy-based weighted ensemble algorithm, that is, the weights of base models are determined so that on the most recent data batch the more accurate a base model is, the more weight it is endowed with. For a fixed ensemble size M, let $E_m = \{\hat{f}_1^{(m)}(\cdot), \cdots, \hat{f}_M^{(m)}(\cdot)\}$ be the ensemble

set of base models to be combined at the current time point m, and the output of a base model be a class probability rather than the class label. The ensemble prediction model is given by a weighted average of the ensemble members

$$\hat{f}_E(\mathbf{x}) = \sum_{j=1}^{M} w_j \hat{f}_j^{(m)}(\mathbf{x}), \tag{1}$$

with the constraints $\sum_{j=1}^{M} w_j = 1 \quad \text{and} \quad w_j \geq 0 \quad \text{for all } j = 1, \ldots, M$. The weights $\{w_1, \ldots, w_M\}$ are determined so that they are proportional to each corresponding base model's accuracy on the most recent data batch D_m. When the new data batch D_{m+1} comes in, the ensemble update is performed so that the new ensemble set consists of M top ranked base models in terms of their accuracy on D_{m+1} among $\{\hat{f}_1^{(m)}(\cdot), \cdots, \hat{f}_M^{(m)}(\cdot)\}$ and $\hat{f}_{m+1}(\cdot)$. The expected accuracy of base models is measured by their predictive accuracy such as MSE (Mean Squared Error) on the most recent data batch. Note that the resulting weight can be 0 if the MSE is below a certain threshold and then several obsolete base models can be totally discarded at the same time in case of abrupt concept drift. However, the accuracy-based weights are not optimal since the accuracy of each base model is marginally obtained without any consideration of other base models' effects on the final ensemble model.

In contrast to the above two average-type model combining approaches, an additional meta learning can be adopted as a combiner to aggregate the outputs of base models. Typically, classical regression method can be used as a combiner. Chu et al. (2004) used ordinary logistic regression as a combiner with an integrated outlier elimination procedure based on clustering of the likelihood values evaluated at each observation. By explicitly filtering out some noise-like observations in the most recent data batch and constructing an ensemble predictive model based on remaining examples, they developed an algorithm not only adaptive to concept drift but also robust to label noises.

The framework of a regression based combination of base models is as follows. Suppose that we are given a sequence of data batches $D_1, \ldots, D_m$. First, we construct base models $\hat{f}_1(\mathbf{x}), \ldots, \hat{f}_m(\mathbf{x})$ independently from each $D_1, \ldots, D_m$ respectively. As a base learner we can use any types of learning algorithm. If we are considering a classification problem, then each $\hat{f}_j(\mathbf{x})$ could be a confidence output such as class probability or class label itself. Second, we obtain meta learning set from the outputs of base models for examples in the most recent data batch $D_m = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$. That is, for each $\mathbf{x}_i$ $(i = 1, \ldots, n)$, we obtain the outputs of base models denoted by

$$\hat{\mathbf{f}}(\mathbf{x}_i)^T = (\hat{f}_1(\mathbf{x}_i), \ldots, \hat{f}_{m-1}(\mathbf{x}_i), \hat{f}_m^{(-i)}(\mathbf{x}_i)),$$

where $f_m^{(-i)}(\mathbf{x}_i)$ represents a leave-one-out estimate of $f_m(\mathbf{x}_i)$. This is for avoiding over-fitting problem since observations in D_m is used for constructing both $\hat{f}_m$ and meta learning set. Practically it is good to use v-fold cross-validation (e.g. 5 or 10-folds CV) instead of leave-one-out. We will refer to this meta learning set

as a *level-1 set* for reference to stacked generalization (Wolpert, 1992). To sum up, the level-1 set for training a regression combiner is given as follows.

$$X = \begin{pmatrix} \hat{f}_1(\mathbf{x}_1) & \cdots & \hat{f}_{m-1}(\mathbf{x}_1) & \hat{f}_m^{(-1)}(\mathbf{x}_1) \\ \hat{f}_1(\mathbf{x}_2) & \cdots & \hat{f}_{m-1}(\mathbf{x}_2) & \hat{f}_m^{(-2)}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{f}_1(\mathbf{x}_n) & \cdots & \hat{f}_{m-1}(\mathbf{x}_n) & \hat{f}_m^{(-n)}(\mathbf{x}_n) \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \ . \tag{2}$$

Finally, this level-1 set is used for training a regression combiner, which produces a final predictive model

$$\hat{f}_E(\mathbf{x}) = \sum_{j=1}^{m} \hat{w}_j \hat{f}_j(\mathbf{x}) \ . \tag{3}$$

The regression based combining described above is just to regard base models as explanatory variables and fit a regression model to a level-1 set derived using examples in the most recent data set. This method does not need a time window over data batches or ensemble size to be optimally set. A fixed ensemble size can be implemented for a practical reason but it is not a crucial point in concept drift tracking. The concept drift tracking is accomplished by a regression combiner's ability of allocating adaptive weights to base models. Moreover it has only to retain base models instead of data sets themselves, which is a highly required desideratum of an algorithm for data stream mining.

In spite of many advantages, this regression based combining method has also some drawbacks. First, the so called multi-collinearity problem causes the estimation unstable especially in the case of no or gradual concept drift. Second, the resulting weights may have negative values. The negative weights do not indicate the fact that the corresponding base models are negatively correlated with the target concept because the negative signs can be occurred if another highly correlated base models exist. Third, in this algorithm there is no mechanism for discriminating the real concept drift and noises. When there are considerable noises, the resulting weights of this algorithm cannot reflect the appropriateness of each base model.

From the above discussion, the regression combiner described in this section does not seem to produce optimal weights for tracking concept drift. We will propose another regression based combiner in the next section and show that the proposed method can be a good alternative.

3 Model Averaging via Penalized Regression

We propose a regression combiner which produces the final ensemble model (3) for some weights $\{(\hat{w}_1, \ldots, \hat{w}_m) | \sum \hat{w}_j = 1, \hat{w}_j \geq 0\}$, which are to be estimated via a penalized regression. Due to the constraints of nonnegativity and sum-to-one, the algorithm can be regarded as a model averaging procedure. The level-1 set for training a penalized regression is given as in (2). In Fig. 1 are presented

Method 1 (MC.Ridge1+)

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \quad \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} w_j \hat{f}_j(\mathbf{x}_i) \right)^2 + \lambda \sum_{j=1}^{m} w_j^2, \tag{4}$$

$$\text{subject to} \qquad \sum_{j=1}^{m} w_j = 1,\ w_j \geq 0\ . \tag{5}$$

Method 2 (MC.Lse1+)

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \quad \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} w_j \hat{f}_j(\mathbf{x}_i) \right)^2, \quad \text{subject to} \quad \sum_{j=1}^{m} w_j = 1,\ w_j \geq 0\ .$$

Fig. 1. Two regression combiners with constraints.

two regression combiners with the constraints that weights are nonnegative and sum to one. The main proposed Method 1 will be referred to as "MC.Ridge1+" indicating *M*odel *C*ombining via *Ridge* regression with constraints of sum-to-*1* and nonnegativity(+). For the purpose of identifying the effect of ridge penalty, we also consider Method 2 in Fig.1 which is formulated without the ridge penalty and hence referred to as "MC.Lse1+" to indicate *M*odel *C*ombining via *L*east *S*quare *E*stimation with constraints of sum-to-*1* and nonnegativity(+).

At first glance, the proposed algorithm seems to mix simply the weighted average combiner and penalized regression combiner in an attempt to solve the multi-collinearity problem with ridge penalty and to enhance the interpretability with the constraints of nonnegativity and sum-to-one. However, there is another important motivation which is more directly related to concept drift tracking.

3.1 Motivation and Properties

Let us consider the case that there is no concept drift at all. That is, the target concept remains the same and the underlying distribution does not vary over time. In this case, the optimal aggregation is just a simple average of base models if each base model is unbiased. This is shown in Theorem 1.

Theorem 1. *If each base model is unbiased and independent, then the optimal ensemble weight under no concept drift is given by* $\mathbf{w}^* = \{1/m, \ldots, 1/m\}$.

Theorem 1 indicates that all base models are equally important in the case of no concept drift. It motivates us to derive a measure of concept drift. If an ensemble algorithm adapts to concept drift nicely, then its weights $\hat{\mathbf{w}}$ obtained in the case of no concept drift would be very close to the optimal $\mathbf{w}^* = (1/m, \ldots, 1/m)$. However, if a considerable concept drift occurs, the derived weights $\hat{\mathbf{w}}$ would be far from $\mathbf{w}^*$. Since we are considering the weight space confined only on the

hyperplane $\mathbf{w}^T \cdot \mathbf{1} = 1$ with $\mathbf{w} \geq 0$, the degree of departure of $\hat{\mathbf{w}}$ from $\mathbf{w}^*$ can be captured through the angle between the two m-dimensional vectors. Therefore, we define this angle as a measure of concept drift.

Definition 1 (A measure of concept drift). *For* $\mathbf{w}^* = (1/m, \ldots, 1/m)$ *and* $\hat{\mathbf{w}}$ *obtained by any combiner, we define the* degree of concept drift *as the angle between the two vectors given by*

$$\eta(\mathbf{w}^*, \hat{\mathbf{w}}) = \cos^{-1}\left(\frac{|\langle \mathbf{w}^*, \hat{\mathbf{w}} \rangle|}{||\mathbf{w}^*||\,||\hat{\mathbf{w}}||}\right), \tag{6}$$

where $\langle \cdot\,, \cdot \rangle$ *denotes the inner product of two vectors in* $\mathbb{R}^m$ *and* $||\cdot||$ *is the length of a vector defined as* $||\boldsymbol{a}|| = \langle \boldsymbol{a}, \boldsymbol{a} \rangle^{1/2}$ *for any* $\boldsymbol{a} \in \mathbb{R}^m$.

Definition 1 is useful for designing a combiner for tracking concept drift. To be more specific, we need an algorithm which is able to produce a weight vector almost the same as $\mathbf{w}^*$ when there is no or quite gradual concept drift. On the other hand, in the case of more considerable concept drift, a combiner should provide a weight $\hat{\mathbf{w}}$ which makes the angle $\eta(\mathbf{w}^*, \hat{\mathbf{w}})$ large. Note that the algorithms reviewed in Sect. 2 such as simple average, marginally weighted average based on base models' accuracy, and regression combiner do not satisfy this property. But, it will be verified that the proposed algorithm MC.Ridge1+ has such property. This is the key point of the proposed method. We describe this main property through the following two related theorems. These theorems explain why the proposed algorithm can be a reasonable concept drift tracker regardless of the types of changes.

Theorem 2. *In the case of no concept drift, the proposed algorithm* MC.Ridge1+ *produces a weight which converges to the optimal weight* $\mathbf{w}^*$ *as* $\lambda \to \infty$.

Theorem 3. *For the* $\hat{\mathbf{w}}$ *obtained by* MC.Ridge1+, $\eta(\mathbf{w}^*, \hat{\mathbf{w}})$ *in (6) is monotonically decreasing as* λ *increases.*

Theorem 2 and 3 distinguish the proposed algorithm from other previous combining methods described in Sect. 2. As we can see in Theorem 2, the optimal weight can be obtained when the ridge parameter λ in (4) is estimated to be sufficiently large. This indicates that the proposed algorithm can produce a good ensemble prediction model in the case of no or gradual concept drift by estimating the ridge parameter λ in a data-adaptive way at each time. The estimation of λ is usually done by cross-validation which will turn to be satisfactory by simulation results in the next section.

The implication of Theorem 3 is more attractive. The importance of the ridge parameter in tracking concept drift is clearly conceived. The parameter λ controls the adaptivity of the algorithm to concept drift. A larger λ makes the algorithm generate a weight vector $\hat{\mathbf{w}}$ for which $\eta(\mathbf{w}^*, \hat{\mathbf{w}})$ gets smaller corresponding to no or gradual concept drift. On the other hand, a relatively smaller λ induces the algorithm to produce $\hat{\mathbf{w}}$ such that $\eta(\mathbf{w}^*, \hat{\mathbf{w}})$ gets larger which corresponds

to abrupt concept drift. This means that the proposed algorithm can output a weight vector $\hat{\mathbf{w}}$ on which the degree of concept drift is properly reflected. Of course, this is controlled by a well-estimated λ.

3.2 Computation of the Algorithm

Kim and Kim (2004) proposed the gradient LASSO algorithm to provide an approximated solution for the generalized LASSO models. This new computational algorithm employs the coordinatewise gradient descent (CGD) method. They showed that their algorithm is computationally much simpler and stabler than the QP based algorithm, and easily applicable to large dimensional data. They also derived the convergence rate of the algorithm, which does not depend on the dimension of inputs.

We can solve the problem described in Fig. 1 by using the CGD algorithm. Let us denote the objective function and the constraint as follows.

$$C(\mathbf{w}) = \sum_{i=1}^{n} l(y_i, \mathbf{x}_i^T \mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}, \tag{7}$$

$$S = \{\mathbf{w} : \mathbf{w}^T \mathbf{1} = 1,\ \mathbf{w} \geq 0\}, \tag{8}$$

where $l(y, \mathbf{x}^T \mathbf{w})$ is a loss function which can be either L_2 or cross-entropy loss.

The main idea of the CGD algorithm is to find $\hat{\mathbf{w}}$ sequentially as follows. For a given $\mathbf{v} \in S$ and $\alpha \in [0, 1]$, let $\mathbf{w}[\alpha, \mathbf{v}] = \mathbf{w} + \alpha(\mathbf{v} - \mathbf{w})$. Suppose that $\mathbf{w}$ is the current solution. Then, the CGD algorithm searches a direction vector $\mathbf{v}$ such that $C(\mathbf{w}[\alpha, \mathbf{v}])$ decreases most rapidly and update $\mathbf{w}$ to $\mathbf{w}[\alpha, \mathbf{v}]$. Note that $\mathbf{w}[\alpha, \mathbf{v}]$ is still in S. The Taylor expansion implies

$$C(\mathbf{w}[\alpha, \mathbf{v}]) \approx C(\mathbf{w}) + \alpha \langle \nabla C(\mathbf{w}), \mathbf{v} - \mathbf{w} \rangle,$$

where $\nabla C(\mathbf{w}) = \left(\frac{\partial C(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial C(\mathbf{w})}{\partial w_m} \right)^T$. It can be easily shown that

$$\min_{\mathbf{v} \in S} \langle \nabla C(\mathbf{w}), \mathbf{v} \rangle \ = \ \min \left\{ \frac{\partial C(\mathbf{w})}{\partial w_j}, \cdots, \frac{\partial C(\mathbf{w})}{\partial w_m} \right\} \ .$$

Hence, the desired direction is a j^*-th coordinate unit vector $\mathbf{u}_{j^*}$ such that

$$j^* = \underset{j \in \{1, \ldots, m\}}{\arg\min} \left\{ \frac{\partial C(\mathbf{w})}{\partial w_j} \right\}$$

$$\mathbf{u}_{j^*} = (0, \ldots, 0, \underset{\substack{\uparrow \\ j^*-th}}{1}, 0, \ldots, 0) \ .$$

The optimization procedure is summarized in Fig. 2. The convergence of the algorithm is guaranteed by Theorem 1 in Kim and Kim (2004).

Let $C(\mathbf{w}) = (\mathbf{y} - X\mathbf{w})^T(\mathbf{y} - X\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$.
For $k = 1, 2, 3, \ldots,$

① $\nabla C(\mathbf{w}^k) = \left(\frac{\partial C(\mathbf{w}^k)}{\partial w_1}, \cdots, \frac{\partial C(\mathbf{w}^k)}{\partial w_m}\right)^T$, $\mathbf{w}^1 = \mathbf{w}^*$: initial value.
② $j^* = \underset{j\in\{1,\cdots,m\}}{\arg\min} \langle \nabla C(\mathbf{w}^k), \mathbf{u}_j \rangle$, $\mathbf{u}_j$: j-th unit vector.
③ $\hat{\alpha} = \underset{0\le\alpha\le 1}{\arg\min}\ C\left((1-\alpha)\mathbf{w}^k + \alpha\mathbf{u}_{j^*}\right)$
④ $\mathbf{w}^{k+1} = (1-\hat{\alpha})\mathbf{w}^k + \hat{\alpha}\mathbf{u}_{j^*}$

Iterate ① $\sim$ ④ until the solution converges.

Fig. 2. Coordinatewise gradient descent method for MC.Ridge1+

4 Simulation Results

We present some simulation results for 2-class classification problems. It has to be mentioned that although we also obtained some good results on regression problems, they are not presented because of limited space.

We compare four model-combining algorithms described: model-combining via simple average (MC.Avg), model-combining via weighted average (MC.WAvg), model-combining via least squares estimation with constraints (MC.Lse1+), and model-combining via ridge regression with constraints (MC.Ridge1+). For reference, we add one more model, denoted as MC.1, which is constructed using only the most recent data batch.

4.1 2-class classification : moving hyperplane data

This data is a synthetic concept-drifting streaming data in which concept drift is simulated with a moving hyperplane. Hyperplanes have been used to simulate time-changing concepts by many authors (Hulten et al., 2001; Wang et al., 2003; Yang et al., 2005).

We considered 10-dimensional hyperplane such as $\sum_{i=1}^{10} \alpha_i X_i = \alpha_0$. Examples are generated independently from Uniform $[0,1]^{10}$ and they are assigned a class label y as follows:

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{10} \alpha_i X_i > \alpha_0, \\ 0 & \text{otherwise.} \end{cases}$$

By setting $\alpha_0 = \frac{1}{2}\sum_{i=1}^{10} \alpha_i$, we made the hyperplane divide the unit hypercube into two parts of the same volume. Thus, the number of examples in each class is made to be balanced. We made up 30 data batches. Each data batch has 100 examples. Concept drift between data batches is simulated by changing

the magnitude of coefficients of the hyperplane. We initialized $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{10})$ with $(1, \ldots, 1)$. For each $\delta = 0.01,\ 0.02,\ 0.03$, gradual concept drift is incorporated by randomly adding or subtracting δ from each component of the one step previous $\boldsymbol{\alpha}$. Abrupt concept drift is also substantiated with $\delta = 1$ at a certain time-point. Furthermore, we observed the effect of noises by inserting 10% or 20% class label noises at each data batch.

To compare the prediction accuracy of the algorithms we also constructed test data sets. Each test set has 1,000 examples which are generated without noises from the same concept as the corresponding training data batch. As a base learner we used decision trees constructed independently from each data batch utilizing a pruning procedure with 10-folds cross-validation. Moreover, the probabilistic outputs of base models are combined by the combiners.

Figure 3 represents the results of the simulation in the case that ensemble size is 20 batches. In the figures, accuracy denotes the averaged value of 20 test accuracies obtained through 20 repeated simulations. We can notice that MC.Ridge1+ performs best in most cases except the case of no concept drift. Therefore, the proposed algorithm is verified as very effective in concept drift tracking and also robust to the class noises.

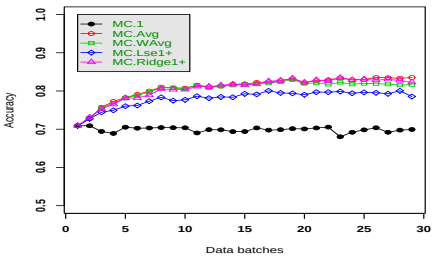
4.2 2-class classification : SEA data

This simulated data was used in Street and Kim (2001). There are 3 predictor variables X_1, X_2, X_3 generated from Uniform [0,10] independently. The target variable y is a class label determined by the first two predictor variables X_1, X_2 such that

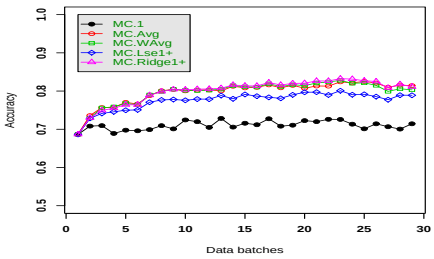
$$y = \begin{cases} 1 & \text{if} \quad X_1 + X_2 \leq \theta, \\ 0 & \text{otherwise.} \end{cases}$$

We can think of θ as an identifier of target concept. To simulate concept drift, we considered 4 concepts specified by $\theta = 8,\ 9,\ 7,\ 9.5$, respectively. We constructed 25 data batches from each concept. That is, $D_1, \ldots, D_{25}$ are generated from concept 1 ($\theta = 8$), $D_{26}, \ldots, D_{50}$ from concept 2 ($\theta = 9$), $D_{51}, \ldots, D_{75}$ from concept 3 ($\theta = 7$), and $D_{76}, \ldots, D_{100}$ from concept 4 ($\theta = 9.5$). Each data batch consists of 500 examples. We inserted about 10% class noises into each data batch. $D_1, \ldots, D_{100}$ are regarded as a sequential data stream. Hence, concept change occurs 3 times at $t = 25$, 50, and 75. At each time point $t = 1, \ldots, 99$, we constructed a prediction model using $D_1, \ldots, D_t$ and then predicted examples in a test set which was generated from the same concept as D_{t+1} without class noises. Each test data set has 2,500 examples. As in the previous simulation with moving hyperplane data, all model-combining algorithms use the same base models. That is, decision trees are constructed independently using only each data batch and the probabilistic outputs of these trees are used by all the combiners.

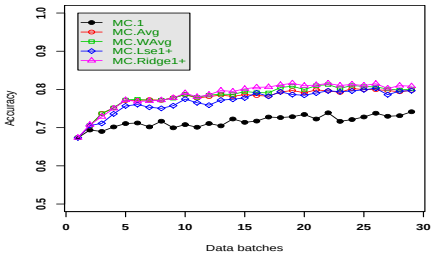
Figure 4 represents the simulation result. From this, we can confirm that the proposed algorithm MC.Ridge1+ recovers the prediction accuracy very fast



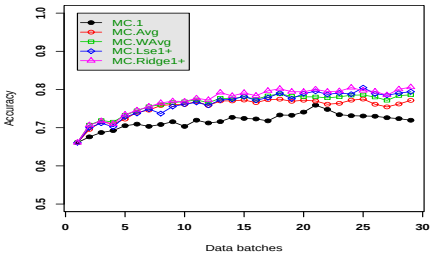
(a) No concept drift



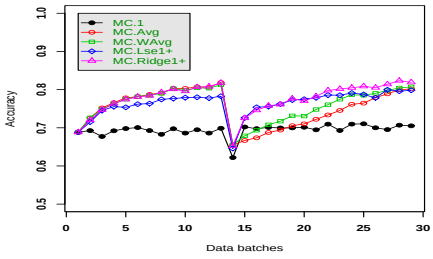
(b) Gradual drift ($\delta = 0.1$)



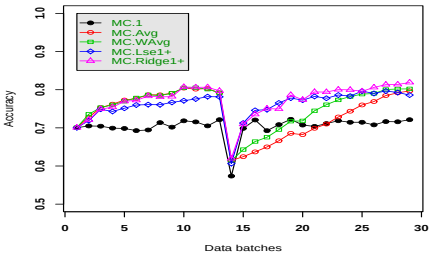
(c) Gradual drift ($\delta = 0.2$)



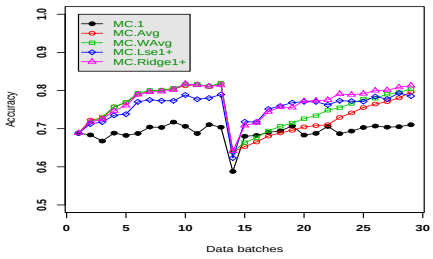
(d) Gradual drift ($\delta = 0.3$)



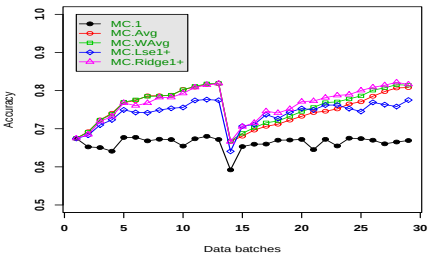
(e) Abrupt concept drift



(f) Abrupt+Gradual drift

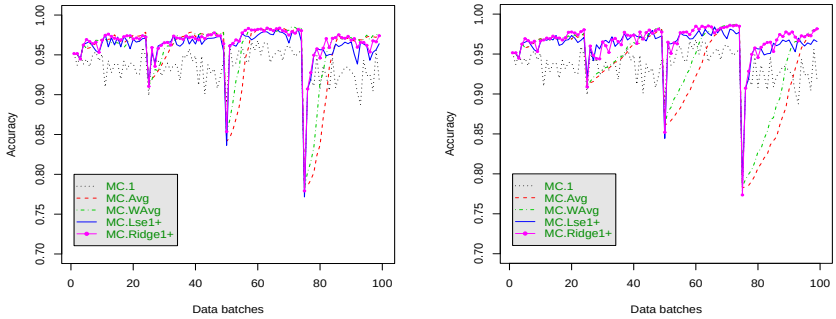


(g) Abrupt+Gradual drift (noise 10%)



(h) Abrupt+Gradual drift (noise 20%)

Fig. 3. Concept drift tracking in Moving-hyperplane data



(a) Ensemble size = 10 (b) Ensemble size = 25

Fig. 4. Concept drift tracking in SEA data

after a sudden dropping at the concept drift point and keeps its accuracy higher than others. One thing worth mentioning is that MC.Ridge1+ shows fast adaptivity regardless of the ensemble size, but average-type combiners MC.Avg and MC.WAvg are ineffective especially in case of large ensemble size.

5 Conclusions

In this paper, we proposed a new model-averaging method for tracking concept drift in data streams. The motivation on the proposed algorithm stems from a newly defined measure of concept drift. We showed that the new measure of concept drift, defined as an angle between two weight vectors, can be a good guidance for designing an ensemble tracker of concept drift. The proposed algorithm tackles the concept drift problem by constructing an ensemble prediction model, which is formulated as a weighted average of base models. The core of the algorithm is the method of determining the combining weights, which is performed by a ridge regression with the constraints such that weights are non-negative and sum to one. It was shown under simulation settings for 2-class classification problems that the proposed algorithm can track concept drift better than other existing methods under both abrupt and gradual changes. Some good advantages of the proposed algorithm is that it can select (i.e., exactly 0 weights for some base models) or weight base models automatically according to the types of concept drift, and does not need to tune an optimal ensemble size, and does not depend on an intuitive restructuring of the current ensemble in order to adapt to concept drift, and has only to retain base models instead of all the previous data sets.

References

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

Chu, F., Y. Wang, and C. Zaniolo (2004). Mining noisy data streams via a discriminative model. In *Discovery Science*, pp. 47–59.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science 55*(1), 119–139.

Hulten, G., L. Spencer, and P. Domingos (2001). Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, pp. 97–106. ACM Press.

Joachims, T. (2000). Estimating the generalization performance of a SVM efficiently. In P. Langley (Ed.), *Proceedings of ICML-00, 17th International Conference on Machine Learning*, Stanford, US, pp. 431–438. Morgan Kaufmann Publishers, San Francisco, US.

Kim, Y. and J. Kim (2004). Gradient lasso for feature selection. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, NY, USA, pp. 473–480. ACM Press.

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

Klinkenberg, R. and T. Joachims (2000). Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pp. 487–494. Morgan Kaufmann.

Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In *Multiple Classifier Systems*, pp. 1–15.

Street, W. N. and Y. S. Kim (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 377–382. ACM Press.

Wang, H., W. Fan, P. S. Yu, and J. Han (2003). Mining concept drifting data streams using ensemble classifiers. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 226–235. ACM Press.

Widmer, G. and M. Kubat (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning 23*(2), 69–101.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks 5*(2), 241–259.

Yang, Y., X. Wu, and X. Zhu (2005). Proactive-reactive prediction for data streams. Technical Report CS-05-03, Computer Science, University of Vermont.

Incremental training of Markov mixture models

Andreas Kakoliris and Konstantinos Blekas

Department of Computer Science, University of Ioannina, 45110 Ioannina, Greece
E-mail: {akakolir, kblekas}@cs.uoi.gr

Abstract. This paper presents an incremental approach for training a Markov mixture model to a set of sequences of discrete states. Starting from a single Markov model that captures the background information, at each step a new component is added to the mixture in order to improve the data fit. This is done by making at first an exploration of a relevant parametric space to initialize the new component, based on an extension of the k-means algorithm. Then, by performing a two-stage scheme of the EM algorithm, the new component is optimally incorporated to the body of the current mixture. To assess the effectiveness of the proposed method, we have conducted experiments with several data sets and we make a performance comparison with the classical mixture model.

1 Introduction

Sequential data analysis is an important research area with a wide range of applications, such as web log mining, bioinformatics, speech recognition, robotics, natural language processing and many others. Since clustering can be seen as a fundamental tool in understanding and exploring a data set, several attempts have been made on the task of clustering sequential data of discrete states [1–4]. In model-based clustering approaches a flexible and powerful scheme used is through mixture models [5]. It is assumed that each cluster is described by a generative model and the aim of clustering is to find an optimal set of such models in order to best fit the data. Markov models [6] provide an efficient method for modeling sequential data. In most of the these approaches, the EM algorithm [7] is used for estimating the parameters of the Markov mixture models. Since the EM algorithm has the drawback to be dependent on the initial values of the mixture parameters, several methods have been introduced to reduce this effect. In [3] for example, a noisy-marginal scheme is proposed by perturbing the parameters of a single model to obtain K copies of it. An alternative approach is presented in [1], where an agglomerative clustering technique is applied together with a suitable distance function for sequences, in order to initialize the K parametric models of the mixture. Many efforts have been made recently to address visualization capabilities of clustering approaches using Markov models [3, 8–10]. In this spirit, the behavior of the sequences within clusters can be displayed and an explanatory analysis for the dynamics of data can be provided.

In this paper we propose an incremental approach for training Markov mixture models. Borrowing strength from recent advances on mixture models [11,

12], our method performs a systematic exploration of the parameter space and simultaneously tries to eliminate the dependence of the EM algorithm on the initialization. The method starts with a single Markov model that fits all sequences, and sequentially adds new components to the mixture following three major steps. At first we initialize the new inserted Markov model by searching over a parametric likelihood space. The latter is specified by a set of candidate models that have been constructed through the use of an adaptation of the classical k-means algorithm for treating sequential data. This is the initialization step. Then, we perform a partial EM scheme allowing the adjustment of only the new model parameters. Finally, the new component is optimally incorporated to the current mixture by normally applying the EM algorithm and thus best fitting the new mixture model with the data. The procedure stops when reaching a number of K components. We have tested our training method on a suite of artificial and real benchmarks taking into account a variety of cases with excellent results. During the experiments we have evaluated the proposed scheme in terms of its capability to fit the data and measure its robustness. Comparative results have been also obtained with the classical Markov mixture model under two schemes for initialization.

In section 2 we give the basic scheme of the Markov mixture models, while section 3 describes the proposed approach for incremental training. In section 4 we present experimental results and finally, in section 5 we give some concluded remarks.

2 Markov mixture models

Consider a dataset $X = \{X_1, \ldots, X_N\}$, where each data point $X_i = (X_{il})_{l=1}^{L_i}$ is a sequence of length L_i observed states. We further assume that each state takes values from a discrete alphabet of M symbols, i.e. $X_{il} \in \{1, \ldots, M\}$. The clustering problem is to find K disjoint subsets of X, called clusters, containing sequences with common properties. In this study we consider that every cluster corresponds to a generative model that fits well the observed data that supports.

Mixture models represent an efficient architecture that is particularly suitable for clustering. It assumes that data have been generated from a mixture model with K components according to the following density function

$$f(X_i|\Theta_K) = \sum_{j=1}^{K} \pi_j p(X_i|\theta^j) \ , \tag{1}$$

where $\Theta_K = \{\pi_j, \theta^j\}$ denotes the set of the mixture parameters. In particular, the parameters $\pi_j = P(j)$ determine the prior probabilities of the K components satisfying that $\sum_{j=1}^{K} \pi_j = 1$. Moreover, every component has a probability distribution function $p(X_i|\theta^j)$, whose parameters θ^j are unknown. A natural way for modeling sequential data is through the first-order Markov model, defined by the initial states probabilities $\theta^j_{0m} = P(X_{i1} = m)$, as well as the transition probabilities $\theta^j_{nm} = P(X_{i,l+1} = m|X_{il} = n)$ from a state n to another state m,

$n, m = 1, \ldots, M$. Thus, each model parameter θ^j is a stochastic matrix with a set of $M+1$ rows (multinomial distributions), holding that $\sum_{m=1}^{M} \theta_{nm}^j = 1$, $\forall n = 0, \ldots, M$. The density function for the jth component is then written as

$$p(X_i|\theta^j) = \theta_{0,X_{i1}}^j \prod_{l=1}^{L_i-1} \theta_{X_{il},X_{i,l+1}}^j = \prod_{m=1}^{M} (\theta_{0m}^j)^{\gamma_i(m)} \prod_{n=1}^{M} \prod_{m=1}^{M} (\theta_{nm}^j)^{\delta_i(n,m)}, \quad (2)$$

where $\gamma_i(m) = \begin{cases} 1 \text{ if } X_{i1} = m \\ 0 \text{ otherwise} \end{cases}$ and $\delta_i(n,m)$ defines the number of transitions from state n to state m in the sequence X_i. Following the Bayes rule, we can then associate every sequence X_i to the cluster j that has the maximum posterior probability value $P(j|X_i) = \frac{\pi_j p(X_i|\theta^j)}{f(X_i|\Theta_K)}$. The clustering problem is then equivalent to estimating the mixture model parameters Θ_K, by maximizing the log-likelihood function arisen from the model. Furthermore, we can introduce non-informative Dirichlet priors of the form $p(\theta_n^j|a_n^j) = \frac{\Gamma(\sum_{m=1}^{M}(a_{nm}^j+1))}{\prod_{m=1}^{M}\Gamma(a_{nm}^j+1)} \prod_{m=1}^{M}(\theta_{nm}^j)^{a_{nm}^j}$, where the parameter a_n^j is a M-vector with components $a_{nm}^j > 0$. The derived *maximum a-posteriori* (MAP) log-likelihood function is then given by

$$L(X|\Theta_K) = \sum_{i=1}^{N} \log f(X_i|\Theta_K) + \sum_{j=1}^{K} \sum_{n=0}^{M} \log p(\theta_n^j|a_n^j) \ . \quad (3)$$

It must be noted that the Dirichlet parameters a_n^j were common to every component j and set equal to a small proportion (e.g. 10%) of the corresponding maximum likelihood (ML) estimated multinomial parameter values of the single Markov model that fits the data set X (using relative frequencies of states). The latter from now on it will be referred to as "single ML-estimated Markov model".

The EM algorithm [7] is an efficient framework for estimating the mixture model parameters. It requires the computation of the conditional expectation values z_{ij} (posterior probabilities) of the hidden variables during the E-step $z_{ij}^{(t)} = \frac{\pi_j^{(t)} p(X_i|\theta^{j\,(t)})}{\sum_{j'=1}^{K} \pi_{j'}^{(t)} p(X_i|\theta^{j'\,(t)})}$, while at the M-step the maximization of the log-likelihood function of the complete dataset is performed. This leads to the following updated equations for the mixture model parameters:

$$\pi_j^{(t+1)} = \frac{\sum_{i=1}^{N} z_{ij}^{(t)}}{N} \ , \ \theta_{nm}^{j\ (t+1)} = \begin{cases} \dfrac{\sum_{i=1}^{N} z_{ij}^{(t)} \gamma_i(m) + a_{0m}^j}{\sum_{i=1}^{N} z_{ij}^{(t)} + \sum_{m'=0}^{M} a_{0m'}^j} & \text{, if } n = 0 \\ \dfrac{\sum_{i=1}^{N} z_{ij}^{(t)} \delta_i(n,m) + a_{nm}^j}{\sum_{i=1}^{N} z_{ij}^{(t)} \sum_{m'=1}^{M} \delta_i(n,m') + \sum_{m'=1}^{M} a_{nm'}^j} & \text{, if } n > 0 \end{cases} \quad (4)$$

The EM algorithm guarantees the convergence of the log-likelihood function to a local maximum satisfying all the constraints of the parameters. However, the great dependence on the initial parameter values may drastically effect its performance [5]. In the next section we present an incremental approach for building a Markov mixture models that eliminates this problem of poor initialization.

3 Incremental mixture training

The proposed method starts with a simple model with one component that comes from the single ML-estimated Markov model of the whole dataset X. At each step a new component is added to the mixture by performing a combined scheme of searching for good initial estimators and for fine local tuning its parameters. It must be noted that a same in nature strategy have been also presented in [11] and [12] for Gaussian mixture models and for discovering patterns in biological sequences, correspondingly.

Lets assume that we have already constructed a k-length mixture model with Θ_k parameters. By inserting a new component, the resulting mixture can take the following form

$$f(X_i|\Theta_k, \pi^*, \theta^*) = (1 - \pi^*) f(X_i|\Theta_k) + \pi^* p(X_i|\theta^*) \ . \tag{5}$$

where $\pi^* \in (0, 1)$ is the prior probability of the new component. The above scheme can be viewed as a two-component mixture model, where the first one captures the current mixture with density function $f(X_i|\Theta_k)$ and the second one the new Markov model that has a density function $p(X_i|\theta^*)$ with an unknown stochastic matrix θ^*.

If we fix the parameters of the old mixture model Θ_k, we can then maximize the resulting log-likelihood function $\mathcal{L}_k$ of the above two-components mixture with respect only to the new model parameters $\{\pi^*, \theta^*\}$:

$$\mathcal{L}_k = \sum_{i=1}^{N} \log\{(1 - \pi^*) f(X_i|\Theta_k) + \pi^* p(X_i|\theta^*)\} + \sum_{n=0}^{M} \log p(\theta_n^*|a_n) \ . \tag{6}$$

In this light, we can apply the EM algorithm for estimating only the parameters of the new model, namely as *partial EM*. This results into obtaining the following update equations: a) at the E-step

$$\zeta_i^{(t)} = \frac{\pi^{*(t)} p(X_i|\theta^{*(t)})}{(1 - \pi^{*(t)}) f(X_i|\Theta_k) + \pi^{*(t)} p(X_i|\theta^{*(t)})} \ , \tag{7}$$

and b) at the M-step

$$\pi^{*(t+1)} = \frac{\sum_{i=1}^{N} \zeta_i^{(t)}}{N},\ \theta^{*}_{nm}{}^{(t+1)} = \begin{cases} \dfrac{\sum_{i=1}^{N} \zeta_i^{(t)} \gamma_i(m) + a_{0m}}{\sum_{i=1}^{N} \zeta_i^{(t)} + \sum_{m'=0}^{M} a_{0m'}} & , \text{if } n = 0 \\ \dfrac{\sum_{i=1}^{N} \zeta_i^{(t)} \delta_i(n,m) + a_{nm}}{\sum_{i=1}^{N} \zeta_i^{(t)} \sum_{m'=1}^{M} \delta_i(n,m') + \sum_{m'=1}^{M} a_{nm'}} & , \text{if } n > 0 \end{cases} \tag{8}$$

The above partial EM steps offer more flexibility to the general scheme and simplifies the estimation problem during the insertion of a new Markov model to the mixture.

At a second stage, the new component can be incorporated to the body of the current mixture and construct a new mixture $f(X_i|\Theta_{k+1})$ with $k+1$ components. Again, the EM algorithm can be used to maximize the log-likelihood function $L(X|\Theta_{k+1})$ in the new parameter space Θ_{k+1}, following Eqs. 4. The mixture parameters are initialized from the solution of the partial EM, i.e. $\pi_{k+1}^{(0)} = \pi^*$, $\pi_j^{(0)} = (1-\pi^*)\pi_j$, $\forall j = 1, \ldots, k$, and $\theta_{k+1}^{(0)} = \theta^*$. This iterative procedure is repeated until the desired order K of the Markov mixture model is reached.

3.1 Initializing new model parameters

From the above analysis, a problem that arises is how to initialize properly the new component parameters during the partial EM scheme. This can be accomplished by establishing a parametric search space through a set of K_m candidate Markov models $\{\phi_j\}_{j=1}^{K_m}$. In particular, we perform one step of the partial EM, after initializing the multinomial parameters of the new model with a candidate Markov model ($\theta^{*(0)} = \phi_j$) and the prior probability π^* with the typical value $\pi^{*(0)} = \frac{1}{k+1}$. Finally, we select the solution that corresponds to the maximum value of the log-likelihood function $\mathcal{L}_k$ (Eq. 6) for initializing the parameters $\{\pi^*, \theta^*\}$.

In our study we have used an extension of the known k-means algorithm to create such a set of candidate models. In the general case, the k-means algorithm aims at finding a partition of K_m disjoint clusters C_j to a set of N objects, so as the overall sum of distances between cluster centers μ_j and objects X_i is minimized. In order to adopt this framework in the case of sequential data we need to make some modifications. At first, a distance function between two sequences X_i and X_k must be provided so as to encapsulate an appropriate measure of dissimilarity between data. For this purpose we have used a symmetrized log-likelihood

distance defined as [1]

$$D(i,k) = \frac{1}{2}\{\log p(X_i|\vartheta_k) + \log p(X_k|\vartheta_i)\} \ , \tag{9}$$

where the parameters ϑ_i denote the single ML-estimated Markov model specified by each sequence X_i. Furthermore, at each step t of the k-means algorithm we re-estimate the new center $\mu_j^{(t+1)}$ of every cluster C_j by finding the *medoid* sequence among the sequences that currently supports, i.e. $\mu_j^{(t+1)} = \arg\min_{X_i \in C_j^{(t)}} \sum_{X_k \in C_j^{(t)}} D(i,k)$. At the end of the algorithm, we correspond a Markov model ϕ_j to every cluster C_j, by finding the single (ML-estimated) Markov model that best fits all sequences associated with this cluster. The above scheme creates a pool of K_m candidate models capable for initializing the parameter θ^* during the partial EM steps. As experimental study has shown, the proposed method is not sensitive to the value of K_m. A small proportion of the population size of sequences N (e.g. 5%) is enough for constructing a rich search space with good initial estimators. Another advantage of the proposed k-means algorithm is that is computationally faster than other distance-based clustering schemes (e.g. hierarchical clustering) that can be alternatively applied using the same distance function (Eq. 9).

3.2 The proposed algorithm

The proposed incremental approach for training a mixture of K Markov models can be summarized in the following algorithmic form.

- Set $\Theta_1 = \{\theta^1, \pi_1 = 1\}$ using the single ML-estimated Markov model from the data set X. Use k-means to provide K_m candidate Markov models ϕ_j.
- for $k = 1 : K-1$
  1. $\forall j = 1, \ldots, K_m$ perform one partial EM step (Eqs.7-8) by setting ${\pi^*}^{(0)} = \frac{1}{k+1}$ and ${\theta^*}^{(0)} = \phi_j$. Select the solution that has the maximum log-likelihood value $\mathcal{L}_k$ (Eq. 6).
  2. Perform partial EM (Eqs.7-8) until convergence and estimate new model parameters $\{\pi^*, \theta^*\}$.
  3. Set $\Theta_{k+1} = \Theta_k \cup \{\pi_{k+1}, \theta_{k+1}\}$, where ${\pi_{k+1}}^{(0)} = \pi^*$, ${\pi_j}^{(0)} = (1-\pi^*)\pi_j$ $\forall j \leq k$, ${\theta^{k+1}}^{(0)} = \theta^*$.
  4. Perform general EM (Eqs.4) to maximize $L(X|\Theta_{k+1})$.

4 Experimental results

Several experiments have been made in an attempt to evaluate the performance of the proposed incremental training approach, namely as IMM. Comparative results have been also obtained using two methods for initializing classical Markov mixture models: a) the RMM, that follows the initialization scheme presented in [3] which creates K noisy copies from the single ML-estimated Markov model,

and b) the KMM, that first applies the k-means algorithm as described previously for discovering K clusters ($K_m = K$), and then initializes every component with the single ML-estimated Markov model of every cluster found. In any case, the prior parameters are initially set as $\pi_j = 1/K$. Since both last methods depends on the initialization, twenty (20) runs of the EM algorithm were performed for each data set. We kept records of the mean value and the standard deviation of the log-likelihood. Also, the proposed IMM model was executed only once for fitting a K-order Markov mixture model to each data set.

Table 1. Percentage of times the correct model was detected by the three methods IMM, KMM and RMM.

# symbols (M)	*mixture model*	# components (K)			
		5	8	10	15
5	*IMM*	100 %	100 %	100 %	100 %
	RMM	80.5 %	67.5 %	50 %	25.5 %
	KMM	56 %	49.5 %	31.5 %	7 %
8	*IMM*	100 %	100 %	100 %	100 %
	RMM	67 %	47.5 %	35.5 %	11.5 %
	KMM	50 %	32 %	19 %	7 %
10	*IMM*	100 %	100 %	100 %	100 %
	RMM	74.5 %	45.5 %	28.5 %	10 %
	KMM	47 %	28.5 %	15.5 %	2.5 %
12	*IMM*	100 %	100 %	100 %	100 %
	RMM	70 %	42 %	26.5 %	9.5 %
	KMM	35 %	25 %	11 %	2.5 %
15	*IMM*	100 %	100 %	100 %	100 %
	RMM	75 %	35.5 %	21 %	6 %
	KMM	52 %	20.5 %	9.5 %	1 %

The first series of experiments was carried out using artificial data to evaluate the robustness of our method. We created sets of artificial sequences by sampling from several K-order Markov mixture models using various values for the alphabet size M. In particular, using five and four different values for the parameters K and M, correspondingly, we created ten (10) different datasets for each pair (M, K). In each dataset $N = 1000$ number of sequences were generated of length between 50 and 100 states ($L_i \in [50, 100]$). Since we were aware of the true model that best fit the experimental datasets, we evaluated each method by calculating the percentage of times that the global maximum log-likelihood value was found. Table 1 summarizes the depicted results. The weakness of both the RMM and KMM approaches in obtaining the global maximum value, is obvious, especially in higher values of K. On the other hand, the proposed IMM approach was able to estimate correctly the true model in all cases.

Another series of experiments with artificial sequences has been made using sets of K randomly selected patterns of equal length 50 from an alphabet of

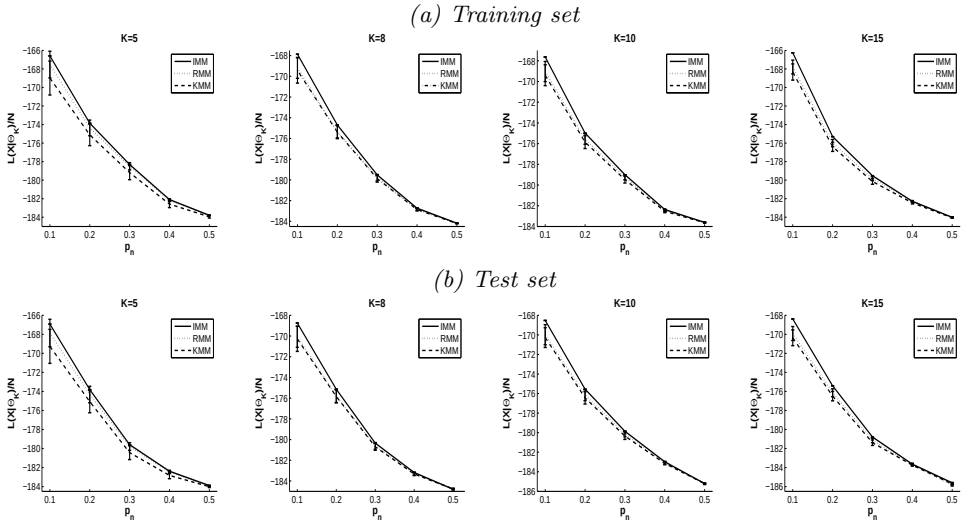


Fig. 1. The log-likelihood values found by the three comparative methods as a function of noise parameter p_n.

M symbols. The data generation mechanism was the following: A pattern is randomly selected at first, and then a noisy copy of it is located at a random position in the sequence. Pattern noise is governed by using a probability p_n for mutation, common to every pattern site. The rest non-pattern sites are filled uniformly from the same alphabet. Using this scheme, two sets of $N = 1000$ sequences of length $L_i \in [50, 100]$ were created; one used for training and another one for testing. As it is clear, this clustering problem is more difficult since the Markov property exists only locally in the sequences and under different levels of noise. Likewise, for each randomly selected pattern family we generated ten (10) different datasets and we evaluated each method in terms of the log-likelihood value found in both training and test sets. Figure 1 illustrates the results obtained with four values of $K = \{5, 8, 10, 15\}$ and five different levels of noise $p_n = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ in the case of $M = 10$ alphabet size. In each diagram, the error bars indicate the standard deviation of the log-likelihood difference between the true model that is known and the model under consideration. Our method was able to achieve a high degree of noise tolerance, since always managed to discover the correct model, even for extremely noisy datasets.

Additional experiments have been performed using the msnbc.com web navigation dataset [3], which is a collection of sequences that corresponds to $M = 17$ page-category views (symbols) of users during twenty-four hour period. Here we have considered only a subset of the total collection containing 4600 sequences of length $L_i \in [40, 100]$. We randomly divided it into two subsets (training / test) of approximately equal size. Figure 2 shows the calculated log-likelihood value per sequence ($L(X|\Theta_K)/N$) as a function of the mixture order K on both

sets. Our method was executed only once until reaching $K = 16$ components. In the case of the RMM and KMM methods we plot the mean value and the standard deviations (error bars) of the log-likelihood over 20 different runs (initializations) of the EM algorithm per each value of K. The proposed method showed an improvement performance with better generalization capabilities on the test set in comparison with the other two approaches. Note that we have repeated this study with different divisions into training and test subsets of this dataset and the results were similar. Finally, in Figure 3 we give an example of the visualization capabilities of clustering sequential data that can be used for identifying user behavior patterns in applications such as web log mining [3, 8]. Each one of the ten images corresponds to a cluster found when applying our method for training a mixture model with $K = 10$ Markov components. By associating every symbol with a unique color, sequences that belong to the same cluster are represented as raws of colored squares.

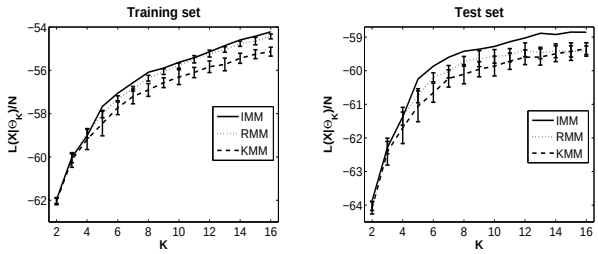


Fig. 2. Application of the three methods to the msnbc.com dataset. The log-likelihood values are calculated for several values of K in the training and the test set.

5 Conclusions

In this paper we have presented an incremental strategy for training Markov mixture models by maximum likelihood on a set of sequences of discrete states. The approach sequentially adds components to a mixture model by performing a combined scheme of the EM algorithm. In order to initialize properly each new component, an efficient parameter search space of Markov models has been constructed. Experiments on a variety of benchmarks have shown the ability of our method to improve the data fit and also demonstrated its generalization capability. The determination of the proper value of K for terminating the incremental procedure can be seen as one of our future studies on this area. Finally, we plan to focus our attention on mixtures of hidden Markov models, since they can be seen as more general probabilistic models for sequential data.

Acknowledgments. This research is partially supported by the EPEAEK II Program.

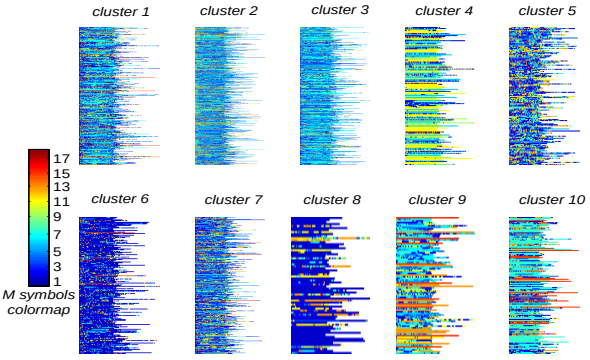


Fig. 3. Visualization of the clustering results ($K = 10$) on the msnbc.com data. Each image represents a cluster of user sessions in a colored raw form.

References

1. P Smyth. Clustering sequences with hidden Markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 648–654. The MIT Press, 1997.
2. G. Ridgeway. Finite discrete markov process clustering. Technical Report MSR-TR-97-24. Microsoft Research, Redmod, WA, 1997.
3. I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Model-based clustering and visualization of navigation patterns on a web site. *Data Mining and Knowledge Discovery*, 7(4):399–424, 2003.
4. M. Bicego, V. Murino, and M. Figueiredo. Similarity-based classification of sequences using hidden Markov models. *Pattern Recognition*, 37:2281–2291, 2004.
5. G.M. McLachlan and D. Peel. *Finite mixture models*. New York: John Wiley & Sons, Inc., 2001.
6. L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.
7. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Statist. Soc. B*, 39:1–38, 1977.
8. E. Manavoglu, D. Pavlov, and C.L. Giles. Probabilistic user behavior models. In *IEEE International Conference on Data Mining (ICDM'03)*, pages 203–210, 2003.
9. A. Ypma and T.M. Heskes. Automatic categorization of web pages and user clustering with mixtures of hidden Markov models. In *WEBKDD 2002 - Mining web data for discovering usage patterns and profiles*, pages 35–49, Berlin, 2003.
10. P. Tino, A. Kaban, and Y. Sun. A generative probabilistic approach to visualizing sets of symbolic sequences. In *ACM SIGKDD - International Conference on Knowledge Discovery and Data Mining - KDD-2004*, pages 701–706, 2004.
11. N. Vlassis and A. Likas. A greedy EM algorithm for Gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87, 2002.
12. K. Blekas, D.I. Fotiadis, and A. Likas. Greedy mixture learning for multiple motif discovering in biological sequences. *Bioinformatics*, 19(5):607–617, 2003.

Improving Prediction Accuracy of an Incremental Algorithm Driven by Error Margins *

José del Campo-Ávila[1], Gonzalo Ramos-Jiménez[1], João Gama[2], and Rafael Morales-Bueno[1]

[1] Departamento de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática. Universidad de Málaga, Spain
{ramos, jcampo, morales}@lcc.uma.es
[2] LIACC - University of Porto, Portugal
jgama@liacc.up.pt

Abstract. Incremental learning is an approach to deal with the classification task when datasets are too large or when new examples can arrive at any time. One possible approach uses concentration bounds (like Chernoff or Hoeffding bounds) to ensure that expansions are done when the number of examples supports the change. Two algorithms that use this approach are VFDT or IADEM. In this paper we extend the IADEM system in two directions: adding the ability to deal with continuous data and including the use of more powerful classification techniques at tree leaves. The proposed system, IADEMc, can incorporate and classify new information as the basic algorithms do, using shorter time per example. Another relevant property of our system is the ability to obtain a performance similar to a standard decision tree algorithm independently of the datasets size.

1 Introduction

Machine learning systems are currently required to deal with large datasets. Moreover, data streams have recently become a new challenge for data mining because of certain features [1] of which infinite data flow is the most notable. Thus, datasets can indefinitely grow at a very high rate and it would be highly desirable to be able to induce concepts from them. Trying to extract knowledge from such numbers of examples can become an inaccessible problem for traditional algorithms such as ID3 [2] or C4.5 [3] due to system requirements. There are other approaches that are less demanding like disk-based learners, such as SLIQ [4] or SPRINT [5], but, although they make more efficient use of memory, they still have some limitations: they require multiple scans of the dataset, they cannot handle infinite data flows because the datasets are needed to be completed before executing, and they also fail when datasets are too large.

Probabilistic representation [6] provides the opportunity of exploring data and keeping the most relevant information from seen examples. Therefore, required memory does not have to depend on the number of examples in the dataset, but on the structure and associated statistics. Some algorithms use this scheme in one way or another, such

* This work has been partially supported by the FPI program and the MOISES-TA project, number TIN2005-08832-C03, of the MEC, Spain.

as ID4 [7], ITI [8] or VFDT [9]. However, the potential of this representation is limited depending on which memory model [10] was used. Considering the characteristics of large datasets and data streams we must say that partial example memory with forgetting mechanisms and no example memory are the models more appropriates because of their memory requirements. Chernoff [11] and Hoeffding [12] bounds have recently been used in Machine Learning area [9, 13–15] in conjunction with probabilistic representation . They have been used to provide statistical evidence in favour of a particular split test, to ensure a minimum number of scans through a sequence database, etc.

Incremental learning methods (including online, successive or sequential methods) have some features that make them very suitable to deal with huge amounts of data. The concept "incremental learning" has been used rather loosely, ranging from very relaxed interpretations (such as the "revolutionary" approach of Michalski [16]) to excessively restrictive ones (such as proposed by Polikar [17]), but two main features describe an incremental learning algorithm: its capability to incorporate new examples into the knowledge base [7] and its capability to evolve this knowledge base from a very basic concept to another more complex one [3].

In this paper we propose some extensions to the incremental algorithm IADEM: the ability to deal with numerical attributes and the capability to apply naive Bayes classifiers in the tree leaves. The paper is organised as follows. The next section describes IADEM and other related works that are the basis for this paper. In Section 3 we introduce the extensions made to IADEM leading to the algorithm IADEMc. The experimental evaluation is presented in Section 4. Finally, in Section 5, we summarise our conclusions and suggest future lines of research.

2 Related Work

In this section we present related work about powerful strategies for predicting and we briefly summarise the basic algorithm that we are extending: IADEM.

2.1 Functional Tree Leaves

The standard algorithms that induce decision trees usually install at each leaf node a constant that minimises a given loss function. In the classification setting, the constant that minimises the 0-1 loss function is the mode of the target attribute of the examples that fall at this leaf. Several authors have studied the use of other functions at tree leaves [18, 19]. One of the earliest works is the Perceptron tree algorithm [20] where leaf nodes may implement a general linear discriminant function. Also Kohavi [18] has presented the naive Bayes tree that uses functional leaves. NBtree is a hybrid algorithm that generates a regular univariate decision tree, but the leaves contain a naive Bayes classifier built from the examples that fall at this node. The approach retains the interpretability of naive Bayes and decision trees, while resulting in classifiers that frequently outperform both constituents, especially in large datasets.

In this paper we explore this idea in the context of incremental learning. As we show in the experimental section, there are strong advantages in the performance of resulting decision models.

2.2 Incremental Algorithm Driven by Error Margins

IADEM [15] is an incremental algorithm with no example memory which knowledge base is represented using decision trees. IADEM receives examples to learn from, but once they are processed they are forgotten. No example is saved, only the relevant information for IADEM is kept using counters stored in the decision tree. Thus, the memory requirements of this algorithm only depend on the size of the decision tree and its associated counters. This way of processing data lets us deal with any kind of dataset or data stream independently of size. The examples used by the algorithm are sampled with reposition from the dataset or they can be taken directly from the data stream. It is important that the source of data does not have concept drift because IADEM is not ready to afford it yet.

In this algorithm, a very important point are the user's requirements about maximum level of error and confidence. Thus, the user must set two arguments for the algorithm: the maximum desired error for the tree that is going to be induced (ε) and its confidence ($1 - \delta$). IADEM calculates some statistics using the counters stored in the decision tree. In addition to the estimated values for that statistics, the error margins for those estimated values are calculated too. Using the statistics and their error margins, IADEM maintains, at every moment, an estimation of the superior and inferior bounds of the error in the decision tree. That estimation is fundamental for the operation of this algorithm:

- the superior bound of the error ($\sup(error)$) is used to stop the algorithm when the user's requirements are satisfied ($\sup(error) \leq \varepsilon$). Thus, the number of examples needed to stop the algorithm does not depend on the size of the dataset. If the dataset is too small, it will continue sampling with reposition until the maximum error in the tree comes below the desired level. On the other hand, it is possible that the algorithm uses less examples than the number of examples in the dataset.
- the inferior bound of the error ($\inf(error)$) is used to limit the number of expansions. IADEM considers the size of the tree as an important point too. If $\inf(error)$ is lower than ε, that means, in the best case, that the error in the tree is below the desired error, so we can continue sampling and trying to reduce $\sup(error)$. Thus, the algorithm could find a solution without any additional expansion. Therefore, the first condition for considering to do an expansion is that $\inf(error)$ must be close to ε.
- the different values of $\sup(error)$ when expansions occur are used to detect noise. When IADEM expands a leaf node, $\inf(error)$ and $\sup(error)$ usually decrease. But, when the error in the tree is close to the noise in the datasets, the behaviour changes and it is usual that $\sup(error)$ increase after an expansion. We consider a maximum level of increases of $\sup(error)$ in the last expansions, and the algorithm stops is this level is exceeded.

IADEM uses two kinds of nodes in the border of the tree: the real and the virtual nodes. The real ones are the leaves of the tree and they constitute the real border of the tree. Every real node (or leaf) has as many virtual nodes as attributes are unused in the branch that ends in that leaf. The set of virtual nodes that corresponds to a real node represents all the possible expansions for that leaf and they register all the information

that will be needed to do an expansion. The information stored in the real nodes is the following: the total number of examples sampled since the node exists as virtual ($t \in \mathbb{N}$), the number of those that match with the branch that reaches that node ($m \in \mathbb{N}$) and the number of those examples depending on the class label ($m_k \in \mathbb{N}$ where $k \in \{1 \dots z\}$ and z is the number of classes). On the other hand, the information stored in the virtual nodes is: the total number of examples sampled since the virtual node exists ($t' \in \mathbb{N}$), the number of those that match with the branch that reaches that virtual node depending on the value for the corresponding virtual node attribute ($m'_v \in \mathbb{N}$ where $v \in \{1 \dots r\}$ and r is the number of values of the attribute) and the number of those examples depending on the class label ($m'_{v,k} \in \mathbb{N}$).

Considering these counters and the number of examples that have been processed, and using the Chernoff and Hoeffding bounds, IADEM provides estimated values and error margins for different calculated elements: the probability of one example reaching a real node ($w = m/t$); and the probability of an example being classified as k class in a real node ($p_k = m_k/m$) or in a virtual node ($p'_{v,k} = m'_{v,k}/m'$). To calculate the error margins for them the following general expression is used:

$$\varepsilon\_margin(x) = \min \left\{ \sqrt{\frac{3 \cdot a}{b} \ln(2/\delta)}\,,\ \sqrt{\frac{1}{2 \cdot b} \ln(2/\delta)}\,,\ 1 \right\} \tag{1}$$

and the values for a and b depend on the calculated element (x) which error margin is being considered. Thus, for $x = w$ we have $a = m/t$ and $b = t$; for $x = p_k$ we have $a = m_k/m$ and $b = m$; and for $x = p'_{v,k}$ we have: $a = m'_{v,k}/m'_v$ and $b = m'_v$.

With those elements and their error margins we can calculate the error produced in every leaf node of the tree. Using those errors we can estimate the superior and inferior bound of the error in the tree. Those calculated elements are also used in the expansion process. To do an expansion we need to identify which is the node that will be expanded. That leaf will be the one that contributes the most error to the tree. Once we have selected the leaf node, we find the best attribute to do the expansion using the information in the virtual nodes.

3 IADEMc

Despite IADEM's characteristics, we have added two new features in order to increase the usability and performance of the algorithm: the ability to deal with numerical attributes and the capability to apply naive Bayes classifiers in the tree leaves. This new extended version has been called IADEMc and presents the same extensions that VFDTc [14] added to VFDT, although there are some differences.

3.1 Numerical Attributes

Real world problems usually contain numerical attributes, so this issue should be addressed by learning algorithms. Preprocessing is a possibility of facing this kind of problems, but we have opted for including a local method in the algorithm.

Batch decision tree learners need a sort operation to work with these attributes, and it is the most time consuming operation. In this subsection we will present an efficient

method to deal with numerical attributes in the context of incremental decision tree learning. This method is based in the one proposed by Gama et al. [14].

In IADEMc a decision node that contains a split-test based on a continuous attribute ($attr_j$) has a condition and two descendant branches. The condition sets a cut point (cut_point) and the branches corresponds to the values that are less or equal than the cut point ($attr_j \leq cut\_point$) and greater than the cut point ($attr_j > cut\_point$). The cut_point is chosen using all the possible observed values for that attribute. The problem of accessing and storing counters for all different values of attributes in a short time is solved using balanced binary trees. Accessing to a value and inserting a new value in this structure is $O(\log_2(n))$ in the worst case, where n is the number of distinct values for the attribute seen so far.

The structure that we have used to store those values is managed in continuous virtual nodes. Every real node (leaf) has as many continuous virtual nodes as continuous attributes are defined in the problem. The same continuous attribute can be used multiple times in the same branch. The structure that stores the different values of an attribute differs from the one used in VFDTc in two points: the binary tree is a balanced tree (what allow insertion and access in $O(\log_2(n))$) and the amount of information stored in every node is lesser. Every node in the balanced binary tree is labelled with a value (i) of the corresponding continuous attribute and it keeps one vector which dimension is equal to the number of classes (k) and one additional counter. The vector ($counters\_per\_class_i$) is used to count the number of examples that have the value that labels that node taking into account the class label. The additional counter ($sum\_counters_i$) is the sum of all the counters in the vector. When an example reaches, the value of the considered attribute is examined and only the node corresponding with that value is updated; if the value is not in the balanced binary tree, a new node is inserted. The binary tree includes some other general information as the total number of examples registered in the tree ($total$) and this number taking into account the class label ($total\_per\_class$). This general information in combination with the counters stored in the nodes of the balanced binary tree is used to calculate the inferior and superior bound of the probability of one example being classified as k class in a virtual node ($\inf(p'_{v,k})$ and $\sup(p'_{v,k})$). These bounds are calculated for every possible cut_point using an exhaustive method that only traverses the balanced binary tree once.

Every possible condition that can be built using the different cut points can be seen as a nominal attribute with two values: true (when values are $\leq cut\_point$) and false (when values are $> cut\_point$). To calculate the estimated value of $p'_{true,k}$ and $p'_{false,k}$ in every cut_point we only need the number of examples per class ($m'_{true,k}$ and $m'_{false,k}$) and the total number of examples (m'_{true} and m'_{false}) at both sides of the cut_point. This counts are instantly calculated for the first cut point (after the smallest value –$smallest\_value$ –) of the attribute:

- the true branch ($\leq smallest\_value$) uses directly the counters in the node identified by the smallest value: $counters\_per\_class_{smallest\_value}$ (that corresponds with $m'_{true,k}$) and $sum\_counters_{smallest\_value}$ (that corresponds with m'_true).
- the false branch ($> smallest\_value$) uses the general counters $total\_per\_class$ and $total$; and subtracts from them the counts calculated in the true branch. Thus, $m'_{false,k} = total\_per\_class_k - m'_{true,k}$ and $m'_{false} = total - m'\_true$

To calculate the values of $p'_{v,k}$ for successive cut points is very simple:

- the true branch ($\leq cut\_point$) increases its counters using the counters of the node that change from the false side of the cut_point to the true side.
- the false branch ($> cut\_point$) decreases its accumulated counters using the counters of the node that leaves the false side of the cut_point and goes to the true side.

At the same time that the estimated values for $p'_{v,k}$ are calculated, the error margins for them are calculated too using the Chernoff and Hoeffding bounds (Equation 1).

We have explained how to calculate the interval where probability of one example being classified as k class in a virtual node ($p'_{v,k}$) can vary, but we must note that this process is not executed when a new group of examples is sampled and it is not executed for every continuous virtual node. The calculation is only made when an expansion is allowed and the leaf node that contributes the most error to the tree ($worst\_node$) is selected. Then we have to find which is the best attribute for doing the expansion in that node and it is in that moment when the previous intervals are calculated.

The selection of the best attribute keeps similar to the process used when we only considered nominal attributes: we look for the attribute which superior bound of the disorder measure (we used entropy) is the lowest.

3.2 Improving Prediction With Functional Tree Leaves

A very simple strategy to classify a test example is to find the leaf node that match with that example and classify it as the most representative class of the training examples that fall at that leaf. One extension incorporated in IADEMc is the ability to use the naive Bayes classifiers at tree leaves as it is used in VFDTc [14]. Thus, a test example is classified with the class that maximises the posterior probability given by Bayes rule assuming the independence of the attributes given the class.

There is a simple motivation for this option. IADEMc only changes a leaf to a decision node when there are an attribute that it is clearly the best attribute (given a confidence). Usually hundreds or even thousands of examples are kept in leaf nodes before an expansion is done. To classify a test example, the majority class strategy only use the information about class distributions and does not look for the attribute-values. It uses only a small part of the available information, a crude approximation to the distribution of the examples. On the other hand, naive Bayes takes into account not only the prior distribution of the classes, but also the conditional probabilities of the attribute-values given the class. In this way, there is a much better exploitation of the available information at each leaf. Moreover, naive Bayes is naturally incremental. It deals with heterogeneous data and missing values. It has been observed [21] that for small datasets naive Bayes is a very competitive algorithm.

Given the example $\overrightarrow{e} = (x_1, \ldots, x_j)$ and applying Bayes theorem, we obtain: $P(C_k|\overrightarrow{e}) \propto P(C_k) \cdot \prod P(x_j|C_k)$. To compute the conditional probabilities $P(x_j|C_k)$ we should distinguish between nominal attributes and continuous ones. In the former the problem is trivial using the counters stored in the nominal virtual nodes. In the latter, there are two usual approaches: assuming that each attribute follows a normal

distribution or discretizing the attributes. Assuming a normal distribution, the sufficient statistics can be computed on the fly. Nevertheless, it is possible to compute the required statistics from the binary-tree structure stored at each continuous virtual node. We have opted to implement this method in IADEMc. Any numerical attribute is discretized into $\min(10, Number\_of\_different\_values)$ equal width intervals.

We should note, that the use of naive Bayes classifiers at tree leaves does not introduce any overhead in the training phase. In this phase and for nominal attributes, the counters constitute (directly) the naive Bayes tables. For continuous attributes, the naive Bayes contingency tables are efficiently derived from the balanced binary trees used to store the numeric attribute-values. The overhead introduced is proportional to depth of the binary tree, that is at most $\log_2(n)$, where n is the number of different values observed for a given attribute.

4 Experiments and Results

The experiments we have done and the results that we have obtained are exposed in this section. The experiments have been done using synthetic datasets from *UCI Machine Learning Repository* [22]. We have selected *LED* (24 nominal attributes) and *Waveform* (21 and 40 continuous attributes) in order to show how the new extended algorithm IADEMc can deal with nominal and continuous attributes and how the using of functional leaves can improve the prediction method. The dataset generators have been used to build datasets of different sizes (from 10K to 1000K examples) with 10% noise. In addition, for every size of dataset we have built 10 datasets using different seeds.

For studying the performance of IADEM we have selected two criteria: the classification accuracy – the percentage of test examples that are correctly classified – and the size of the tree – number of nodes. The results given are the average value and the standard deviation. In addition, induction time has been also measured and its average value is given too. We have compared IADEMc with several algorithms. As a traditional algorithm we have chosen C4.5 [3], while as an incremental algorithm we have selected VFDTc [14]. We have selected these algorithms because they are well-documented algorithms, they induce decision trees, and their implementations can be found available in the Internet. For the experiments, we have used the implementation of C4.5 given in Weka [23], and the implementation of VFDTc algorithm available in http://www.liacc.up.pt/∼jgama/ales2/vfdtc. For experiments with IADEMc, we have implemented a prototype in Java and it has been configured with default parameters, desired error (ε) equals to 0.001 and confidence (δ) equals to 0.1. We must note that all algorithms have been executed using their default configuration.

As we can see in Table 1, the results show some interesting points. One of the most important is about the memory requirements of different algorithms: we can see how traditional algorithms are not good for dealing with very large datasets (or data streams). C4.5 crashed because it went out of memory, while VFDTc or IADEMc had no problem.

The improvement of using functional tree leaves is clear in experiments done with VFDTc and IADEMc. As we have mentioned before, using stronger classification strategies at tree leaves improves classifier's performance. Considering the accuracy achieved

Table 1. Accuracy (average ± standard deviation). *Algorithm*-MC means that the prediction uses Majority Class while *Algorithm*-NB means that it uses Naive Bayes.

Led dataset - 24 attributes					
	C4.5	VFDTc-MC	VFDTc-NB	IADEMc-MC	IADEMc-NB
100K	73.28 ± 0.04	25.12 ± 0.07	73.80 ± 0.11	71.13 ± 1.09	73.60 ± 0.25
500K	Out of memory	25.13 ± 0.04	73.79 ± 0.16	70.89 ± 1.02	73.63 ± 0.38
1000K	Out of memory	25.27 ± 0.25	73.86 ± 0.06	70.77 ± 0.96	73.60 ± 0.51
Waveform dataset - 21 attributes					
	C4.5	VFDTc-MC	VFDTc-NB	IADEMc-MC	IADEMc-NB
100K	80.17 ± 0.08	74.87 ± 0.42	82.15 ± 0.46	77.13 ± 0.76	81.34 ± 0.38
500K	81.48 ± 0.08	78.69 ± 0.20	83.27 ± 0.21	75.94 ± 1.82	81.08 ± 0.71
1000K	Out of memory	79.81 ± 0.20	83.50 ± 0.10	76.12 ± 2.23	81.08 ± 1.03
Waveform dataset - 40 attributes					
	C4.5	VFDTc-MC	VFDTc-NB	IADEMc-MC	IADEMc-NB
100K	79.05 ± 0.09	74.98 ± 0.50	81.78 ± 0.42	77.00 ± 0.90	80.51 ± 0.56
500K	80.50 ± 0.10	78.68 ± 0.25	82.63 ± 0.21	76.93 ± 1.14	80.99 ± 0.95
1000K	Out of memory	79.71 ± 0.10	82.73 ± 0.10	76.33 ± 0.92	81.22 ± 0.96

by IADEMc we can see that it is similar to the accuracy achieved by C4.5 and very close to that achieved by VFDTc. The reason for not reaching higher degrees of accuracy (and producing a higher variance) is a premature trigger of the noise detection heuristic. An anomalous behaviour can be observed while using VFDTc with LED dataset but it may be produced because of a bad selection of the expansions. The trees induced by VFDTc have many empty leaves when using LED dataset.

Table 2. Number of nodes (average ± standard deviation) and induction time (msec. average)

Led dataset - 24 attributes - NODES				Led-24 - MSEC.		
	C4.5	VFDTc	IADEMc	C4.5	VFDTc	IADEMc
100K	8055.00 ± 154.76	17.20 ± 1.14	47.80 ± 16.50	9579	820	4634
500K	Out of memory	81.60 ± 1.65	44.60 ± 12.57	Out mem.	4542	4531
1000K	Out of memory	154.20 ± 2.15	43.80 ± 12.19	Out mem.	10190	4311
Waveform dataset - 21 attributes - NODES				Waveform-21 - MSEC.		
	C4.5	VFDTc	IADEMc	C4.5	VFDTc	IADEMc
100K	8042.60 ± 236.89	49.20 ± 1.75	119.80 ± 63.04	76713	5707	35939
500K	33434.80 ± 316.38	245.00 ± 5.50	72.80 ± 41.34	1005955	32245	23487
1000K	Out of memory	487.80 ± 9.53	84.20 ± 52.02	Out mem.	67368	29022
Waveform dataset - 40 attributes - NODES				Waveform-40 - MSEC.		
	C4.5	VFDTc	IADEMc	C4.5	VFDTc	IADEMc
100K	9699.60 ± 184.65	49.60 ± 2.67	122.60 ± 54.05	133497	11536	75082
500K	42682.80 ± 450.45	242.80 ± 5.20	81.20 ± 36.15	1769030	64045	55299
1000K	Out of memory	490.60 ± 5.72	69.00 ± 33.12	Out mem.	132297	44455

Considering the results shown in Table 2 and in Figure 1 we can notice other interesting points. The size of the decision trees always increase in C4.5 and VFDTc while

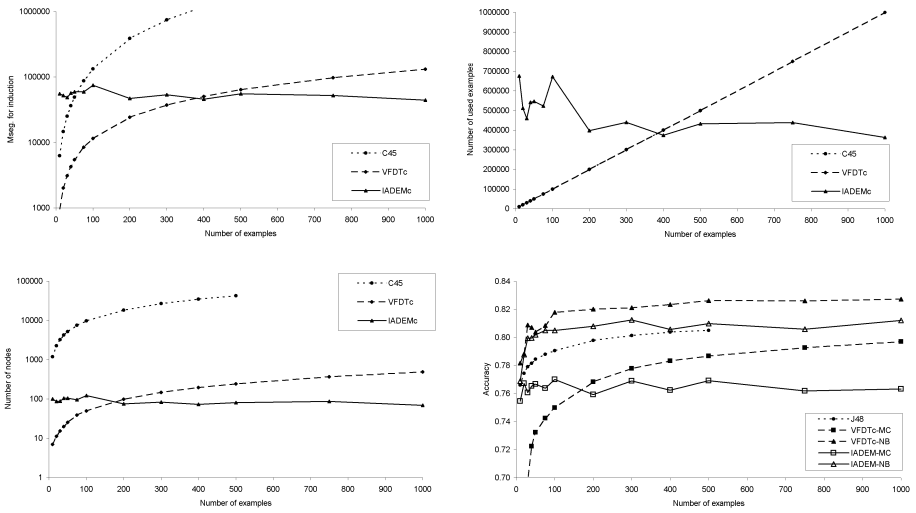


Fig. 1. Induction time, number of examples used, tree size and accuracy

this size keeps quite stable in IADEMc. The main reason for this behaviour is the noise detection heuristic and how it is used to stop the algorithm. The algorithm tries to satisfy the user requirements, but, if noise is detected, the execution stops. Thus, examples are sampled while the algorithm runs and this is a great difference with respect to C4.5 and VFDTc because they use the entire dataset. The accuracy achieved by VFDTc when there are few examples is not very good because it needs many examples to expand the decision tree. On the other hand IADEMc samples with reposition as many examples as are needed and achieves good accuracy even with small datasets.

5 Conclusion

This paper introduces two major extensions to IADEM, an incremental classifier for learning from increasingly common high-volume datasets and data streams. The first one is the ability to deal with numerical attributes and the second one is the ability to apply naive Bayes classifiers in tree leaves. While the former extends the domain of applicability of the algorithm to heterogeneous data, the latter reinforces the any-time characteristic, an important property for any incremental learning algorithm. IADEMc maintains all the desirable properties of IADEM. It is an incremental algorithm, new examples can be incorporated as they arrive, it works online, only see one example once, and using a small processing time per example. The experimental evaluation of IADEMc clear illustrates the advantages of using more powerful classification techniques.

Out aim of improving IADEMc involves some issues. We are working to improve the detection of noise in order to get accuracy closer to the maximum ones. Another important aspect that we will tackle is learning in the presence of concept drift.

References

1. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM Press (2003) 226– 235
2. Quinlan, J.R.: Induction of decision trees. Machine Learning **1** (1986) 81– 106
3. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
4. Mehta, M., Agrawal, R., Rissanen, J.: SLIQ: A fast scalable classifier for data mining. Lecture Notes in Computer Science **1057** (1996) 18– 32
5. Shafer, J.C., Agrawal, R., Mehta, M.: SPRINT: A scalable parallel classifier for data mining. In: Proc. 22th Int. Conf. on Very Large Data Bases, Morgan Kaufmann Publishers (1996) 544– 555
6. Smith, E.E., Medin, D.L.: Categories and Concepts. Harvard University Press (1981)
7. Schlimmer, J.C., Fisher, D.H.: A case study of incremental concept induction. In: Proc. 5th Nat. Conf. on Artificial Intelligence, Philadelphia, Morgan Kaufmann (1986) 496– 501
8. Utgoff, P.E., Berkman, N.C., Clouse, J.A.: Decision tree induction based on efficient tree restructuring. Machine Learning **29**(1) (1997) 5– 44
9. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proc. of the 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM Press (2000) 71–80
10. Maloof, M.A., Michalski, R.S.: Selecting examples for partial memory learning. Machine Learning **41**(1) (2000) 27– 52
11. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. Annals of Mathematical Statistics **23** (1952) 493– 507
12. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58** (1963) 13– 30
13. Yang, J., Wang, W., Yu, P.S., Han, J.: Mining long sequential patterns in a noisy environment. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, ACM Press (2002) 406– 417
14. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: Proc. of the 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM Press (2003) 523–528
15. Ramos-Jiménez, G., del Campo-Ávila, J., Morales-Bueno, R.: Incremental algorithm driven by error margins. Lecture Notes in Artificial Intelligence: 9th Int. Conf. on Discovery Science (2006) to appear
16. Michalski, R.S.: Knowledge repair mechanisms: Evolution vs. revolution. In: Proc. 3rd Int. Workshop on Machine Learning, Skytop, PA. (1985) 116– 119
17. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man, and Cybernetics **31** (2001) 497– 508
18. Kohavi, R.: Scaling up the accuracy of Naive-Bayes classifiers: a decision tree hybrid. In: Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining, AAAI Press (1996) 202–207
19. Gama, J.: Functional trees. Machine Learning **55**(3) (2004) 18– 32
20. Utgoff, P.: Perceptron trees: a case study in hybrid concept representations. In: Proc. of the 7th National Conference on Artificial Intelligence, Morgan Kaufmann (1988) 601–606
21. Domingos, P., Pazzani, M.: On the optimality of the simple bayesian classiffier under zero-one loss. Machine Learning **29** (1997) 103–129
22. Blake, C., Merz, C.J.: UCI repository of machine learning databases. University of California, Department of Information and Computer Science (2000)
23. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco (2000)

Analyzing Data Streams by Online DFT

Alexander Hinneburg[1], Dirk Habich[2], and Marcel Karnstedt[3]

[1] Martin-Luther University of Halle-Wittenberg, Germany
hinneburg@informatik.uni-halle.de
[2] Dresden University of Technology, Germany
dirk.habich@tu-dresden.de
[3] Technical University of Ilmenau, Germany
marcel.karnstedt@tu-ilmenau.de

Abstract. Sensor data have become very huge and single measures are produced at high rates, resulting in *streaming* sensor data. In this paper, we present a new mining tool called Online DFT, which is particularly powerful for estimating the spectrum of a data stream. Unique features of our new method include its low update complexity with high-accuracy estimations for very long periods, and its ability of long-range forecasting based on our Online DFT. Furthermore, we describe some applications of our Online DFT.

1 Motivation

Physical sensor environments can be found all over the world, and they are involved in a wide range of applications. In recent time, sensor data have become very huge and single measures are produced at high rates, resulting in *streaming* sensor data. Examples can be found in financial applications dealing with time series for various stocks, in applications monitoring sensors to detect failures and problems in large buildings at an early stage, or in real-time market analyses of supermarket customer data.

Aside from application-related issues, interesting general questions in data stream environments are: What are the dominant frequencies of a streaming signal? Which of them are currently most active in the stream? Is the stream very volatile (mostly high frequencies are active) or more stable (low frequencies are active)? How does the stream look like in the near future? To analyze the frequencies in a data stream, the discrete Fourier transform (DFT) in its original form is an unqualified tool because the Fourier coefficients do not contain any information about the temporal location of the frequencies. To overcome this disadvantage, the short time Fourier transform (STFT) can be used, which chops the stream into overlapping pieces and estimates frequencies from these pieces. The STFT represents a compromise between the resolution in the frequency domain and in the time domain. However, due to the processing of pieces, it is difficult to use the frequency information to forecast the stream. Wavelets have been used to analyze streams as well. Aside from many advantages of wavelets, such as fast online decomposition, there are also some disadvantages. First, there is no one-to-one relation between wavelet coefficients and frequency intervals.

This complicates the task of interpreting the results of a wavelet transform. Second, the wavelet transform is not shift-invariant; that means, the wavelet coefficients of a stationary signal (i.e., frequencies do not change) also depend on the starting point of the transformation. This is not the case for the DFT, and therefore, we decided not to use a wavelet model as starting point.

Our main contribution is a one-pass method with sublinear runtime for estimating the frequencies of a data stream. Our Online DFT approach partitions the frequency range into subintervals of increasing size. Low frequencies are estimated with higher resolution, while high frequencies are estimated more coarsely. This is justified by the observation that many data streams are driven mainly by low frequencies. Frequencies within the subintervals are estimated separately, which allows using different sample rates for different frequency subintervals. This leads to enormous savings in runtime because the sample rates can be adjusted to the estimated frequencies (low frequencies need only low sample rates). The second feature of the new Online DFT is its adaptive choice of sample points, which increases the power of our approach. The properties of the Online DFT lead to a number of interesting applications, e.g., forecasting, cleaning, and monitoring of streams, which are described in detail in the experiment section.

Related Work Stream mining is related to time series analyses and forecasting (e.g., [5]), which is based on discovering patterns and periodicity [3, 6]. Traditional generative time series models include auto-regressive (AR) models and their generalizations (ARMA etc.). They can be used for forecasting, but usually they fail in streaming environments and often have a number of limitations (see [11] for more details on this). There are also approaches for nonlinear forecasting, but they always require human intervention [12]. Papadimitriou et al. [11] utilize incremental DWT [4] in order to find periods in streams and to do long-range forecasting. In their AWSOM method, the authors model the wavelet coefficients by an online linear auto-regression model [2]. This work focuses on forecasting and thus, it is related to the proposed Online DFT, but it is based on completely different methods. In this work, we focus on the DFT approach and its characteristics only.

The Fourier transform has successfully been applied to numerous tasks. Recent work deals with the approximation of large Fourier coefficients [9] by sampling only a part of a time series. [5] discover representative trends using sketches, and they utilize FFT to compute distances between stream sections. Based on this, [1] define approximate periodicity and apply efficient sampling techniques in order to find potential periods. Most relevant to our work is indexing of time series (e.g., [8]), which would benefit from the availability of a fast Online DFT. An approach closely related to the presented Online DFT is the incremental algorithm introduced by Zhu and Shasha [13]. They use DFT to summarize streams within a finite window. In contrast to our work, they do not focus on forecasting but on finding correlations between different streams. Moreover, they uniformly discretize the frequency domain, whereas we represent lower frequencies with higher accuracy.

The remainder of this paper is organized as follows: After presenting necessary prerequisites in Section 2, we introduce our novel Online DFT approach in Section 3. In Section 4 we discuss some complexity aspects and present some data stream experiments. The paper ends with the conclusion in Section 5.

2 Prerequisites

Discrete Fourier Transform The n-point discrete Fourier transform of a time sequence $x = x_0, \ldots, x_{n-1}$ is a sequence $X = X_0, \ldots, X_{n-1}$ of complex numbers given by

$$X_f = {}^1/_{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cdot e^{-j2\pi f \cdot t/n}, \quad f = 0, \ldots, n-1$$

where j is the imaginary unit $j = \sqrt{-1}$. The original signal x can be recovered by the inverse Fourier transform. The runtime complexity of the discrete Fourier transform is $O(n^2)$. The fast Fourier transform (FFT) can compute the DFT coefficients in $O(n \log n)$.

Filtering When estimating a frequency spectrum from a finite sample, aliasing becomes an issue. Aliasing happens if the stream contains frequencies larger than half the sample rate. Those frequencies distort the spectrum in an unpredictable way and cannot be removed afterwards. To be on the safe side, high frequencies are removed from the stream by a low pass filter before applying the DFT. Filtering can be done in the frequency as well as in the time domain. The advantage of a time-domain filter is that the output values have the same rate as the input data. Linear filters in the time domain take a sequence x_t of input values and produce a sequence y_t of output values by the formula: $y_t = \sum_{k=0}^{M} c_k \cdot x_{t-k} + \sum_{j=1}^{N} d_j \cdot y_{t-j}$. The $M + 1$ coefficients c_k and the N coefficients d_j are fixed and define the filter response. There exists an elaborate theory on how to determine the coefficients c_k and d_j for a given pass band. In the later sections, we use Chebyshev filters, but any other filter can be used as well. We observed that the setting $M = 5$ and $N = 5$ gives good results in the experiments. However, the results are not sensitive to the particular setting.

Any filter delays the signal. Note that the delay is in general not a constant but a function depending on the frequency. The delay for specific frequencies can be determined from the filter coefficients. We denote the phase delay by $\tau(f)$, where f is the normalized frequency. Details on how to determine the filter coefficients and the delay can be found in [7].

3 Online DFT

Now, we propose our novel data analyzing approach for streams based on Online DFT. We introduce a model for the stream which is based on Fourier coefficients. The approach is well-suited for streaming scenarios as it has sublinear runtime complexity.

The method slides a window over the stream, which includes the last T elements. The straight-forward estimation of the frequencies from the window using the fast Fourier transform (FFT) runs in $O(T \log T)$. For large T, online processing is not feasible, since the FFT has to be re-applied when new data arrives. Our solution to the problem is to break up the frequency domain into smaller intervals. The frequency intensities in each subinterval are estimated from a small sample.

3.1 Online DFT Streaming Model

In this paper, we consider streaming elements $S = x_0, x_1, \ldots, x_{t-1}, x_t$, which are measured at an even rate. The window of the last T stream elements at the current time index t is denoted by $W_t = x_{t-T-1}, \ldots, x_{t-1}, x_t$. We assume that (i) the duration between the arrival of two consecutive data stream elements is one standard time unit, and (ii) the stream does not contain frequencies above the critical Nyquist frequency[4] $f_c = 1/2$.

The goal is to estimate all frequencies between zero and $1/2$ of the window W_t to model the stream. Our idea is to partition the frequency range $(0, 1/2)$ into L subintervals. The l-th interval is $(1/2^{l+1}, 1/2^l)$ for $1 \leq l < L$ and $(0, 1/2^l)$ for $l = L$. So, the numbering goes from right to left as shown below:

$$(0, 1/2^L),\ (1/2^L, 1/2^{L-1}),\ \ldots, (1/2^{2+1}, 1/2^2),\ (1/2^{1+1}, 1/2^1)$$

The intervals can be estimated separately. We use short sliding windows for high frequencies and long sliding windows for low frequencies. Using filters, the input for the estimation of the l-th frequency interval $(1/2^{l+1}, 1/2^l)$ is bandwidth-limited to $1/2^l$. Due to the Nyquist theorem, the sample rate for the input can be much lower without loss of information. We use a bandpass filter with a pass band $(1/2^{l+1}, 1/2^l)$ for $1 \leq l < L$ and a lowpass filter $(0, 1/2^l)$ when $l = L$, respectively. The filter used for the l-th interval is denoted by H_l and the filtered stream by $H_l(S) = y_1^{(l)}, y_2^{(l)}, \ldots, y_{t-1}^{(l)}, y_t^{(l)}$.

The distance between sample points, δ_l, increases with l. Thus, for high frequencies, the sample points are close, while for low frequencies, they lie more apart: $\delta_l = 2^{l-1},\ 1 \leq l \leq L$. To determine the frequencies in the l-th interval, we consider only the filtered data stream sample points with a distance of δ_l. This is an enormous saving, e.g., when $l = 8$, only every $\delta_l = 2^{8-1} = 128$th point has to be used as sample point, instead of having to use all points. As explained before, this saving comes without loss of information.

The number of sample points for each frequency interval is upper-bounded by $N_{\max}$, where $N_{\max} \ll T$ and $\delta_L \cdot N_{\max} \leq T$. The sample points are the points that arrived last in the stream. The actual number of sample points, N_l, for each frequency subinterval l is adaptively chosen with respect to the data stream in order to avoid artifacts. The adaptive choice of N_l is crucial for the power of our approach. Details on how to choose N_l will be explained in subsection 3.2.

[4] Filtering can be used to guarantee this assumption.

The number of intervals, L, depends on the maximal number of sample points, $N_{\max}$, and the size of the sliding window, T: $L = \max\{i \in \mathbb{N}\colon 2^{i-1} \cdot N_{\max} \leq T\} = \lfloor \log {}^{T-1}/_{N_{\max}-1} \rfloor + 1$. The input for the frequency estimation in the l-th subinterval is the sequence of sample points taken from the output of the filter H_l: $S^{(l)} = y^{(l)}_{t-(N_l-1)\delta_l}, y^{(l)}_{t-(N_l-2)\delta_l}, \ldots, y^{(l)}_{t-\delta_l}, y^{(l)}_t$, where $N_l \leq N_{\max}$. The frequencies in the l-th subinterval are estimated from $S^{(l)}$ by DFT. The output consists of N_l Fourier coefficients, from which those between $\lfloor {}^{N_l}/_4 \rfloor$ and $\lceil {}^{N_l}/_2 \rceil$ estimate frequencies in the desired interval. The first quarter of coefficients is zero due to the band pass filter. An exception is the estimate of the L-th interval, from which the Fourier coefficients between 1 and $\lceil {}^{N_l}/_2 \rceil$ are used. Figure 1 illustrates the interplay of the frequency estimation for different subintervals.

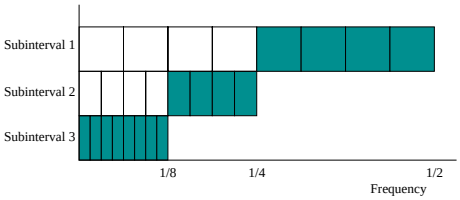
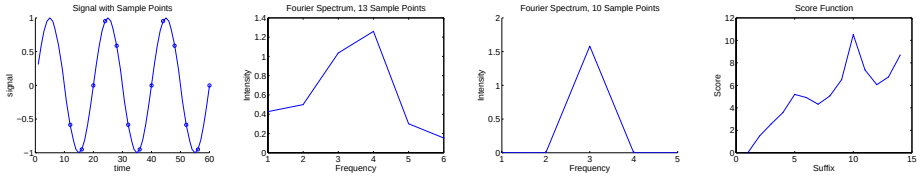


Fig. 1. Results of the DFT of $S^{(1)}, S^{(2)}, S^{(3)}$ in the frequency domain, $(L = 3, N_{\max} = 16)$. Each rectangle represents a Fourier coefficient, the shaded coefficients are non-zero.

After this overview of the Online DFT model, the next sections discuss some further details and explain how the model is used for several example applications on data streams. In subsection 3.2, we explain how to choose the number of sample points per level. In subsection 3.3, we introduce the inverse Online DFT.

3.2 The Difficult Choice of Sample Points

The estimation of dominant frequencies by the DFT from a finite set of sample points is very sensitive to the choice of actual sample points. As the DFT needs infinite support of the signal, it must be possible to cyclically continue to sample points. Otherwise, the Fourier spectrum would include some artificial frequencies. For example, let the signal be a pure sinus $s(t) = sin({}^{2\pi}/_{20} \cdot t)$, and



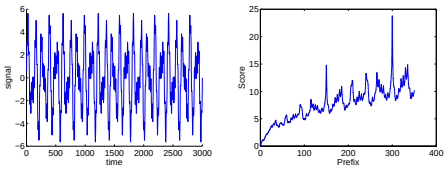
(a) (b) (c) (d)

Fig. 2. (a) Signal with sample points, (b) spectrum when all 13 points are used, (c) spectrum when only the last 10 points are used, (d) score function.

the current time point $t = 60$. If 13 sample points are used with $\delta = 4$ (Figure 2(a)), the spectrum based on this choice of sample points includes more than one non-zero component (Figure 2(b)). Ideally, the spectrum should include exactly one non-zero component because the signal is a pure tone. In case the right-most

three sample points are discarded, the DFT gives the desired spectrum (shown in Figure 2(c)).

When $N_{\max}$ sample points of the filtered signal $y_{t-(N_{\max}-1)\delta_l}$, $y_{t-(N_{\max}-2)\delta_l}$, $\ldots$, $y_{t-\delta_l}$, y_t are available, it remains to be decided which suffix of that sequence is the best choice to determine the spectrum by the DFT. The first observation is: if the sample points cannot be cyclically continued, some extra effort has to be applied to compensate the jumps at the borders. We call the sum of the absolute Fourier coefficients the energy E of the DFT, $E = \sum_{i=1}^{N_{\max}} |Y_i|$. If some compensations at the borders are necessary, this is reflected by increased energy. Thus, a criterion for the choice of a good suffix is to take one with low energy. As a second observation, the suffix needs a certain length to include information about all relevant vibrations. Very short suffixes may have low energy but do not reveal much about the signal's spectrum. The length of a suffix is denoted by $N_i \leq N_{\max}$ and its energy is E_i, which is computed by a DFT of that particular suffix of sample points. So, we are looking for a long suffix with low energy. We combine both requirements to a scoring function



(a) (b)

Fig. 3. (a) Complex signal and (b) the score function for the suffixes.

$$sc(N_i) = \frac{N_i \cdot \log N_i}{E_i} = \frac{N_i \cdot \log N_i}{\sum_{i=1}^{N_i} |Y_i|}$$

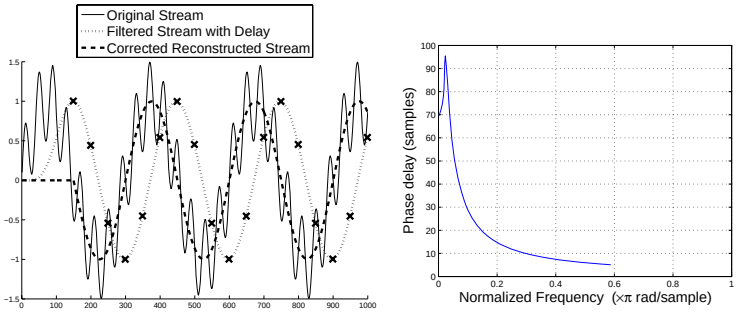
The scoring function of our example (Figure 2(d)) has a local maximum at suffix length 10, which gives the ideal spectrum, as shown in Figure 2(c). The figure shows that small, non-informative suffixes can also get a high score. To exclude those from further considerations, we set a lower bound $N_{\min}$ for the minimum length of a suffix. How to set $N_{\min}$ depends on the data stream, but we do not further investigate it here.

A more complex signal is shown in Figure 3(a). Our proposed scoring function (Figure 3(b)) has two clear peaks for the suffixes of length 150 and 300, which are exactly the positions at which the true spectrum of the signal is revealed.

3.3 Inverse Online DFT

The main obstacle for the inverse Online DFT is the delay from the used filters. We only gain from the splitting strategy if the bandpass filters can be applied at little runtime cost. Therefore, we use fast linear filters, e.g., Chebyshev filters, where filter delay is unavoidable. The example in Figure 4(a) shows the original stream (solid curve), which consists of two sinuses $x = \sin(2\pi/300) + 1/2 \sin(2\pi/40)$. The output of the filter is the dotted curve, which is delayed with respect to the original curve. The phase delay is not a constant but a function $\tau(f)$ depending on the normalized frequency (see Figure 4b).

Our idea is to correct the phase delay during the inverse DFT to be able to reconstruct the stream correctly. In the example, the corrected result is the dashed curve, which is also reconstructed between the sample points. The reconstructed curve (dashed) matches almost perfectly with the original $\sin(2\pi/300)$.



(a) (b)

Fig. 4. The left plot shows the original stream consisting of two sinuses, the filtered stream from which sample points (x) are taken, and the reconstructed stream with phase delay correction. The right plot shows the phase delay depending on the frequency.

To explain our example formally, we assume n sample points $y_1, \ldots, y_n$ with sample distance δ. From the sample points, we get n complex Fourier coefficients $Y_1, \ldots, Y_n$ by DFT. The reconstruction of the n sample points with corrected phase delay works as follows:

$$\tilde{y}_i = re\left(\frac{2}{\sqrt{n}} \cdot \sum_{k=0}^{\lceil \frac{n}{2} \rceil - 1} Y_k \cdot \exp\big(j \cdot 2\pi \cdot k \cdot (i + \tau(2\pi/kn\delta))/n\big)\right)$$

with $i = 0, \ldots, n-1$. The function $re()$ returns the real part of a complex number. In general, an inverse DFT of a real-valued stream does not need to use the $re()$ function, since – due to the symmetry property – each coefficient in the lower half has a counterpart in the upper half with the same real part, but with a negative imaginary part. Thus, the imaginary parts would be canceled out, while the real part is doubled. Here, we are additionally dealing with the phase delay correction. It is simpler to double the real part and to cut off the imaginary part explicitly.

When i is incremented in steps smaller than one, intermediate points between the sample points are computed. By that strategy, the reconstructed stream can be refined to the original resolution, or when $i > n-1$, even future stream values can be predicted. For the final inverse Online DFT, the inverse DFTs with phase delay correction from different levels are added up at the stream's original resolution.

4 Complexity Consideration and Experiments

In this section, we (1) describe some runtime and space complexity experiments and (2) present some applications of our Online DFT. As presented previously,

when a new stream element arrives, the L subintervals are completely recomputed and the sliding window is shifted. This is the simplest implementation. The main runtime factor is formed by choosing the best of all $N_{max} - N_{min}$ suffixes for each subinterval. When the DFT is implemented by FFT, this step takes time $\sum_{i=N_{\min}}^{N_{\max}} N_i \cdot \log N_i = O(N_{max}^2 \log N_{max})$ in each subinterval l, where N_i is the length of each possible suffix of sample points. Thus, the update runtime complexity of the core algorithm is $O(L \cdot N_{max}^2 \log N_{max})$, which can be simplified to $O(\lfloor \log {}^{T-1}/_{N_{\max}-1} \rfloor \cdot N_{max}^2 \log N_{max})$. This shows that the update runtime complexity is sublinear in the size of the sliding window.

Figure 5 illustrates these results on an empirical basis. We figure the runtime of the core algorithm (for $N_{\max} = 50$ and $N_{\max} = 100$) in comparison to the FFT applied to the entire sliding window, where T is varied and $N_{\min} = 1$. In both cases, the runtime grows sublinearly with the size T of the sliding window. The runtime of the Online DFT is a stair-case function: each step indicates the introduction of an additional subinterval. A better implementation is to investigate the best suffix only when the score for that suffix drops. We conducted an experiment on the sunspot data stream (see Figure 6) and computed the length of the best suffixes for each subinterval, where the window slided from time $1,000$ till $1,500$. The percentage of how often the score dropped and the suffix length changed with the next stream element is $l = 1 : 20\%, l = 2 : 76\%, l = 3 : 35\%, l = 4 : 19\%$. This shows that there is still room for optimization.

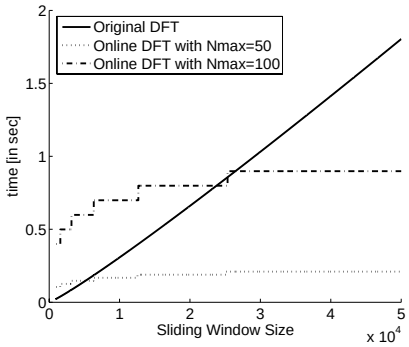


Fig. 5. Runtime Comparison of our Core Algorithm against the DFT applied on the entire sliding window.

The space complexity of our algorithm depends on the size of the sliding window. In order to be able to shift the window by one element, all elements have to be stored. We have to store $\delta_l N_{max}$ elements per subinterval, where the sum over all subintervals is in $O(T \cdot N_{max})$. The space needed can be reduced by applying time series compression techniques, e.g., piecewise linear approximation [10]. More optimization issues are the focus of our ongoing work.

Prediction of the future behaviors of data streams and time series is important for various applications. With the Online DFT, any time forecasting is possible using the inverse transformation. Figure 6 shows the original sunspot data stream, the forecasts based on our Online DFT, and the AWSOM forecast. As code of AWSOM is not available, we took the best plot for AWSOM for that experiment from [11]. The part to be forecasted (starting at time point 1000) shows 7 peaks in the original stream. AWSOM's forecast contains 5 peaks, while the forecast of our Online DFT includes 8 peaks. Both forecasts find the average peak width quite precisely, even though they are based on different approaches. Using the Online DFT, the average peak height is predicted more accurately.

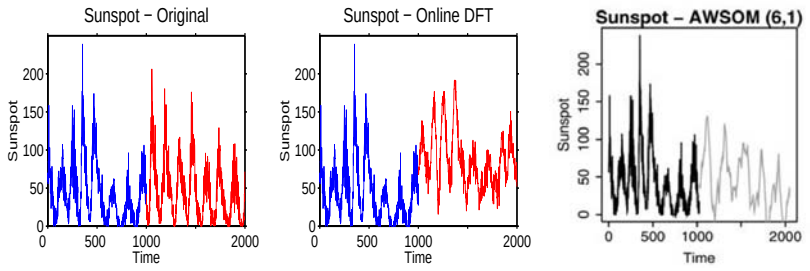


Fig. 6. Experiment for forecasting data streams; (a) original Sunspot stream; (b) forecasting based on Online DFT; (c) forecasting based on AWSOM.

Based on such a prediction property, we are also able to detect outliers and other kinds of deviations by comparing the current element with the prediction for that element. In general, outlier detection in data streams is a further important issue, e.g., it may be used to quickly detect situations in which something unexpected happens. Furthermore, knowledge about the frequency distribution in a data stream can be used for cleaning tasks. In particular, several anomalies are easier to correct in the frequency domain than in the time domain. There are many possible reasons for dirty sensor data. This includes missing and/or incorrect values (often forming noise). Reasons for those errors may be sensors low on power, network or communication problems, inaccurate sensors, etc. Therefore, streaming sensor data must be appropriately cleaned with respect to errors.

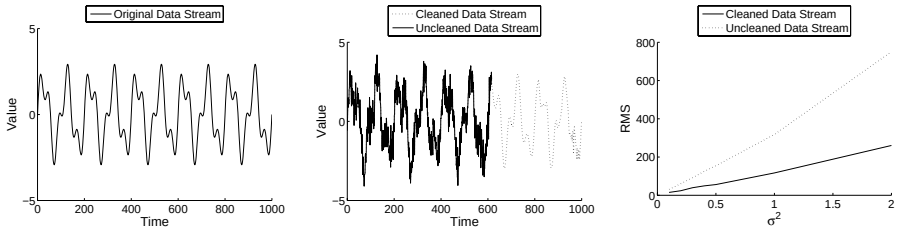


Fig. 7. Experiment for noise reduction; (a) original data stream, (b) data stream after cleaning, and (c) root means square errors (RMS).

To demonstrate the utilization of our Online DFT for cleaning tasks, we generated a synthetic data stream consisting of two sinuses. The original data stream is depicted in Figure 7(a). We simulated noise by adding a Gaussian distributed random number to each stream element. We investigated this noisy data stream with our proposed Online DFT approach. Figure 7(b) depicts the result of the noise reduction starting at time point 610. The noise cleaning task is processed by neglecting high frequencies and frequencies with small intensities. Using the inverse Online DFT, we are able to reconstruct a data stream with reduced noise. As the figures show, the cleaned data stream is close to the original generated data stream without noise.

To highlight the quality of this cleaning process, we determined the root mean square values (RMS) as error measures of (i) the cleaned data stream, and

(ii) the data stream with noise, and compared it against the pure data stream without noise. Figure 7(c) shows the RMS depending on different variances. The higher the variance, the more noise is included in the generated data stream. With our proposed Online DFT approach, we are able to significantly reduce noise in data streams.

5 Conclusion

In this paper, we proposed a novel online model for data streams, called Online DFT, for estimating the frequencies of a data stream. Characteristics of our Online DFT are its low update complexity with high-accuracy estimations for periods and its ability of long-range forecasting. Furthermore, we described some applications of our approach. Aside from forecasting of data streams, Online DFT can be applied to detect outliers by comparing the current arriving element with the prediction. We demonstrated how to use our Online DFT for data stream cleaning tasks. Open issues for future research are specialized implementations for sensor applications as well as cost- and quality-based optimizations of the used algorithms, especially an incremental version of our Online DFT. Additionally, we plan to do performance comparisons to related methods based on more real-world data sets.

References

1. F. Ergün, S. Muthukrishnan, and S. C. Sahinalp. Sublinear methods for detecting periodic trends in data streams. In *LATIN*, 2004.
2. Box et al. *Time Series Analysis: Forecasting and control*. Prentice Hall, 1994.
3. Elfeky et al. Periodicity detection in time series databases. *IEEE Trans. Knowl. Data Eng.*, 17(7), 2005.
4. Gilbert et al. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, 2001.
5. Indyk et al. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, 2000.
6. Keogh et al. Finding surprising patterns in a time series database in linear time and space. In *Proc. of the ACM SIGKDD*, 2002.
7. Oppenheim et al. *Discrete-time signal processing*. Prentice-Hall, Inc., 1989.
8. Ch. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.
9. A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. In *STOC*, 2002.
10. D. P. Kacso. Approximation by means of piecewise linear functions. *Results in Mathematics*, 35, 1999.
11. S. Papadimitriou, A. Brockwell, and Ch. Faloutsos. Adaptive, unsupervised stream mining. *The VLDB Journal*, 13(3), 2004.
12. A. S. Weigend and N. A. Gerschenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1994.
13. Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB*, 2002.

Early Drift Detection Method *

Manuel Baena-García[1], José del Campo-Ávila[1], Raúl Fidalgo[1], Albert Bifet[2], Ricard Gavaldà[2], and Rafael Morales-Bueno[1]

[1] Departamento de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática. Universidad de Málaga, Spain
{mbaena, jcampo, rfm, morales}@lcc.uma.es
[2] Universitat Politècnica de Catalunya, Spain
{abifet, gavalda}@lsi.upc.edu

Abstract. An emerging problem in Data Streams is the detection of concept drift. This problem is aggravated when the drift is gradual over time. In this work we define a method for detecting concept drift, even in the case of slow gradual change. It is based on the estimated distribution of the distances between classification errors. The proposed method can be used with any learning algorithm in two ways: using it as a wrapper of a batch learning algorithm or implementing it inside an incremental and online algorithm. The experimentation results compare our method (EDDM) with a similar one (DDM). Latter uses the error-rate instead of distance-error-rate.

1 Introduction

Many approaches in machine learning assume that training data has been generated from a stationary source. This assumption is likely to be false if the data has been collected over a long period of time, as it is often the case: it is likely that the distribution that generates the examples changes over time. In this case, change detection becomes a necessity. Real applications examples are user modelling, monitoring in biomedicine and industrial processes, fault detection and diagnosis, safety of complex systems, etc.

We are interested in drift detection over data streams. Data streams are unbounded sequence of examples received at so high a rate that each one can be read at most once [1]. So, we can not store all the examples in memory, only partially at maximum, and we can not spend so much time processing data, due to the high rate that it arrives.

In this work we present Early Drift Detection Method, a method to detect concept drift, that gets good results with slow gradual changes. Abrupt changes are easier to detect by current methods, but difficulties arise with the slow gradual changes. Our method uses the distance between classification errors (number of examples between two classification errors) to detect change, instead of using

* This work has been partially supported by the FPI program and the MOISES-TA project, number TIN2005-08832-C03, of the MEC, Spain.

errors classifications (as it is done in previous work [2]). It detects change faster, without increasing the rate of false positives, and it is able to detect slow gradual changes.

The paper is organised as follows. The next section presents related work in detecting concept drifting. In section 3 we present our method, Early Drift Detection Method. In Section 4 we evaluate the method and Section 5 concludes the paper and present future work.

2 Related Work

To deal with change over time, most previous work has been classified observing if they use full, partial or not examples memory.

The partial memory methods used variations of the sliding-window idea: at every moment, one window (or more) containing the most recently read examples is kept, and only those are considered relevant for learning. A critical point in any such strategy is the choice of a window size. The easiest strategy is deciding (or asking the user for) a window size W and keeping it fixed through the execution of the algorithm (see e.g. [3–5]). In order to detect change, one can keep a reference window with data from the past, also of some fixed size, and decide that change has occurred if some statistical test indicates that the distributions in the reference and current windows differ.

Another approach, using no examples memory, only aggregates, applies a "decay function" to examples so that they become less important over time [6].

Other approach to detect concept drift monitors the values of three performance indicators [7]: accuracy, recall and precision over time. Then they are compared with a confidence interval of standard sample errors for a moving average value (using the last M batches) of each particular indicator. The key idea is to select the window size so that the estimated generalisation error on new examples is minimised. This approach uses unlabelled data to reduce the need for labelled data, it doesn't require complicated parameterization and it works effectively and efficiently in practise.

A new method to detect changes in the distribution of the training examples monitors the online error-rate of the algorithm [2]. In this method learning takes place in a sequence of trials. When a new training example is available, it is classified using the current model. The method controls the trace of the online error of the algorithm. For the actual context they define a warning level, and a drift level. A new context is declared, if in a sequence of examples, the error increases reaching the warning level at example k_w, and the drift level at example k_d. They take this is as an indication of a change in the distribution of the examples. It uses 6σ ideas, well known in quality theory.

Our method it is based on the latter, but taking into account distances between classification errors as it is presented in the next section.

3 Drift Detection Methods

We consider that the examples arrive one at a time, but it would be easy to assume that the examples arrive in bundles. In online learning approach, the decision model must make a prediction when an example becomes available. Once the prediction has been made, the system can learn from the example (using the attributes and the class) and incorporate it to the learning model. Examples can be represented using pairs $(\overrightarrow{x}, y)$ where $\overrightarrow{x}$ is the vector of values for different attributes and y is the class label. Thus, i-th example will be represented by $(\overrightarrow{x}_i, y_i)$. When the current model makes a prediction (y'_i), it can be correct $(y_i = y'_i)$ or not $(y_i \neq y'_i)$.

3.1 DDM: Drift Detection Method [2]

There are approaches that pay attention to the number of errors produced by the learning model during prediction. The drift detection method (DDM) proposed by Gama et al. [2] uses a binomial distribution. That distribution gives the general form of the probability for the random variable that represents the number of errors in a sample of n examples. For each point i in the sequence that is being sampled, the error rate is the probability of missclassifying (p_i), with standard deviation given by $s_i = \sqrt{p_i(1-p_i)/i}$. They assume (as states the PAC learning model [8]) that the error rate of the learning algorithm (p_i) will decrease while the number of examples increases if the distribution of the examples is stationary. A significant increase in the error of the algorithm, suggests that the class distribution is changing and, hence, the actual decision model is supposed to be inappropriate. Thus, they store the values of p_i and s_i when $p_i + s_i$ reaches its minimum value during the process (obtaining p_{min} and s_{min}). And it checks when the following conditions triggers:

- $p_i + s_i \geq p_{min} + 2 \cdot s_{min}$ for the warning level. Beyond this level, the examples are stored in anticipation of a possible change of context.
- $p_i + s_i \geq p_{min} + 3 \cdot s_{min}$ for the drift level. Beyond this level the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p_{min} and s_{min} are reset too.

This approach has a good behaviour detecting abrupt changes and gradual changes when the gradual change is not very slow, but it has difficulties when the change is slowly gradual. In that case, the examples will be stored for long time, the drift level can take too much time to trigger and the examples memory can be exceeded.

3.2 EDDM: Early Drift Detection Method

The method that we propose in this paper, called Early Drift Detection Method (EDDM), has been developed to improve the detection in presence of gradual

concept drift. At the same time, it keeps a good performance with abrupt concept drift. The basic idea is to consider the distance between two errors classification instead of considering only the number of errors. While the learning method is learning, it will improve the predictions and the distance between two errors will increase. We can calculate the average distance between two errors (p'_i) and its standard deviation (s'_i). What we store are the values of p'_i and s'_i when $p'_i + 2 \cdot s'_i$ reaches its maximum value (obtaining p'_{max} and s'_{max}). Thus, the value of $p'_{max} + 2 \cdot s'_{max}$ corresponds with the point where the distribution of distances between errors is maximum. This point is reached when the model that it is being induced best approximates the current concepts in the dataset.

Our method defines two thresholds too:

- $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \alpha$ for the warning level. Beyond this level, the examples are stored in advance of a possible change of context.
- $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \beta$ for the drift level. Beyond this level the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p'_{max} and s'_{max} are reset too.

The method considers the thresholds and searches for a concept drift when a minimum of 30 errors have happened (note that it could appear a large amount of examples between 30 classification errors). After occurring 30 classification errors, the method uses the thresholds to detect when a concept drift happens. We have selected 30 classification errors because we want to estimate the distribution of the distances between two consecutive errors and compare it with future distributions in order to find differences. Thus, $p'_{max} + 2 \cdot s'_{max}$ represents the 95% of the distribution. For the experimental section, the values used for α and β have been set to 0.95 and 0.90. These values have been determined after some experimentation.

If the similarity between the actual value of $p'_i + 2 \cdot s'_i$ and the maximum value ($p'_{max} + 2 \cdot s'_{max}$) increase over the warning threshold, the stored examples are removed and the method returns to normality.

4 Experiments And Results

In this section we describe the evaluation of the proposed method: EDDM. The evaluation is similar to the one proposed in [2]. We have used three distinct learning algorithms with the drift detection methods: a decision tree and two nearest-neighbourhood learning algorithms. These learning algorithms use different representations to generalise examples. The decision tree uses DNF to represent generalisation of the examples. The nearest-neighbourhood learning algorithms uses examples to describe the induced knowledge. We use the weka implementation [9] of these learning algorithms: J48[10] (C4.5, decision tree), IB1[11] (nearest-neighbourhood, it is not able to deal with noise) and NNge[12] (nearest-neighbourhood with generalisation). We have used four artificial datasets previously used in concept drift detection [13], a new data set with

very slow gradual change and a real-world problem [14]. As we want to know how our algorithms work in different conditions, we have chosen those artificial datasets that have several different characteristics - abrupt and gradual drift, presence and absence of noise, presence of irrelevant and symbolic attributes, numerical and mixed data descriptions.

4.1 Artificial Datasets

The five artificial datasets used later (Subsections 4.2 and 4.3) are briefly described. All the problems have two classes and each class is represented by 50% of the examples in each context. To ensure a stable learning environment within each context, the positive and negative examples in the training set are alternated. The number of examples in each concept is 1000, except in Sine1g that have 2000 examples in each concept and 1000 examples to transit from one concept to another.

- SINE1. Abrupt concept drift, noise-free examples. The dataset has two relevant attributes. Each attribute has values uniformly distributed in $[0, 1]$. In the first concept, points that lie below the curve $y = sin(x)$ are classified as positive, otherwise they are labelled as negative. After the concept change the classification is reversed.
- CIRCLES. Gradual concept drift, noise-free examples. The examples are labelled according to a circular function: if an example is inside the circle, then its label is positive, otherwise is negative. The gradual change is achieved by displacing the centre of the circle and growing its size. This dataset has four contexts defined by four circles:

Center	(0.2,0.5)	(0.4,0.5)	(0.6,0.5)	(0.8,0.5)
Radius	0.15	0.2	0.25	0.3

- GAUSS. Abrupt concept drift, noisy examples. The examples are labelled according to two different but overlapped gaussian density functions ($N([0, 0], 1)$ and $N([2, 0], 4)$). The overlapping can be considered as noise. After each context change, the classification is reversed.
- MIXED. Abrupt concept drift, boolean noise-free examples. Two boolean attributes (v, w) and two numeric attributes (x, y). The examples are classified positive if at least two of the three following conditions are satisfied: v, w, $y < 0.5 + 0.3 \sin(3\pi x)$. After each concept change the classification is reversed.
- SINE1G. Very slow gradual drift, noise-free examples. This dataset remains the same as Sine1, but the concept drift is made by gradually choosing of examples from the old and the new concept. So, there is a transition time between concepts. The probability of selecting an example from the old concept becomes lower gradually and the probability of selecting an example from the new concept becomes higher when the transition time is ended.

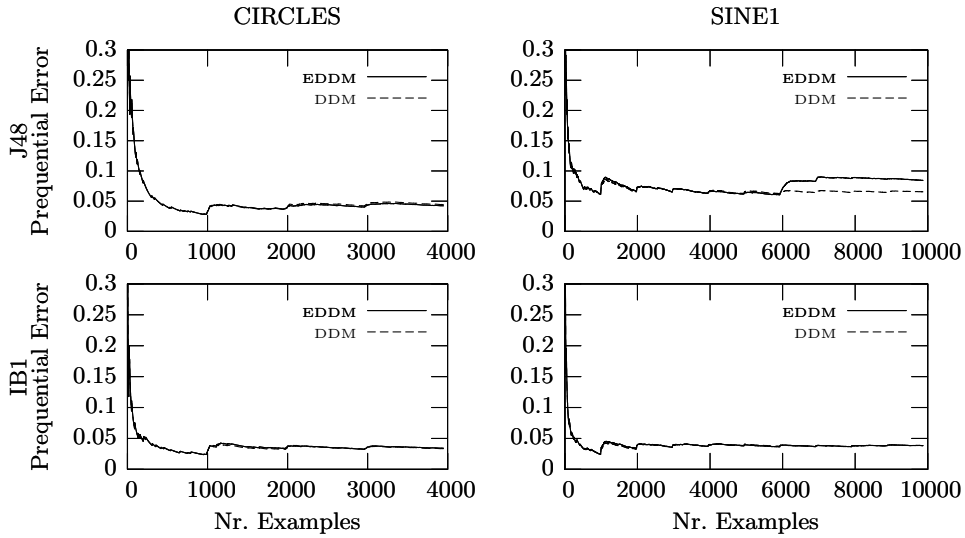


Fig. 1. Prequential error of J48 (up) and IB1 (down) on Circles (left) and Sine1 (right) datasets

4.2 Results On Artificial Domains: Abrupt And Gradual Drifts

The purpose of these experiments is to analyse how the proposed drift detection method works with different learning algorithms, and compare it to the one proposed by Gama et al. [2] (we refer to it as DDM, Drift Detection Method) in terms of prequential [15] error (prequential error is the mean of the classification errors obtained with the examples to be learnt). It is not the aim of this article to compare the results of the different learning algorithms used. Figure 1 shows the prequential error results when Circles and Sine1 datasets are faced with two learning algorithms with the proposed drift detection method (EDDM) and with DDM. Note that changes occur every 1000 examples.

We can observe that prequential error curves obtained by EDDM and DDM on Sine1 dataset (an abrupt changing dataset) are almost the same. When they deal with a gradual dataset (Circles) the results of both methods are very similar, independently of the learning method used.

Table 1 presents the prequential error results and the total number of changes detected by the learning algorithms with both drift detection methods at the end of each dataset. On datasets with abrupt concept change, both algorithms react quickly and reach low error rates. When noise is present, EDDM is more sensitive than DDM, detecting changes and improving the performance when the base algorithm does not support noise (i.e. IB1). This is so because after some time (some number of examples) the base algorithms overfit and the frequency of classification errors increases.

Table 1. Prequential error and total number of changes detected by methods

		EDDM		DDM	
		Prequential	Drifts	Prequential	Drifts
Circle	IB1	0.0340	3	0.0343	3
	J48	0.0421	3	0.0449	3
	NNge	0.0504	4	0.0431	4
Gauss	IB1	0.1927	32	0.1888	9
	J48	0.1736	22	0.1530	9
	NNge	0.1763	31	0.1685	12
Mixed	IB1	0.0322	9	0.0330	10
	J48	0.0425	9	0.0449	11
	NNge	0.0639	10	0.0562	9
Sine1	IB1	0.0376	9	0.0377	9
	J48	0.0847	11	0.0637	10
	NNge	0.0819	14	0.0767	11

Table 2. Prequential error and total number of changes detected by methods in Sine1g dataset

		EDDM		DDM	
		Prequential	Drifts	Prequential	Drifts
Sine1g	IB1	0.1462	50	0.2107	12
	J48	0.1350	34	0.1516	12
	NNge	0.2104	28	0.2327	12

4.3 Results On Artificial Domains: Slow Gradual Drift

There are many real-world problems that have slow gradual drift. In this section we use the Sine1g dataset to illustrate how EDDM works with this kind of change. Figure 2 shows the prequential error curves obtained when EDDM and DDM deal with this dataset. The plots on the left are prequential error calculated with the whole dataset, and the plots on the right are prequential error calculated from scratch after every concept change detected by both methods.

Although the global curves are similar, the local prequential curves show that EDDM reacts before and more times than DDM when a slow concept drift is present. During the transition from the previous concept to the next, EDDM detects repeatedly concept drifts. This is so because the frequency of classification errors continuously increase until the next concept is stable. Meanwhile, DDM shows less sensitivity to this kind of problems, reacting later and less than EDDM. Table 2 presents the final prequential errors and number of drifts obtained by these two methods on Sine1g dataset.

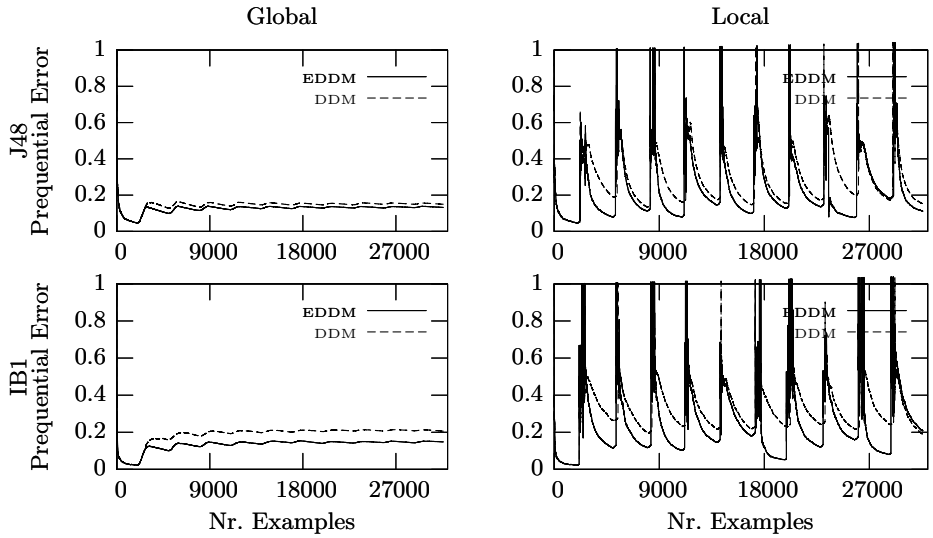


Fig. 2. Prequential global error (left) and local error (right) for EDDM and DDM in Sine1g dataset

4.4 The Electricity Market Dataset

The data used in this experiment was first described by M. Harries [14]. The data was collected from the Australian New South Wales Electricity Market. In this market, the prices are not fixed and they are affected by demand and supply of the market. A factor for the price evolution is the time evolution of the electricity market. During the time period described in the data the electricity market was expanded with the inclusion of adjacent areas. This produced a more elaborated management of the supply. The production surplus of one region could be sold in the adjacent region. A consequence of this expansion was a dampener of the extreme prices. The ELEC2 dataset contains 45312 instances dated from May 1996 to December 1998. Each example of the dataset refers to a period of 30 minutes. Each example on the dataset has 5 fields, the day of week, the time stamp, the NSW electricity demand, the Vic electricity demand, the scheduled electricity transfer between states and the class label.

The class label identifies the change of the price related to a moving average of the last 24 hours. The class level only reflect deviations of the price on a one day average and removes the impact of longer term price trends. The interest of this dataset is that it is a real-world dataset. We do not know when drift occurs or if there is drift. We have considered this problem as a short term prediction: predict the changes in the prices relative to the next 30 minutes.

In Figure 3 we present the traces of the prequential error rate of EDDM and DDM, with the base learning algorithms, through the full ELEC2 dataset. As

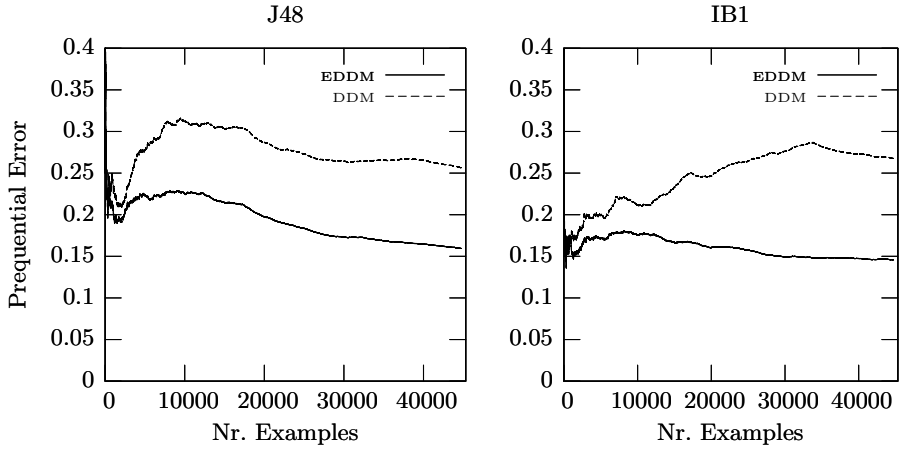


Fig. 3. Prequential error for EDDM and DDM in ELEC2 dataset

Table 3. Prequential error and total number of changes detected by methods in ELEC2 dataset

		EDDM		DDM	
		Prequential	Drifts	Prequential	Drifts
Elect2	IB1	0.1426	171	0.2764	44
	J48	0.1564	187	0.2123	10
	NNge	0.1594	193	0.2110	130

can be seen, EDDM outperforms DDM, detecting concept changes earlier and with a better sensitivity. Table 3 shows the final prequential errors and number of drifts obtained by these two methods on the electricity market dataset.

5 Conclusion

This paper introduces a new method for detecting concept drifts based on the distances between classification errors. This method achieves an early detection in presence of gradual changes, even when that change is very slow. The results that are presented have been obtained after using the method as a wrapper for different learning algorithms, but it would be easy to implement it in a local way inside those algorithms. The experimental evaluation of EDDM illustrates the advantages of using this detection method. As well as obtaining good results detecting concept drifts, this method shows itself as a way to deal with noisy datasets even when the base algorithm is not designed with that aim. When the base algorithm begins to overfit, the frequency of classification errors begins to increase and that is detected by the proposed method.

Out aim of improving EDDM involves some issues where the most important is finding a way to determine the values for the parameters of the method (α and β) in an automatic way.

References

1. Muthukrishnan, S.: Data streams: algorithms and applications. In: Proc. of the 4th annual ACM-SIAM symposium on discrete algorithms. (2003)
2. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. Lecture Notes in Computer Science **3171** (2004)
3. Dong, G., Han, J., Lakshmanan, L.V.S., Pei, J., Wang, H., Yu, P.S.: Online mining of changes from data streams: research problems and preliminary results. In: Proc. of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams. (2003)
4. Fan, W.: Streamminer: A classifier ensemble-based engine to mine concept-drifting data streams. In: Proc. of the 30th VLDB Conference. (2004)
5. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, ACM Press (2003) 226– 235
6. Cohen, E., Strauss, M.: Maintaining time-decaying stream aggregates. In: Proc. of the 21nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. (2003)
7. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. In: Proc. of the 17th Int. Conf. on Machine Learning. (2000) 487 – 494
8. Mitchell, T.: Machine Learning. McGraw Hill (1997)
9. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2 edn. Morgan Kaufmann, San Francisco (2005)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
11. Aha, D., Kibler, D.: Instance-based learning algorithms. Machine Learning **6** (1991) 37–66
12. Martin, B.: Instance-based learning : Nearest neighbor with generalization. Master's thesis, University of Waikato, Hamilton, New Zealand (1995)
13. Kubat, M., Widmer, G.: Adapting to drift in continuous domain. In: Proc. of the 8th European Conference on Machine Learning, Springer Verlag (1995) 307–310
14. Harries, M.: Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales (1999)
15. Dawid, A., Vovk, V.: Prequential probability: Principles and properties (1999)

Mining Frequent Items in a Stream Using Flexible Windows (Extended Abstract)

Toon Calders, Nele Dexters and Bart Goethals

University of Antwerp, Belgium
firstname.lastname@ua.ac.be

Abstract. In this paper, we study the problem of finding frequent items in a continuous stream of items. A new frequency measure is introduced, based on a flexible window length. For a given item, its current frequency in the stream is defined as the maximal frequency over all windows from any point in the past until the current state. We study the properties of the new measure, and propose an incremental algorithm that allows to produce the current frequency of an item immediately at any time. It is shown experimentally that the memory requirements of the algorithm are extremely small for many different realistic data distributions.

1 Introduction

Mining frequent items over a stream of items presents interesting new challenges over traditional mining in static databases. It is assumed that the stream can only be scanned once, and hence if an item is passed, it can not be revisited, unless it is stored in main memory. Storing large parts of the stream, however, is not possible because the amount of data passing by is typically huge.

Most previous work on mining frequently occurring items from a stream either focusses on (1) the whole stream, (2) on only the most recent items in a window of fixed length [1, 4, 6], or (3) where a time-decaying factor fades out the history of the stream [5]. In many applications, however, it is not possible to fix a window length or a decay factor that is most appropriate for every item at every timepoint in an evolving stream. For example, consider a large retail chain of which sales can be considered as a stream. Then, in order to do market basket analysis, it is very difficult to choose in which period of the collected data you are particularly interested. For many products, the amount of them sold depends highly on the period of the year. In summer time, e.g., sales of ice cream increase. During the world cup, sales of beer increase. Such seasonal behavior of a specific item can only be discovered when choosing the correct window size for that item, but this size can then also hide a similar behavior of other items. Therefore, we propose to consider for each item the window in which it has the highest frequency. More specifically, we define the current frequency of an item as the maximum over all windows from the past until the current state. The disadvantage of having a frequency of 100%, when the stream

ends with the particular item, can be resolved by setting a minimum window length all windows have to obey. Hence, when the stream evolves, the length of the window containing the highest frequency for a given item can grow and shrink continuously. We show some important properties on how the length of the maximal window can evolve.

In our approach, on every timestamp, a new item arrives in the stream. We present an incremental algorithm that maintains a small summary of relevant information of the history of the stream that allows to produce the current frequency of an item immediately at any time. That is, when a new item arrives, the summary is updated, and when at a certain point in time, the current frequency is required, the result can be obtained instantly from the summary. The structure of the summary is based on some critical observations about the windows with the maximal frequency. In short, many points in the stream can never become the starting point of a maximal window, no matter what the continuation of the stream will be. The summary will thus consist of some statistics about the few points in the stream that are still candidate starting points of a maximal window. These important points in the stream will be called the *borders*.

Critical for the usefulness of the technique are the memory requirements of the summary that needs to be maintained in memory. We show experimentally that, even though in worst case the summary depends on the length of the stream, for realistic data distributionsits size is extremely small. Obviously, this property is highly desirable as it allows for an efficient and effective computation of our new measure. Also note that our approach allows exact information as compared to many approximations considered in other works.

The organization of the paper is as follows. In Section 2, the new measure is defined and the problem is formally introduced. Section 3 gives the theoretical results for the basic theorem, on which the incremental algorithm in Section 4 is based. Section 5 contains experiments that show that the memory requirements for the algorithm are extremely small for many real-life data distributions.

2 New Frequency Measure in Stream Mining

In this section, we define our new frequency measure for streams and we formally introduce the problem. Throughout the paper, we assume that the items in the stream come from a finite set of items $\mathcal{I}$, unless explicitly mentioned otherwise.

2.1 Streams and Max-Frequency

A *stream* $\mathbb{S}$ is a sequence $\langle i_1, i_2, \ldots, i_n \rangle$ of items, where n is the *length* of the stream, denoted $|\mathbb{S}|$. The number of occurrences of an item i in the stream $\mathbb{S}$ will be denoted $\mathit{count}(i, \mathbb{S})$. The *frequency* of i in $\mathbb{S}$, is defined as

$$\mathit{freq}(i, \mathbb{S}) := \frac{\mathit{count}(i, \mathbb{S})}{|\mathbb{S}|} \quad .$$

The *concatenation* of m streams $\mathbb{S}_1, \ldots, \mathbb{S}_m$ is denoted $\mathbb{S}_1 \cdot \mathbb{S}_2 \cdot \ldots \cdot \mathbb{S}_m$. Every stream is glued at the end of the previous stream. Let $\mathbb{S} = \langle i_1, i_2, \ldots, i_n \rangle$. Then, $\mathbb{S}[s,t]$ denotes the *sub-stream* or *window* $\langle i_s, i_{s+1}, \ldots, i_t \rangle$. The sub-stream of $\mathbb{S}$ consisting of the last k items of $\mathbb{S}$, denoted $last(k, \mathbb{S})$, is

$$last(k, \mathbb{S}) = \mathbb{S}\big[|\mathbb{S}| - k + 1, |\mathbb{S}|\big] \ .$$

We are now ready to define our new frequency measure:

Definition 1. *The* max-frequency *$mfreq(i, \mathbb{S})$ of an item i in a stream $\mathbb{S}$ is defined as the maximum of the frequency of i over all windows extending from the end of the stream; that is:*

$$mfreq(i, \mathbb{S}) \ := \ \max_{k=1\ldots|\mathbb{S}|} (freq(i, last(k, \mathbb{S}))) \ .$$

This $mfreq(i, \mathbb{S})$ is used as a new frequency measure for stream mining. For a given item, its current frequency in the stream is defined as the maximal frequency over all evolving windows from the end to the beginning of the stream.

The longest window in which the max-support is reached, is called the maximal window for i in $\mathbb{S}$*, and its starting point is denoted $maxwin(i, \mathbb{S})$. That is, $maxwin(i, \mathbb{S})$ is the smallest index such that*

$$mfreq(i, \mathbb{S}) \ = \ freq(i, \mathbb{S}\big[maxwin(i, \mathbb{S}), |\mathbb{S}|\big]) \ . \qquad \square$$

Note that, by definition, the max-frequency of an item a in a stream that ends with an a, is always 100%, independently of the overall frequency of a. Hence, even in a stream where a is extremely rare, at some points, the max-frequency will be maximal! This disadvantage of max-frequency, however, can easily be resolved by either considering streams of blocks of items instead of single items, or by setting a minimal length all windows must obey. We did not include these solutions in the paper, as they are not fundamental for the theory developed.

Example 1. We focus on target item a.

$$\begin{aligned} mfreq(a, abaaab) &= \max_{k=1\ldots6} (freq(a, last(k, abaaab))) \\ &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = \frac{3}{4} \ . \\ mfreq(a, bcdabcda) &= \max\left(\frac{1}{1}, \cdots\right) = 1 \ . \\ mfreq(a, xaaxaax) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = \frac{2}{3} \ . \end{aligned}$$

Notice that our definition of frequency is quite different from the usual approaches, where the frequency of an item i in a stream $\mathbb{S}$ is either defined as $freq(i, last(wl, \mathbb{S}))$ for a fixed window length wl, or with a time-decaying factor; e.g., $freq(i, \mathbb{S}) = \sum_{j=1}^{|\mathbb{S}|/wl} d^j freq(i, \mathbb{S}[(j-1)wl + 1, jwl])$, with $d < 1$.

2.2 Problem Statement

Notice that we defined a stream as a statical object. In reality, however, a stream is an evolving object. At every timepoint, a new item might be inserted at the end of the stream. As such, evolving streams are essentially unbounded, and when processing them, it is to be assumed that only a small partcan be kept in memory.

In our examples, new entrieswill be added on the right side of the stream. This means that the older items are on the left side of the stream. For simplicity we assume that the first item arrived at timestamp 1, and since then, at every timestamp a new item was inserted. $\mathbb{S}_t$ denotes the stream up to timestamp t.

The problem we study in the paper is the following: *For an evolving stream* $\mathbb{S}$ *and a fixed item* a*, maintain a small summary of the stream in time, such that, at any timepoint* t*,* $mfreq(a, \mathbb{S}_t)$ *can be produced instantly from the summary.*

More formally, we will introduce a summary of a stream $summary(\mathbb{S})$, an update procedure *Update*, and a procedure *Get_mfreq*, such that, if we assume that on timestamp $t+1$ item i is added to the stream, $Update(summary(\mathbb{S}_t), i)$ equals $summary(\mathbb{S}_t \cdot \langle i \rangle)$ equals $summary(\mathbb{S}_{t+1})$, and $Get\_mfreq(summary(\mathbb{S}_{t'}))$ equals $mfreq(a, \mathbb{S}_{t'})$. Because *Update* has to be executed every time a new item arrives, it has to be extremely efficient, in order to be finished before the next item arrives. Similarly, because the stream continuously grows, the summary must be independent of the number of items seen so far, or, at least grow very slowly as the stream evolves. The method we develop will indeed meet these criteria, as the experiments will show.

stream	time stamp	$mfreq(a, \mathbb{S}_t)$
|a	1	$\max(\frac{1}{1}) = 1$
|aa	2	$\max(\frac{1}{1}, \frac{2}{2}) = \frac{2}{2} = 1$
|aaa	3	$\max(\frac{1}{1}, \frac{2}{2}, \frac{3}{3}) = \frac{3}{3} = 1$
|aaab	4	$\max(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}) = \frac{3}{4}$
|aaabb	5	$\max(\frac{0}{1}, \frac{0}{2}, \frac{1}{3}, \frac{2}{4}, \frac{3}{5}) = \frac{3}{5}$
|aaabbb	6	$\max(\frac{0}{1}, \frac{0}{2}, \frac{0}{3}, \frac{1}{4}, \frac{2}{5}, \frac{3}{6}) = \frac{3}{6}$
aaabbb|a	7	$\max(\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}, \frac{3}{6}, \frac{4}{7}) = 1$
aaabbb|aa	8	$\max(\frac{1}{1}, \frac{2}{2}, \frac{2}{3}, \frac{2}{4}, \frac{2}{5}, \frac{3}{6}, \frac{4}{7}, \frac{5}{8}) = \frac{2}{2}$
...	...	...

Fig. 1. Max-frequency of a stream at every timepoint.

In Fig. 1, the max-frequency has been given for an example evolving stream. The starting point $maxwin(a, \mathbb{S})$ of each maximal window is marked with |.

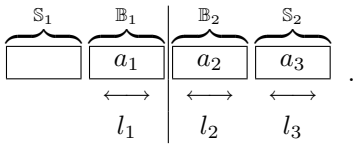
3 Properties of Max-Frequency

In this section we show some properties of max-frequency that are crucial for the incremental algorithm that maintains the summary of the stream. Obviously,

checking all possible windows to find the maximal one is infeasible algorithmically, given the constraints of stream problems. Fortunately, not every point in the stream needs to be checked. The theoretical results from this section show exactly which points need to be inspected. These points will be called the *borders* in the stream. The summary of the stream will consist exactly of the recording of these borders, and the frequency of the target item up to the most recent timepoint.

Theorem 1. *Consider a stream* $\mathbb{S}_1 \cdot \mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{S}_2$. *If* $\mathbb{B}_2 \cdot \mathbb{S}_2$ *is the maximal window for* a *in* $\mathbb{S}$, *then* $freq(a, \mathbb{B}_1) < freq(a, \mathbb{B}_2)$

Proof. If $\mathbb{B}_2 \cdot \mathbb{S}_2$ is the maximal window for a in $\mathbb{S}$, then this implies that the frequency of a in $\mathbb{B}_2 \cdot \mathbb{S}_2$ is strictly higher than in $\mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{S}_2$ and at least as high as in $\mathbb{S}_2$ (remember that in the case of multiple windows with maximal frequency, the largest one is selected). Let now $l_1 = |\mathbb{B}_1|$, $l_2 = |\mathbb{B}_2|$, and $l_3 = |\mathbb{S}_2|$, and let $a_1 = count(a, \mathbb{B}_1)$, $a_2 = count(a, \mathbb{B}_2)$, and $a_3 = count(a, \mathbb{S}_2)$, as depicted in:



Then, the conditions on the frequency translate into:

$$\frac{a_2 + a_3}{l_2 + l_3} > \frac{a_1 + a_2 + a_3}{l_1 + l_2 + l_3} \quad \text{and} \quad \frac{a_2 + a_3}{l_2 + l_3} \geq \frac{a_3}{l_3}.$$

From these conditions, it can be derived that

$$freq(a, \mathbb{B}_1) = \frac{a_1}{l_1} < \frac{a_2}{l_2} = freq(a, \mathbb{B}_2). \qquad \square$$

Based on this theorem, it is possible to give an exact characterization of which points in $\mathbb{S}_t$ can potentially become the starting point of the maximal window at a future point in time, after new items have been added. The next corollary gives this characterization.

We now formally define the important notion of a *border*. Intuitively, a border is a point in the stream that can still become the starting point of the maximal window.

Definition 2. *The position* q *in* $\mathbb{S}$ *is called a* border for a in $\mathbb{S}$ *if there exists another stream* $\mathbb{B}$ *such that* $q = maxwin(a, \mathbb{S} \cdot \mathbb{B})$.

Corollary 1. *Let* $\mathbb{S}$ *be a stream, and let* $q = 1$. *Position* q *is a border for item* a *if and only if the first item in the stream equals the target item* a.

Let $\mathbb{S}$ *be a stream, and let* $2 \leq q \leq |\mathbb{S}|$. *Position* q *is a border for item* a *in* $\mathbb{S}$ *if and only if for all indices* j, k *with* $1 \leq j < q$ *and* $q \leq k \leq |\mathbb{S}|$, *it holds that* $freq(a, \mathbb{S}[j, q-1]) < freq(a, \mathbb{S}[q, k])$.

Proof. **Only if:** Suppose that there exist indices j and k, and a stream $\mathbb{B}$ such that $freq(a, \mathbb{S}[j, q-1]) \geq freq(a, \mathbb{S}[q, k])$, and $q = maxwin(a, \mathbb{S} \cdot \mathbb{B})$. This situation is in contradiction with Theorem 1: split the stream $\mathbb{S} \cdot \mathbb{B}$ as $(\mathbb{S}[1, j-1]) \cdot (\mathbb{S}[j, q-1]) \cdot (\mathbb{S}[q, k]) \cdot (\mathbb{S}[k+1, |\mathbb{S}|] \cdot \mathbb{B})$. In this stream, $(\mathbb{S}[q, |\mathbb{S}|]) \cdot \mathbb{B}$ is the maximal window, while $freq(a, \mathbb{S}[j, q-1]) \geq freq(a, \mathbb{S}[q, k])$.

If: It can be shown that the item at timepoint q must be the target item a. If enough non-target items are added to the stream $\mathbb{S}$, eventually q will become the starting point of the maximal window. The full proof of this part, however, is omitted due to space restrictions. □

Example 2. Assume we have the following stream $\mathbb{S}_{27}$:

4/9		4/10		2/3	1/2	
aaabbbabb	|	ababababbb	b	aab	| ab	ba

In the stream, two positions have been marked with |. Both these points do not meet the criteria given in Corollary 1 to be a border. Indeed, for both positions, a block before and after it is indicated such that the frequency in the before-block is higher than in the after-block. In this stream, the only positions that meet the requirement are indicated in $aaabbbabbababababbbb|aababb|a$. □

The following simple facts play an important role in the algorithm that will be developed in the next section.

Corollary 2. *Every border always ends on a target item. If p is not a border in $\mathbb{S}$, then neither it can ever be a border in any extension $\mathbb{S} \cdot \mathbb{B}$.*

4 Algorithm

Based on the theorems of Section 3, we now present an incremental algorithm to maintain a summary.

The summary. The summary for an item a in the stream $\mathbb{S}_t$ is the array of that contains a pair $(p, x/y)$ for every border on position p, with x the number of occurrences of a since p, i.e., $count(a, \mathbb{S}_t[p, t])$, and y the length of the block from p until the end of the stream $\mathbb{S}_t$, i.e., $t-p+1$. The output of the algorithm in the case of r borders is written as an array of the form $[(p_1, x_1/y_1), \cdots, (p_r, x_r/y_r)]$, visualized by

$$T_t = \begin{array}{|c|c|c|} \hline p_1 & \cdots & p_r \\ \hline x_1/y_1 & \cdots & x_r/y_r \\ \hline \end{array} \ .$$

This array is in fact $summary(\mathbb{S}_t)$, and is abbreviated by T_t. In this array, the border positions are ordered from old to most recent, reflecting in $p_1 < \cdots < p_r$. The corresponding frequencies must follow the same ordering $x_1/y_1 < \cdots < x_r/y_r$; indeed, consider two consecutive borders p_i and p_{i+1}. Suppose for the sake of contradiction, that $x_i/y_i \geq x_{i+1}/y_{i+1}$. From this, it follows that

$$freq(a, \mathbb{S}[p_i, p_{i+1}-1]) = \frac{x_i - x_{i+1}}{y_i - y_{i+1}} \geq \frac{x_i}{y_i} = freq(a, \mathbb{S}[p_{i+1}, |\mathbb{S}|]) \ .$$

According to Theorem 1 this implies that p_{i+1} cannot be a border because the frequency of a in a before-block is at least the frequency of a in an after-block for p_{i+1}. *Notice that this implies that the current frequency can always be obtained immediately from the summary; the most recent entry in the summary is always the largest and thus holds the current max-support.*

In every next step, the algorithm adjusts the stored values in the array based on the newly entered item in the stream. Hence, at every step, we need to test for every border of $\mathbb{S}_t$ if it is still a border in $\mathbb{S}_{t+1}$. Hence, we need to check if still the frequency in all before blocks is smaller than the frequency in all after blocks. *However, adding a new item to the stream does not introduce a new before-block, and only one after-block!* Hence, only one test has to be performed for every border of $\mathbb{S}_t$: the before-block with the highest frequency of a has to be compared to the new after-block. The frequency of the new after-block for border p_i can easily be obtained from the pair $(p_i, x_i/y_i)$ in the summary of $\mathbb{S}_t$: if the new item is a non-target item, the frequency of a in the new after-block is $x_i/(y_i+1)$. *Notice that, as the before block never changes when a new item enters the stream, and the insertion of the target item a only results in an increased frequency in the new after block, the addition of a target-item will never result in the removal of borders.*

Based on the observation that in any summary the borders must be in order w.r.t. the frequency of a, it is not too hard to see that the before-block with the maximal frequency is exactly the block $\mathbb{S}_t[p_{i-1}, p_i - 1]$. Via a similar reasoning as above, it follows that p_i is still a border for $\mathbb{S}_{t+1}$ if and only if the updated frequency $x_i/(y_i + 1)$ is still larger than the updated frequency $x_{i-1}/(y_i + 1)$ of p_{i-1}. To summarize, we have the following properties:

- The frequencies in the summary are always increasing.
- When the target item is added to the stream, all borders remain borders. The frequencies of the borders can be updated by incrementing all nominators and denominators by 1. A new entry with the current timestamp and frequency 1/1 can be added, unless the last entry has also 100% frequency.
- If a non-target item enters the stream, the frequencies of the borders can be updated by adding 1 to the denominators of the frequencies. All former borders for which after the update, the frequency no longer is larger than in the previous entry, are no borders anymore.

Algorithm 1 is based on these observations.

The Algorithm. Before the first target item enters the stream, the array will remain empty. The pseudo-code of the algorithm to create T_{t+1}, based on T_t and the item t that enters the stream at time $t + 1$ is given in Algorithm 1. In short, when a new item i enters the stream, the frequencies are updated by increasing the nominators if i equals the target item, and always increasing the denominators. If the item i is the target item, a new border will be added only if the frequency of the last block in T_t was not equal to 1. Furthermore, we have to take into account that some of the borders of $\mathbb{S}_t$ might no longer be borders in $\mathbb{S}_{t+1}$. This can only happen if the item that enters the stream is a non-target

Algorithm 1 $Update(T_t, i)$ for target item a on time $t+1$

Require: $T_t = summary(\mathbb{S}_t) = [(p_1, x_1/y_1), \cdots, (p_r, x_r/y_r)]$
Ensure: $T_{t+1} = summary(\mathbb{S}_{t+1}) = summary(\mathbb{S}_t \cdot \langle i \rangle)$

```
1: Set T_{t+1} := [ ]
2: if (T_t is empty) then
3:    if (i = target item a) then
4:       T_{t+1} := [(t+1, 1/1)]
5: else
6:    if (i = target item a) then
7:       for 1 ≤ j ≤ r do
8:          T_{t+1} := T_{t+1} + (p_j, (x_j + 1)/(y_j + 1))
9:       if x_r ≠ y_r then
10:         T_{t+1} := T_{t+1} + (t+1, 1/1)
11:   else
12:      high := 0
13:      for all j := 1...r do
14:         if (x_j)/(y_j + 1) > high then
15:            T_{t+1} := T_{t+1} + (p_j, (x_j)/(y_j + 1))
16:            high := (x_j)/(y_j + 1)
```

item, and is tested in lines 12-16: the frequencies have to increase for increasing positions of the borders.

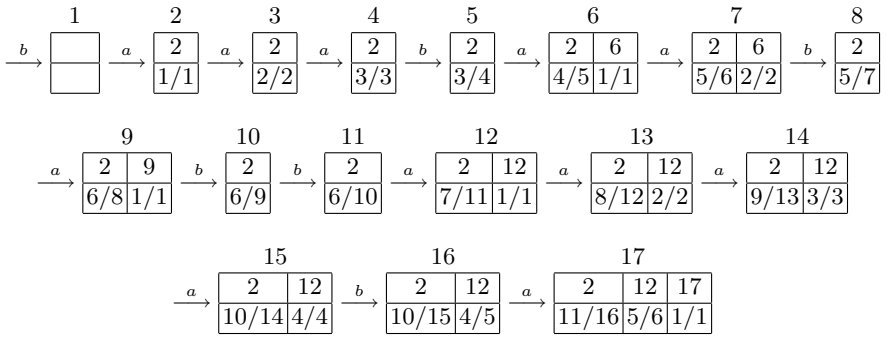
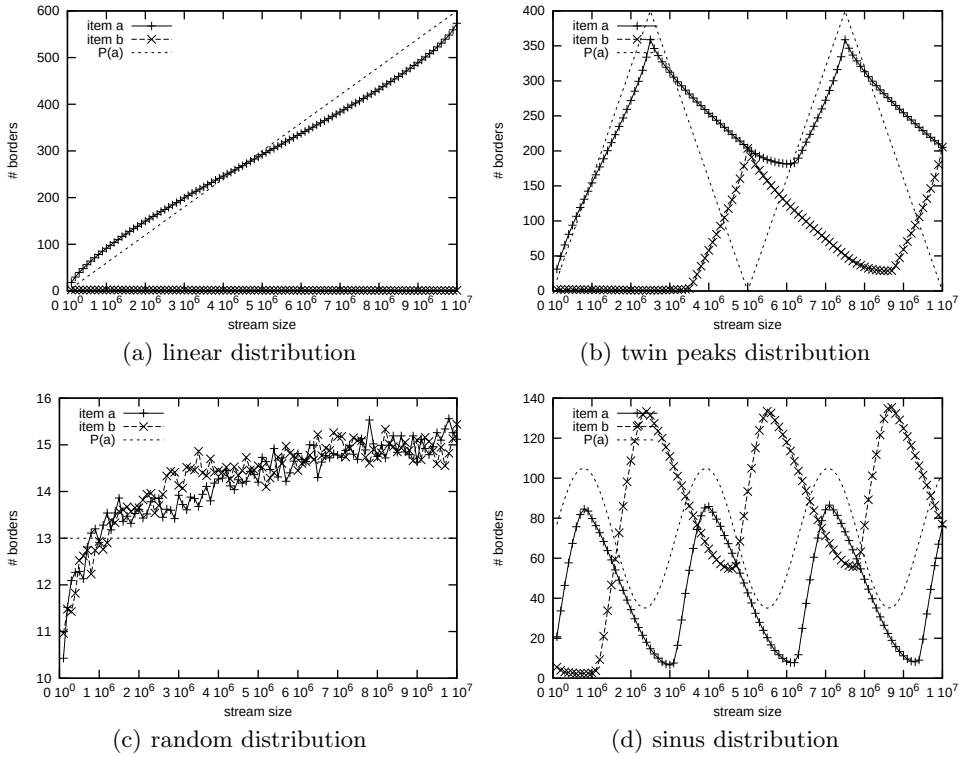


Fig. 2. Example for stream *baaabaababbaaaaba*.

We explain the working of the algorithm for the stream *baaabaababbaaaaba*. For each timestamp, the output of the algorithm is given in Figure 2.

In this example, some interesting things happen. First of all, the stream starts with a junk item b. Therefore, $Update(T_0, b) = Update([\,], b)$ on timestamp 1 remains empty, i.e., $T_1 = [\,]$. The algorithm in fact really starts at timestamp 2. At this moment, $Update([\,], a)$ results in $T_2 = [(2, 1/1)]$, corresponding to the stream $b|a$ with a border at position 2. On timestamp 8, another interesting fact



(a) linear distribution (b) twin peaks distribution

(c) random distribution (d) sinus distribution

Fig. 3. Size of the summaries for two items a and b

happens. $T_7 = [(2, 5/6), (6, (2/2))]$, corresponding with the stream $b|aaab|aa$. $Update(T_7, b)$ will yield $T_8 = [(2, 5/7)]$, and not $[(2, 5/7), (6, 2/3)]$, because the frequency decreases from the border at position 2 to the border at position 6, and hence, we can conclude that position 6 is no longer a border.

5 Experiments

From the description of the algorithm it is clear that the update procedure is very efficient, given the summaries remain small. Producing the current support of the target item is obviously very efficient, as it amounts to simply a lookup in the most recent entry. Hence, the complete approach will be feasible if and only if the summaries remain small. Therefore, for different streams, we have recorded the size of the summary. The results are reported in Figure 3. The streams we consider are over the items a and b, and have length 10^7. After every 10 000 items, the size of the summary for the items a and b are reported. The streams are randomly generated. The probability of having item a in the stream is given by the line $P(a)$. Thus, in the random graph, the probability of having a is $1/2$ in the whole stream, independent of the moment. The probability of b is 1 minus

the probability of a. The graphs report the average over 100 streams, generated with the indicated distributions. In general, we can conclude that the size of the summary is extremely small w.r.t. the size of the stream. If the probability of the target item increases, also the size of the summary will increase, when the probability decreases, the summary will shrink. This is easily explained by the entries in the summary that need to have increasing frequency.

6 Conclusion and Future Work

We presented a new frequency measure for items in streams that does not rely on a fixed window length or a time-decaying factor. Based on the properties of the measure, an algorithm to compute it was shown. An experimental evaluation supported the claim that the new measure can be computed from a summary with extremely small memory requirements, that can be maintained and updated efficiently.

In the future, we will look at the same topic, but try to mine for frequent itemsets instead of items, based on this new frequency measure.

References

1. Ruoming J. and Agrawal G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. in Proc. 5th IEEE Int. Conf. on Data Mining (ICDM'05), pp 210–217.
2. Agrawal R., Imielinski T. and Swami A.: Mining Association Rules between Sets of Items in Large Databases. in Proc. ACM SIGMOD Int. Conf. on Management of Data (1993), pp 207–216.
3. Cormode G. and Muthukrishnan S.: What's Hot and What's Not: Tracking Most Frequent Items Dynamically. in Proc. PODS (2003).
4. Demaine E.D., Lopez-Ortiz A. and Munro, J.I.: Frequency Estimation of Internet Packet Streams with Limited Space. in Proc. of the 10th Annual European Symposium on Algorithms (2002), pp 348–360.
5. Giannella C., Han J., Robertson E. and Liu C.: Mining Frequent Itemsets Over Arbitrary Time Intervals in Data Streams. Technical Report TR587 at Indiana University, Bloomington, (Nov 2003), 37 pages.
6. Karp, R. M., Papadimitriou, C. H. and Shenker, S.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. in ACM Trans. on Database Systems (2003), 28, pp 51–55.
7. Lee, L.K. and Ting, H.F.: A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. in ACM PODS (2006).
8. Misra J. and Gries, D.: Finding Repeated Elements. in Science of Computer Programming (1982), 2(2), pp 143–152.
9. Chi-Wing Wong R. and Wai-Chee Fu A.: Mining Top-K Frequent itemsets from Data Streams. in Data Mining and Knowledge Discovery. to appear 2006

Hard Real-time Analysis of Two Java-based Kernels for Stream Mining

Rasmus Pedersen

Dept. of Informatics, Copenhagen Business School, Copenhagen, Denmark
rup.inf@cbs.dk

Abstract. Embedded real-time data stream mining is an important branch of data mining. We propose and provide some fundamental prerequisites for using statistical pattern recognition in stream-based learning/decision systems in this paper: namely some real-time kernel functions and a real-time analysis of the support vector machine's decision function. In this paper we analyze two kernels and a support vector machine on a real-time Java processor to provide upper bounds (worst cases) for execution time analysis.

1 Introduction

Embedded machine learning is important: One reason is the mere size of the embedded chip market, which may account for 98% of all processors sold [1]. In this paper we combine a worst case execution time analysis (a key field in real-time embedded systems research) with a method for building sparse kernel learning algorithms (a recent method for direct sparcity control of a kernel learning algorithm) by Wu et al.[2].
The support vector machine (SVM) is formulated within Vladimir Vapnik's framework of statistical learning theory [3]. Later, the SVM is extended by Cortes and Vapnik to cover binary classification problems with misclassifications [4]. One particularly significant discovery in terms of enabling the use of SVMs in embedded systems [5] is due to John Platt [6]: using the sequential minimal optimization (SMO) method, we are able to train SVMs using an insignificant memory footprint, which can be mandatory on embedded systems. Later, the SVM is extended beyond classification to other classic machine learning frameworks such as non-linear regression, novelty detection, and clustering [7] [8] [9].
Java-based data mining and machine learning packages exist: such as WEKA [10], YALE [11] [12], or the Java version of LIBSVM [13]. Furthermore, recent developments in real-time Java for embedded systems makes WCET analysis possible [14]. Some relevant applications of distributed data mining is also presented in [15].
An implicit contribution by this paper is that researchers already familiar with machine learning in Java can migrate and analyze many learning algorithms by following the same steps using the open source WCET tool (see package `com.jopdesign.wcet`)[1] based on the Byte Code Engineering Library [16].

[1] see http://www.opencores.net/projects.cgi/web/jop/overview

The paper is organized as follows. In Section 1.1 we discuss the WCET properties of the target processor for this paper. In Section 1.2, it is presented how the advancements of Wu et al. [2] can be combined with WCET analysis. Then we provide WCET experimentation with a departure in Java bytecodes and end up with the parameterized SVM formulas depicting WCET for standard SVMs in Section 2. We also provide some benchmark analysis in Section 2.3 of selected classification sets. We conclude the paper with some pointers to future research directions.

1.1 Java Optimized Processor

The Java Optimized Processor (JOP [14]) is a field programmable gate array-based (FPGA) implementation of the Java Virtual Machine. One design goal for JOP is its applicability to worst-case execution time (WCET) analysis. This design principle is consistent throughout the JOP processor. In particular, this makes it possible to analyze certain data mining algorithms with respect to their WCET properties. As the processor is implemented in an FPGA and all sources are available, it is also possible to add specialized hardware accelerators to this processor [17].

1.2 SVMs for real-time environments

The binary classification SVM provides a decision function: $f(\boldsymbol{x}, \boldsymbol{\alpha}, b) = \{\pm 1\} = sgn\left(\sum_{i=1}^{l} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b\right)$. $\boldsymbol{\alpha}$ is the result of solving the following optimization problem:
$maximize\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{l} y_i \alpha_i = 0$.
The functional output of the decision function works as a classification or categorization of the unknown datum $\boldsymbol{x}$ into either the $+$ or $-$ class. An SVM model is constructed by summing a linear combination of training data (historical data) in feature space. The feature space is implicitly constructed by the use of kernels, k, resulting in a non-linear classifier in the input space. The optimization problem is constructed by enforcing a margin of 1 for the support vectors. Support vectors (SV), are those data $\boldsymbol{x}_i$, which have active constraints, $\alpha_i > 0$.
The idea of using SVMs in distributed and embedded environments–ie. constrained environments–have been argued by Pedersen [5]. In addition, a recent publication by Wu, Schölkopf, and Bakir [2] introduce a method to directly control the number of expansion vectors N_{XV}. It simplifies the task of using kernel based learning algorithms in computationally constrained environments. The reason is that it allows the design and constraints of other parameters such as battery lifetime, real-time schedulability, or HW/SW codesign issues to prescribe the requirements of the sparse kernel learning algorithm. We will see in Table 7 that even a 90% reduction in support vectors generate modest increases of the average error by $\sim$0.1%.

2 WCET Analysis

In this section we analyze the linear kernel and the SVM decision formulas in detail. Later, in Section 2.3, we include the popular gaussian kernel in the analysis. The output from this section is a formula for describing the worst-case-execution-time (WCET) of these two critical components in a kernel learning system. The work is general to the extent that others can apply the same steps to analyze similar systems or similar algorithms.

2.1 Linear Kernel Analysis for Real-time Application

The linear kernel is composed of a dot product between two n-dimensional vectors. We use a 32-bit fixed-point representation, which mandates a 16-bit right shift as part of the multiplication of two 32-bit numbers. In this paper, we do not discuss fixed point scaling. Instead, we settle for an 8-bit right-shift operation on each operand. In Figure 1 the Java kernel is shown, which implements this scheme[2].

```
private static int kernelDot(int i1, int i2) {
    int r;
    int n = KFP.n;
    int a[] = data[i1];
    int b[] = data[i2];
    r = 0;
    while (n != 0) { //@WCA loop=2
        n = n - 1;
        r += (a[n] >> 8) * (b[n] >> 8);
    }
    return r;
}
```

Fig. 1. Linear dot product kernel Java method

The linear kernel in Figure 1 is compiled to Java bytecodes [18] as is shown in Table 1, which forms the basis of the WCET analysis. To deal with the bytecodes, we have divided them into basic blocks [19]. Each block becomes a node in a directed control flow graph. The basic block boundaries are created from a static analysis of the compiled Java class files. A boundary exists on both conditional and unconditional branch instructions and their targets. First, the basic blocks: In $B0$ the static variable $KFP.n$ is assigned to a local variable n, which denotes the dimension of the training vectors. An examination of the cycle count reveals the explanation for this: it takes 16 cycles to retrieve the static

[2] see http://sourceforge.net/projects/dsvm

variable $KFP.n$ and since we will branch (ie. use the variable in a loop) it saves CPU cycles to do this optimization. See `ifeq` at addr. 21 in Table 1. Accordingly, we save cycles in those cases where the data sets are multi-dimensional. The local variable n is loaded onto the stack in 1 cycle as opposed 16 cycles for the static counterpart. A similar approach is taken regarding the two vectors subject to the dot product. Each one is loaded into a local array: $a[]$ and $b[]$. It is a result of Figure 1 that basic block $B2$ is the most frequently used block of code for this method. It decrements the dimension index n and accumulates the dot product into the `int` r. It should be noted that a sparse vector dot product can be implemented as well, which would only perform the multiplication if both inputs are non-zero.

Table 1. Linear kernel with machine cycle mappings

Block	Addr.	Bytecode	Cycles	Misc.
B0	0:	getstatic[178]	16	int dsvmfp.model.smo.kernel.KFP.n
	3:	istore_3[62]	1	->I:n
	4:	getstatic[178]	16	int[][] dsvmfp.model.smo.kernel.KFP.data
	7:	iload_0[26]	1	I:i1
	8:	aaload[50]	29	Ljava/lang/Object;
	9:	astore[58]	2	->[I:a
	11:	getstatic[178]	16	int[][] dsvmfp.model.smo.kernel.KFP.data
	14:	iload_1[27]	1	I:i2
	15:	aaload[50]	29	Ljava/lang/Object;
	16:	astore[58]	2	->[I:b
	18:	iconst_0[3]	1	
	19:	istore_2[61]	1	->I:r sum(B0):115/115
B1	20:	iload_3[29]	1	I:n
	21:	ifeq[153]->49:	4	sum(B1): 5/5
B2	24:	iload_3[29]	1	I:n
	25:	iconst_1[4]	1	
	26:	isub[100]	1	
	27:	istore_3[62]	1	->I:n
	28:	iload_2[28]	1	I:r
	29:	aload[25]	2	[I:a
	31:	iload_3[29]	1	I:n
	32:	iaload[46]	29	I
	33:	bipush[16]	2	
	35:	ishr[122]	1	
	36:	aload[25]	2	[I:b
	38:	iload_3[29]	1	I:n
	39:	iaload[46]	29	I
	40:	bipush[16]	2	
	42:	ishr[122]	1	
	43:	imul[104]	35	
	44:	iadd[96]	1	
	45:	istore_2[61]	1	->I:r
	46:	goto[167]->20:	4	sum(B2):116/116
B3	49:	iload_2[28]	1	I:r
	50:	ireturn[172]	19	sum(B3): 20

An further analysis of the linear kernel in Figure 1 is performed by creating the directed control flow graph of basic blocks. As noted previously, the basic block $B2$ is the one where the multiplication/addition is performed. $B2$ is executed n times: one time for each dimension in the data vectors. The branching block, $B1$, is executed $n + 1$ times. This information is captured by the directed graph of basic blocks in Table 2. The directed graph is parameterized on the dimensionality of the problem: ie. the directed graph is $B0 + (n + 1)B1 + nB2 + B3$ and the result is shown on the bottom line in Table 3.

The directed graph is described by the parameter n, which allows us to describe a maximum and minimum execution time for the dot product

Table 2. Directed graph of basic blocks for linear kernel

	B0	B1	B2	B3
B0	.	1	.	.
B1	.	.	n	1
B2	.	n	.	.
B3	.	.	.	.

kernel. A result of this analysis is presented in Table 3. Thus, the minimum cycle count for $n = 1$ is 261 cycles (assuming that the `kernelDot` method is the cache. Cache analysis is not the subject of this paper).

Table 3. Dot product cycle count analysis of basic blocks based on n

		Max. cycles(n)		Min. cycles(n=1)	
Block	Cycles	Count	Total	Count	Total
B0	115	1	115	1	115
B1	5	n+1	5n+5	2	10
B2	116	n	116n	1	116
B3	20	1	20	1	20
Execution cycle count: 121n+140					261

The SVM decision function uses a kernel function to return the functional output of a given test vector. To this end it uses a bias and sums up the kernel outputs of the test vector and the support vectors. As the kernel is now analyzed, we can proceed to the SVM decision function and discuss the worst-case-execution-time (WCET) and best-case-execution-time (BCET). In Figure 2, the method `getFunctionOutputTestPointFP` takes a test vector reference as argument and returns a fixed-point representation of the functional output of the SVM decision function. Similar to (and for the same reasons as before in Figure 1), we re-assign static references to a local variables for performance reasons. As can be seen in Figure 2, the local variable `functionalOutput_fp` accumulates the dot product using the same 8 bit pre-shifting of the two scalars. It can be seen that the method is inlined such that there are not any invocations of other methods inside the `getFunctionOutputTestPointFP` method. There are several different ways to construct the details of the decision method. In this analysis we will focus on the code in Figure 2 and replace the multiply and add code with a method call outside the current method. As a further step in analyzing the SVM decision function, we must get access to the annotated bytecode with instruction cycle counting, basic block analysis and partial sums of basic block cycle counts.
Using the information in Table 5, we can see that the SVM decision function translates into 7 basic blocks. The analysis of the directed graph associated with the basic blocks is important because there are now two

```
1 :  static public int getFunctionOutputTestPointFP(int[] xtest) {
2 :    int functionalOutput_fp = 0;
3 :    int[][] data_fp_local = data_fp;
4 :    int m = data_fp_local.length;
5 :    int n = xtest.length;
6 :    for (int i = 0; i < m; i++) { //@WCA loop=(data points)
7 :      if (alpha_fp[i] > 0) {
8 :        while (n != 0) { //@WCA loop=(dimensionality)
9 :          n = n - 1;
10:          //functionalOutput_fp += KFP.kernelX(i); // cached
11:          functionalOutput_fp +=
12:            (data_fp_local[m][n] >> 8) * (xtest[n] >> 8);
13:        }
14:      }
15:    }
16:    functionalOutput_fp -= bias_fp;
17:    return functionalOutput_fp;
18:  }
```

Fig. 2. SVM decision function in Java

conditions in the decision function: one is that we do not multiply if the Lagrange multiplier α_i is zero (see $B2$ addr.: 28) and the second is that the loop runs over the training examples (see $B3$ addr.: 33). The directed graph allows for the cycle count estimate to be parameterized on the number of support vectors: sv:$\forall \alpha_i > 0$.

2.2 Real-time analysis of SVM decision function

An analysis the directed graph (same approach as in Table 2) of the SVM decision function shows that basic block $B1$, $B2$, and $B3$ have conditional branches. $B1$ is conditioned on the number of training examples, $B2$ is conditioned on a support vector being non-zero, and $B3$ is conditioned on the dimension of the training problem, n.

Since we have inlined (ie. no outgoing method calls) the SVM decision function allows us to perform a WCET (they are equal if $\forall \alpha_i > 0$) analysis of the SVM decision function parameterized only on m, n and, sv, the number of support vectors without regard to the cache. The directed graph is annotated with this information. Accordingly, it is then possible to construct the table for the SVM decision function, which then depicts the formula for cycle count based on m, n, and sv. This time (as opposed to Table 3) the min. cycle count is omitted.

The SVM decision function is an important part of the code, and we provide an abbreviated bytecode analysis and basic block information in Table 5.

Table 4. Parameterized cycle count analysis of inlined SVM

Block	Cycles	Count	Total
B0	43	1	43
B1	7	m+1	7m+7
B2	51	m	51m
B3	6	$sv(1+n)$	$6sv(1+n)$
B4	148	$sv\,n$	$148sv\,n$
B5	15	m	15m
B6	39	1	39
Execution cycle count: $89 + 73m + 6sv + 154sv\,n$			

Table 5. Bytecodes of the SVM decision function

```
Approach Addr. Bytecode       Cycles Misc.
...
B2          22:   getstatic[178]   16 int[] SMOBinaryClassifierFP.alpha_fp
            25:   iload[21]         2 I:i
            27:   iaload[46]       29 I
            28:   ifle[158]->65:    4 sum(B2): 51/51
B3          31:   iload[21]         2 I:n
            33:   ifeq[153]->65:    4 sum(B3): 6/6
...
```

Table 6. Inline vs. method call in SVM decision function

```
Block Addr. Bytecode          Cycles Misc.

Object-oriented SVM decision function,
Java: functionalOutput_fp += KFP.kernelX(i);
      42:   iload_1[27]             1 I:functionalOutput_fp
      43:   iload[21]               2 I:i
      45:   invokestatic[184] 91/109 kernelX(I)I, invoke(n=7):72/72 return(n=17):19/37
      48:   iadd[96]                1
      49:   istore_1[60]            1 ->I:functionalOutput_fp

Inlined SVM decision function,
Java: functionalOutput_fp += (data_fp_local[m][n] >> 8) * (xtest[n] >> 8);
      42:   iload_1[27]             1 I:functionalOutput_fp
      43:   aload_2[44]             1 [[I:data_fp_local
      44:   iload_3[29]             1 I:m
      45:   aaload[50]             29 Ljava/lang/Object;
      46:   iload[21]               2 I:n
      48:   iaload[46]             29 I
      49:   bipush[16]              2
      51:   ishr[122]               1
      52:   aload_0[42]             1 [I:xtest
      53:   iload[21]               2 I:n
      55:   iaload[46]             29 I
      56:   bipush[16]              2
      58:   ishr[122]               1
      59:   imul[104]              35
      60:   iadd[96]                1
      61:   istore_1[60]            1 ->I:functionalOutput_fp
```

2.3 WCET Benchmarks with Exponential Kernel

Recently, it has been established that the number of support vectors (also named *expansion vectors*) can be explicitly controlled [2]. In order to demonstrate the WCET analysis in action, we can extend the results of Wu et al.[2] to a hard real-time environment.
Up to this point, we have demonstrated that we can provide WCET analysis of a Java program that runs on the Java Optimized Processor. With the results of Wu et al., it is possible to adjust the stream classifier by reducing the number of expansion vectors in the system until some right balance of test error and decision speed is achieved. Subsequently, we can annotate (see for example the `//@WCA loop=2` in Figure 1) the Java code such that the worst-case analysis tool can provide the WCET number.
We want to calculate the WCET numbers of SVMs from the 7 data sets used in [2] and the change in classification accuracy of working with a 90% reduction in the number of support vectors, N_{SV}. The numbers are taken from Wu's paper and adorned with a WCET analysis in Table 7. We use the `dsvmfp.TestSMO.measure()` method in the following experiments. For each analysis we annotate the Java program with n and $m_{N_{SV}}$ (assuming that the arrays are compacted to avoid the check on line 7 in Figure 2) as the dimension and the number of data points respectively. This is also indicated on line 6 and 8 in Figure 2.

Table 7. WCET analysis of data sets using an exponential kernel

Dataset	m	n	$m_{N_{SV}}$	$\Delta e_{10\%SLMC}$(%)	SLMC$_{WCET}$10%
Banana	400	2	∼87	-0.8	257,833
Breast Cancer	200	9	∼113	-0.7	748,461
German	700	20	∼409	0.4	5,060,958
Image	1300	18	∼173	0.8	1,959,972
Titanic	150	3	∼71	0.3	247,623
USPS	7291	256	2683	0.4	364,354,484
Waveform	400	21	∼159	0.0	2,050,864

Table 7 shows the WCET analysis applied to the sparse kernel learning algorithm (SKLA) results for a sparse large margin classifier (SLMC). We display the number of training data m, the dimensionality of the data n, the number of support(expansion) vectors $m_{N_{SV}}$ for a traditional SVM, the difference in error from using the traditional SVM versus a SKLA SVM with 10% expansion vectors, and finally we show the cycle count for an SVM decision function: ie. a direct application of the WCET analysis. So we can *guarantee* that the cycle count of the Banana classification of a test point is 257,833 cycles, which translates to an upper bound of 2578.33 μs for a new digit classification given execution on a JOP processor configured to 100 MHz. Note that the CPU frequency can be reduced for a better energy/performance balance if the WCET does not violate a given hard deadline.

3 Conclusion

We provide two contributions in this paper. The first (to our knowledge) WCET analysis of parts of the SVM and kernel functions. Our second contribution is the identification and combination of the applicability of the SKLA method in addition to the methods described by Pedersen[5] for better design control of constrained embedded or distributed kernel-based learning algorithms. Analysis of experimental results suggests the possibility of achieving significant reductions in the upper bound (ie. WCET) of an SVM decision function by direct application of the SKLA approach by Wu et al. [2].
Future SVM directions may include worst case execution time analysis of feature reduction methods and real-time drift detection [20].

Acknowledgement: Martin Schöberl for providing feedback on the JOP/WCET sections.

References

1. Engblom, J., Ermedahl, A., Nolin, M., Gustafsson, J., Hansson, H.: Worst-case execution-time analysis for embedded real-time systems. International Journal on Software Tools for Technology Transfer **4**(4) (2003) 437–455
2. Wu, M., Schölkopf, B., Bakir, G.: A direct method for building sparse kernel learning algorithms. Journal of Machine Learning Research **7** (2006) 603–624
3. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, NY (1995)
4. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3) (1995) 273–297
5. Pedersen, R.U.: Using Support Vector Machines for Distributed Machine Learning. PhD thesis, University of Copenhagen (2005)
6. Platt, J.: Fast training of support vector machines using sequential minimal optimization in Advances in Kernel Methods — Support Vector Learning. MIT Press (1999)
7. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. In: Neural Information Processing Systems 12. (2000)
8. Schölkopf, B., Bartlett, P.L., Smola, A., Williamson, R.: Shrinking the tube: a new support vector regression algorithm. In Kearns, M.S., Solla, S.A., Cohn, D.A., eds.: Advances in Neural Information Processing Systems 11, Cambridge, MA, MIT Press (1999) 330 – 336
9. Ben-Hur, A., Horn, D., Siegelmann, H., Vapnik, V.: Support vector clustering. Journal of Machine Learning Research **2** (2001) 125–137
10. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I.H., Trigg, L.: Weka - a machine learning workbench for data mining. In Maimon, O., Rokach, L., eds.: The Data Mining and Knowledge Discovery Handbook. Springer (2005) 1305–1314

11. Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006). (2006)
12. Fischer, S., Klinkenberg, R., Mierswa, I., Ritthoff, O.: Yale: Yet another learning environment - tutorial, technical report ci-136/02 (2nd edition). Technical report, Collaborative Research Center on Computational Intelligence (SFB 531), University of Dortmund, Dortmund, Germany (August 2003)
13. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. (2001) Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
14. Schoeberl, M.: JOP: A Java Optimized Processor for Embedded Real-Time Systems. PhD thesis, Vienna University of Technology (2005)
15. Kargupta, H., Park, B., Hershberger, D., Johnson, E.: Collective Data Mining: A New Perspective Towards Distributed Data Mining. In Kargupta, H., Chan, P., eds.: Advances in Distributed and Parallel Knowledge Discovery. MIT/AAAI Press (2000) 133–184
16. Dahm, M.: Byte code engineering with the bcel api. Technical report, Freie Universität at Berlin, Institut für Informatik (2001)
17. Pedersen, R., Scöberl, M.: An embedded support vector machine. In: Proceedings of the Fourth Workshop on Intelligent Solutions in Embedded Systems (WISES 2006), Austria, Vienna (2006)
18. Lindholm, T., Yellin, F.: The Java Virtual Machine Specification. Second edn. Addison-Wesley, Reading, MA, USA (1999)
19. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: principles, techniques, and tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1986)
20. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. In Langley, P., ed.: Proceedings of the 17th International Conference on Machine Learning (ICML), San Francisco, CA, USA, Morgan Kaufmann (2000) 487–494

Dynamic Feature Space and Incremental Feature Selection for the Classification of Textual Data Streams

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas

Department of Informatics,
Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
{katak,greg,vlahavas}@csd.auth.gr

Abstract. Real world text classification applications are of special interest for the machine learning and data mining community, mainly because they introduce and combine a number of special difficulties. They deal with high dimensional, streaming, unstructured, and, in many occasions, concept drifting data. Another important peculiarity of streaming text, not adequately discussed in the relative literature, is the fact that the feature space is initially unavailable. In this paper, we discuss this aspect of textual data streams. We underline the necessity for a dynamic feature space and the utility of incremental feature selection in streaming text classification tasks. In addition, we describe a computationally undemanding incremental learning framework that could serve as a baseline in the field. Finally, we introduce a new concept drifting dataset which could assist other researchers in the evaluation of new methodologies.

1 Introduction

The world wide web is a dynamic environment that offers many sources of continuous textual data, such as web pages, news-feeds, emails, chat rooms, forums, usenet groups, instant messages and blogs. There are many interesting applications involving classification of such textual streams. The most prevalent one is spam filtering. Other applications include filtering of pornographic web pages for safer child surfing and delivering personalized news feeds.

All these applications present great challenge for the data mining community mainly because they introduce and/or combine a number of special difficulties. First of all, the data is high dimensional. We usually consider as feature space a vocabulary of hundreds of thousands of words. Secondly, data in such applications always come in a stream, meaning that we cannot store documents and we are able to process them only upon their arrival. Thirdly, the phenomenon of concept drift [8] might appear. This means that the concept or the distribution of the target-class in the classification problem may change over time.

In this paper we tackle with another issue that, to the best of our knowledge, haven't been given enough attention by the research community. This is, the

initial unavailability of the feature space. There is no prior knowledge of the words that might appear over time and the use of a global vocabulary of millions of words is simply inefficient. To deal with this problem we introduce the idea of the *feature-based* classifier. This, is a special class of classifiers that can execute in a dynamic feature space.

We furthermore investigate the utility of Incremental Feature Selection (IFS) which deals with a specific type of concept drift appearing in applications, where feature selection is of vital importance. The main notion is that as time goes by, different set of features become important for classification and some totally new features with high predictive power may appear.

We also propose a computationally undemanding framework for the classification of text streams that takes under consideration the aforementioned statements: An incremental classifier that can execute in a dynamic feature space, enhanced by an IFS procedure. Due to its simplicity and effectiveness, this approach could serve as a baseline method for other, more advanced, stream learning techniques. Finally we introduce a new concept drifting dataset which we hope it will help other researchers evaluate their work.

This paper extends previous work-in-progress report [4] by introducing: a) a new concept drifting dataset from the news classification domain, b) an investigation on the effect of IFS in three classical stream learning techniques and c) a discussion on the need for a dynamic feature space. The rest of this paper is organized as follows: Section 2, presents background knowledge on text stream mining. In Section 3, we describe the proposed approach and in Section 4 we give details about the experimental setup and discuss results. Finally in Section 5 we conclude and present our future plans.

2 Text Streams and Concept Drift

Data stream mining is a field that draws attention from data mining and database community [1, 7, 3]. The main distinctiveness of the data is that we cannot store incoming records/transactions/documents and therefore we need algorithms that process the data only once. Text streams, have additional difficulties. Some of them, like the initial unavailability of the feature space and the occurrence of concept drift are already mentioned in the introduction.

A lot of effort has been directed for the effective classification of data and text streams, especially in concept drifting environments. There are some simple methodologies dealing with concept drift. We could use a single incremental classifier updating for each document arriving. The main problem with this method is that the model is strongly built on past data and cannot quickly adapt to the drift. Another simple approach is Weighted Examples (WE) which associates recent examples with a weight in order to force the classifier focus on new data. A third well known approach is Time Window (TW). In this case, we retrain the classifier on the newest N examples in order for the classifier to model only the latest knowledge. The main disadvantages of these techniques are firstly, the assumption that older knowledge is useless for future classification (which does

not apply to all cases) and secondly the fact that the classifier is trained only from a small number of examples (equal to the size of the time window)[1]. Some more advanced methodologies involve ensembles of classifiers (a nice overview of such methods can be found in [6]). The main drawbacks of many of these approaches are the fact that they are demanding in computational sources and that many of them need a step of retraining.

3 Our Approach

3.1 Motivation

A type of concept drift that appears in textual data streams, concerns the appearance of new highly predictive features (words) that do not belong to the original feature vector. In spam filtering for example, new words must be learned by a classifier as new unsolicited commercials come into vogue [2]. In addition, spammers exercise the practice of obfuscating perpetual spam topics by replacing letters with numbers or by inserting irrelevant characters between letters (e.g. viagra becomes v1agra or v.i.a.g.r.a etc). Another application where the importance of words changes over time is personalized news filtering. In this case, the interests of the user might change over time. Therefore, new words that can better discriminate the new interests of the user must be introduced in the feature vector.

So far, the feature vector in text classification approaches has been static. The features that are selected based on an initial collection of training documents, are the ones that are subsequently considered by the classifier during the operation of the system. New words are introduced only with periodic retraining, which includes rebuilding the vocabulary and re-vectorization. Retraining demands the storage of all the documents seen so far, a requirement that might either be infeasible or too costly for textual streams. In addition, retraining has the disadvantage that it does not update the model online as new documents arrive, so it might take some time until it catches up with the drift.

Another point, not adequately discussed in the literature is the fact that in personalized applications an initial training set and consequently the feature space is unavailable. Therefore we need to use flexible algorithms and feature selection techniques that are able to execute in a dynamic feature space that would be empty in the beginning and add features when new documents arrive.

A third point is that in many cases we need classification techniques that are flexible, incremental, and, at the same time require minimum computational sources. Consider for example a web-based personalized newspaper. Each user subscribes to certain topics of interest (Sports, Arts), and a classifier is trained (by taking into consideration user feedback) in order to separate interesting messages for the user. Eventually a classifier per user and per topic is required. Therefore, a large number of computationally undemanding classifiers is required.

[1] The introduction of Adaptive Time Windows actually alleviates these problems [8]

3.2 Framework

Our approach uses two components in conjunction: a) an incremental feature ranking method, and b) an incremental learning algorithm that can consider a subset of the features during prediction. Feature selection methods that are commonly used for text classification are filters that evaluate the predictive power of each feature and select the N best. Such methods evaluate each word based on cumulative statistics concerning the number of times that it appears in each different class of documents. This renders such methods inherently incremental: When a new labeled document arrives, the statistics are updated and the evaluation can be immediately calculated without the need of re-processing past data. These methods can also handle new words by including them in the vocabulary (and fulfil the dynamic feature space requirement discussed earlier) and initializing their statistics. Therefore the first component of our approach can be instantiated using a variety of such methods, including information gain, the x^2 statistic or mutual information [10].

The incremental re-evaluation and addition of words will inevitably result into certain words being promoted to / demoted from the top N words. This raises a problem that requires the second component of the proposed approach: a learning algorithm that is able to classify a new instance taking into account different features over time. We call learning algorithms that can deal with it *feature-based*, because learning is based on the new subset of features, in the same way that in instance based algorithms, learning is based on the new instance.

Two inherently feature based algorithms are Naive Bayes (NB) and k Nearest Neighbors (kNN). In both of these algorithms each feature makes an independent contribution towards the prediction of a class. Therefore, these algorithms can be easily expanded in order to instantiate the second component of our approach. Specifically, when these algorithms are used for the classification of a new instance, they should also be provided with an additional parameter denoting the subset of the selected features. NB for example will only consider the calculated probabilities of this subset, while kNN will measure the distance of the new instance with the stored examples based only on this feature subset. Note, that the framework could apply to any classifier that could be converted into a feature-based classifier.

It is also worth noticing that the proposed approach could work without an initial training set and fulfil the dynamic feature space requirement discussed earlier. This is useful in personalized web-content (e-mail, news, etc.) filtering applications that work based largely on the user's perception of the target class.

Figure 1 presents algorithm UPDATE for the incremental update of our approach. When a new Document arrives as an example of class DocClass, the first thing that happens is to check if it contains any new words. If a new Word is present then it is added to the vocabulary (ADDWORD) and the WordStats of this Word are initialized to zero. Then, for each Word in the Vocabulary we update the counts based on the new document and re-calculate the feature evaluation metric. Finally, the Classifier must also be vertically updated based on the new example and also take into account any new words. For the classification

of a new unlabeled Document, the algorithm selects the top-N words based on their evaluation and then predicts the class of the document by taking under consideration only the selected feature subset.

```
input  : Document, DocClass, Classes, Vocabulary
output: Classifier, Vocabulary, WordStats, Evaluation
begin
    foreach Word ∈ Document do
        if Word ∉ Vocabulary then
            AddWord(Word, Vocabulary)
            foreach Class ∈ Classes do
                WordStats [Word][Class][1] ← 0
                WordStats [Word][Class][0] ← 0
    foreach Word ∈ Vocabulary do
        if Word ∈ Document then
            WordStats [Word][DocClass][1] ← WordStats [Word][DocClass][1] + 1
        else
            WordStats [Word][DocClass][0] ← WordStats [Word][DocClass][0] + 1
    foreach Word ∈ Vocabulary do
        Evaluation ← EvaluateFeature(Word, WordStats)
    Classifier ← UpdateClassifier(Document, DocClass)
end
```

Fig. 1. Algorithm Update

4 Experimental Results

4.1 Feature Selection and Learning Algorithm

The x^2 metric was selected for instantiating the feature evaluation method of the proposed approach, due to its simplicity and effectiveness [10]. We extended the implementation of the x^2 feature evaluation method of Weka [9] with a function that allows incremental updates. As we mentioned in the previous section, there are many other similarly simple metrics that could be used for instantiating our framework. Here, we are not focusing on the effectiveness of different feature evaluation methods, but rather on whether a feature based classifier coupled with an incremental version of such a method is useful in textual data stream classification.

The learning algorithm that was selected for instantiating the learning module of the proposed approach was Naive Bayes. kNN is inefficient for data-streams, as it requires the storage of training examples. NB on the other hand stores only the necessary statistics and is also widely used in text classification

applications. In addition, NB can take advantage of the already stored feature statistics for the purpose of feature ranking and thus integrates easier in the proposed approach. We extended Weka's implementation of NB with a function that accepts a feature subset along with a new instance and uses only the features of this subset for the classification of the instance. Note that we are not focusing on the effectiveness of the specific algorithm. Any incremental machine learning algorithm could be used as long as it is, or, could be transformed to, a feature-based classifier.

4.2 Data Sets

The first requirement of an empirical study of the proposed approach is a data set with documents obtained from a real word textual data stream. We actually experimented with two content filtering domains, spam and news.

For the domain of spam filtering we ideally need real-world spam and legitimate emails chronologically ordered according to their date and time of arrival. In this way we can approximate the time-evolving nature of the problem and consequently evaluate more properly the proposed approach. For that reasons, we used the SpamAssasin (http://spamassasin.apache.org/) data collection because a) Every mail of the collection is available with the headers, thus we were are able to extract the exact date and time that the mail was sent or received, and b) It contains both spam and legitimate (ham) messages with a decent spam ratio (about 20 %). This dataset consists of 9324 instances and initially 40000 features. This datasets represents the so-called gradual concept drift.

For the domain of news filtering we needed a collection of news documents corresponding to the interests of a user over time. As such a collection was not available, we tried to simulate it using usenet posts from the 20 Newsgroups collection[2]. The data set was created to simulate concept drift. The scenario involves a user that over time subscribes to and removes from different general mailing lists (or news feeds) (e.g. sports, science etc) but is interested only on certain subcategories of these mailing lists. Table 1 shows, the particular interests of the user and how her general interests change over time. For example the user is initially interested in sports, but then loses this interest and subscribes in a science mailing list. The user is perpetually interested in driving, while in the last part she also gets into religion issues and at the same time unsubscribes from the hardware list. This dataset consists of 6000 instances and initially 28000 features. In both datasets, we have removed headers and used a boolean bag-of-words approach for the representation of documents. Other methods for IFS presented in the literature like [5] haven't been tested on such high-dimensional datasets. This dataset represents the sudden concept drift[3].

[2] http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html
[3] both datasets are available at http://mlkd.csd.auth.gr/datasets.html

Table 1. Interest of user in the various newsgroups over time

	instances		
newsgroup	**1-3000**	**3001-6000**	**4501-6000**
comp.pc.hardware	Yes	Yes	-
comp.mac.hardware	No	No	-
rec.autos	Yes	Yes	Yes
rec.motorcycles	No	No	No
rec.sport.baseball	Yes	-	-
rec.sport.hockey	No	-	-
sci.med	-	No	No
sci.space	-	Yes	Yes
soc.religion.christian	-	-	Yes
alt.atheism	-	-	No

4.3 Methods

To evaluate the effectiveness of our methodology, we applied the basic framework proposed before (IFS) to the three simple learning techniques discussed in the introduction (Simple Incremental Classifier, Weighted Examples, Time Windows).

4.4 Results

We compare the predictive performance of the above methodologies (using x^2 for feature ranking with a classical Incremental Naive Bayes (INB) classifier in the non-IFS approaches and a Feature Based NB classifier for the IFS-enhanced approaches. All methods are executed on the two document collections (spam, news), using as initial training set the first 10, 20 and 30% of the documents. The rest of the documents were used for testing: all methods first predict the class for each new document and then update their models based on the actual class of it. We fixed the number of features to select to 500, as past results have shown that a few hundreds of words are an appropriate size of features. Table 2 shows the results of the experiments. After preliminary experimentation, we concluded that 300 instances was a good window size for the TW method and that a decent way to update the weights in the WE method was $w(n) = w(n-1) + n^2$, where $w(n)$ is the weight of the n-th instance. Note, that we are not focusing on the accuracy of the aforementioned methodologies, but rather on the effect of IFS on them. We first notice that all methods when enhanced with IFS have better predictive performance than the classical approaches in both data sets, all percentages of initial training documents and both metrics. This shows that incremental feature selection manages to catch up with the new predictive words that are introduced over time. In the spam domain, the inclusion of more training data increases the predictive performance of all methodologies due to the inclusion from the beginning of the important features that appear early in the data set. In the news domain on the other hand the inclusion of more training data does

Table 2. Accuracy (acc) and area under the ROC curve (au) for the two data sets and the three different percentages of training documents for three learning methodologies (with and without IFS)

		10%		20%		30%	
Dataset	**Method**	**acc**	**auc**	**acc**	**auc**	**acc**	**auc**
spam	INB	66.06	81.64	51.44	81.53	88.55	93.23
	INB+IFS	86.28	92.48	90.27	95.42	94.02	97.11
	TW	89.71	93.08	90.62	93.03	91.86	92.44
	TW+IFS	90.99	94.42	91.80	94.67	93.56	94.68
	WE	89.76	93.67	92.35	94.60	96.00	97.08
	WE+IFS	93.61	96.75	95.56	98.01	95.81	97.56
news	INB	76.04	87.74	76.06	87.57	74.11	85.28
	INB+IFS	84.07	93.57	84.11	93.53	83.77	93.19
	TW	78.38	86.38	78.41	86.10	78.12	85.80
	TW+IFS	79.27	87.50	79.54	87.55	79.59	87.42
	WE	80.38	88.77	80.00	89.06	78.38	87.63
	WE+IFS	84.98	93.33	85.03	93.15	85.08	93.07

not increase performance significantly as it becomes harder for the classifiers to forget the initial knowledge and adapt to the new predictive features that appear later on. Figures 4a to 4c shows the moving average (over 200 instances) of the prediction accuracy of all methods (with and without IFS) using the first 20% of all messages of the news collection for training[4]. We notice that for the first instances the performance is comparable, but from then on the performance of IFS-enhanced methods becomes and remains much better than simple methods. This happened because at that time-point the user subscribed to new lists and new predictive words appeared. The same thing occurred after the first 3200 examples when the user changed interests for the second time. Non-IFS methods failed to keep up with the new user interests, while IFS-enhanced methods managed to maintain their initial predictive performance. The TW method is the only one that is not significantly affected from IFS, and that can be seen in table 2 (see accuracy TW versus TW+IFS in both datasets) and also from figure 4c. This is mainly because of the small window size that the IFS is applied to. Figures 4e and 4d show the moving average (over 200 instances) of the number of words promoted to/demoted from the top 500 words in both datasets for the INB+FS method. Note that in the news domain, in the beginning more words are promoted to/demoted from the top 500 words as the evaluation scores of already included words continue to change with more training examples, while towards the end they stabilize. The peak in the spam domain is due to the skewness of the collection (a large number of new spam messages arrived at that time point).

SVMs are well known accurate text classifiers and independent of feature selection. Indicatively, we applied an SVM (Weka implementation (SMO), default

[4] The respective figures for the spam corpus are similar

parameter setting (Polynomial Kernel of degree 1, C=1, L=0,001), no initial training) in the news dataset with retraining for every 300 instances and obtained an average accuracy of 70.02%. With TW+IFS method (no retraining) we obtained 77.95% accuracy. Naturally the time needed for the execution of the SVM was much larger (approx. 4 times).

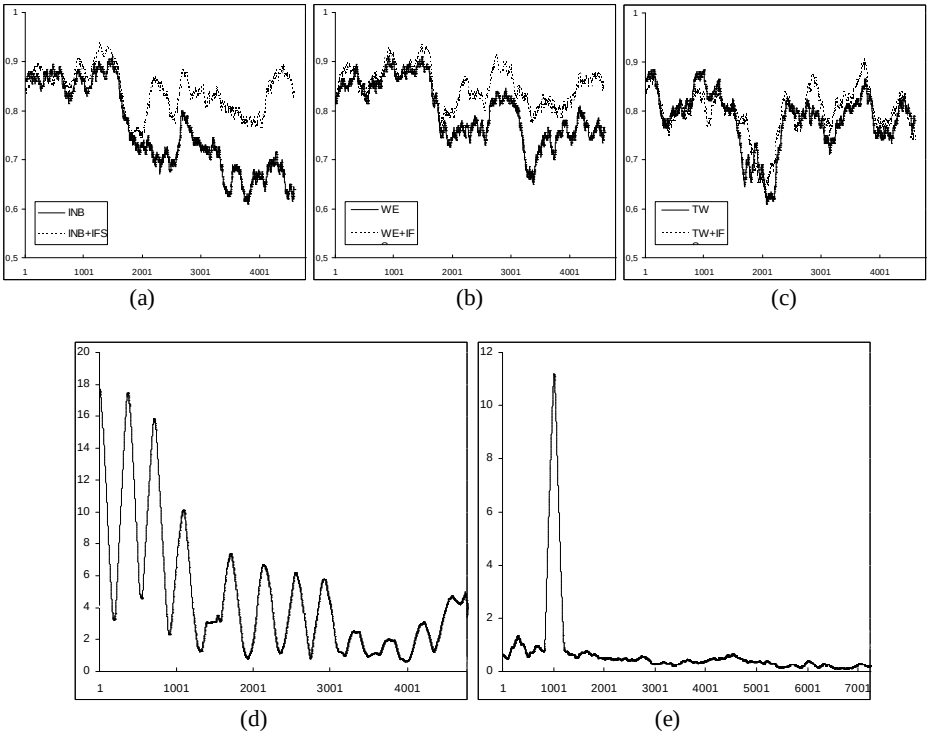


Fig. 2. (a),(b),(c) Moving average of the accuracy for the news domain, and (d),(e) Moving average of the number of words promoted to/domoted from the top 500 words, using the first 20% of all messages of the news collection (d) and spam corpus (e) for training (INB+FS method).

5 Conclusions and Future Work

This paper focused on an interesting special type of concept drift that is inherent to textual data streams: The appearance of new predictive features (words) over time. In the past, this type of concept drift has not been considered by online learning approaches to the best of our knowledge, rather it was confronted with the cumbersome approach of retraining. We presented a computational undemanding approach that combines an incremental feature selection method with

what we called a feature based learning algorithm in order to deal with this problem and we underlined the importance of a dynamic feature space. The experimental results showed that the proposed approach offers better predictive accuracy compared to classical incremental learning and are encouraging for further developments. We also hope that the use of the 20 newsgroups for simulating drifting interests will inspire other researchers for similar experiments. We believe that the proposed approach is a straightforward method for dealing with online learning in high-dimensional data streams, and could be considered as a baseline for comparison with other more complex methods, such as approaches based on ensembles of classifiers, due to its simplicity and effectiveness.

6 Acknowledgements

This work was partially supported by the Greek R&D General Secretariat through a PENED program (EPAN M.8.3.1, No. 03E Δ 73).

References

1. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
2. T. Fawcett. "in vivo" spam filtering: A challenge problem for data mining. *KDD Explorations*, 5(2), December 2003.
3. F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 568–572, New York, NY, USA, 2005. ACM Press.
4. I. Katakis, G. Tsoumakas, and I. Vlahavas. On the utility of incremental feature selection for the classification of textual data streams. In *10th Panhellenic Conference on Informatics (PCI 2005)*, pages 338–348. Springer-Verlag, 2005.
5. S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333–1356, 2003.
6. M. Scholz and R. Klinkenberg. Boosting classifiers for drifting concepts. *Intelligent Data Analysis (IDA), Special Issue on Knowledge Discovery from Data Streams (accepted for publication)*, 2006.
7. P. Wang, H. Wang, X. Wu, W. Wang, and B. Shi. On reducing classifier granularity in mining concept-drifting data streams. In *ICDM*, pages 474–481. IEEE Computer Society, 2005.
8. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.
9. I. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations.* Morgan Kaufmann, 1999.
10. Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In D. H. Fisher, editor, *Proceedings of ICML-97*, pages 412–420. Morgan Kaufmann Publishers, San Francisco, US, 1997.

User constraints over data streams

Carlos Ruiz Moreno[1], Myra Spiliopoulou[2], and Ernestina Menasalvas[1]

[1] Facultad de Informatica, Universidad Politecnica, Madrid, Spain
cruiz@cettico.fi.upm.es,emenasalvas@fi.upm.es
[2] Faculty of Computer Science, Magdeburg University, Germany
myra@iti.cs.uni-magdeburg.de

Abstract. Models and patterns discovered upon accumulating or streamed data must be updated whenever the data change. However, whether a pattern is still valid or useful is not determined solely by its statistical properties; it also depends on the expert's perception and the demands of the application. These can be best expressed as "user constraints". Constraint clustering methods enjoy increasing attention, because constraints can capture the expert's perspective of the domain, so that solutions with given properties are derived, thus improving quality and, sometimes, convergence time of the clustering algorithm.
We propose a model for user constraints upon clusters over data streams. We introduce different types of constraints, needed to express phenomena associated with cluster semantics and with the role of individual objects in the clusters, while taking account of data decay, as is typical in data streams. We further present a mechanism that updates the set of constraints as data decay. To support the enforcement of constraints during re-clustering, we propose a method that ranks the constraints to be enforced and we extend the constraint-based K-means of [16] into a clustering algorithm that enforces a prioritized set of constraints.

1 Introduction

Knowledge Discovery from Streams studies the extraction of useful knowledge from non-stopping data streams [8]. Obviously, a stream is subject to endogenous or exogenous changes that affect the validity of the extracted models and patterns: The customer base of a retailer may change in response to a marketing campaign or due to demographic evolution; the usage patterns of a web site change as its contents change or as users become more experienced; spam distributors change their strategy in response to more elaborate anti-spam filters. Although such changes are of potential interest, the "interestingness" of patterns and of their changes also depend on the perception and expectations of the human expert.

The importance of linking derived patterns with background knowledge and expectations of the expert is stressed among else in [1, 13]. "User constraints" are a powerful concept for the establishment of this link. Recently, user constraints are used in clustering to shift from the generation of unsupervised solutions to "semi-supervised" clusterings [9] that exploit prior knowledge on the data.

In this study, we extend the notion of user constraints from static data to data streams, focussing on clustering and cluster evolution in streams. Constraints are a very appropriate instrument to guide the cluster adaptation and reconstruction process across

a stream: A clustering model is derived upon the first instance of the population; constraints are derived from this model, reflecting the properties of the population; these constraints should be respected by all further instances of the population, otherwise the expert should be alerted. However, current models of constraints cannot support this process, because they mostly refer to concrete data records. Such constraints cannot be transferred to stream mining, because the records of a stream are gradually forgotten.

After discussing the literature on constraints for the clustering of static data (Section 2), we propose a model for constraints over streams (Section 3). Our model covers different types of constraints and deals with the aspect of data decay. It is accompanied by an algorithm that updates the set of constraints at each timepoint of observation, identifying obsolete constraints, updating and re-weighting active constraints on the basis of the age of the associated data records. Constraint weights are used to design a first variation of a "prioritized" constraint-based clustering algorithm that re-clusters data after a constraint violation and attempts to enforce as many high priority constraints as possible. The last section concludes the study with future work on dedicated cluster violation predictors and on constraint enforcement for streamed data.

2 Clustering with user constraints

Recently, clustering with constraints has received particular attention as a means to alleviate some endogen problems of conventional clustering. Halkidi et al point out that the notion of "right clustering" depends on the user's perspective [10]. Bennet et al stress that clustering algorithms cannot always avoid solutions of poor quality, including clusterings with some empty clusters [3]. User constraints can improve the quality of the output clustering and sometimes even improve computational performance [7], [6].

Moreover, user constraints are appropriate for expressing important aspects of domain knowledge, thus leading to solutions with specific properties. In constraint-based clustering, background knowledge often takes the form of *instance-level constraints*, i.e. constraints on whether two records/instances may or may not belong to the same cluster (e.g. [16]). Such constraints have been applied to real-world problems such as detecting road lanes from GPS data [16] or helping the navigation of a Sony Aibo Robot [7], yielding promising results.

2.1 Types of Conventional User Constraints

Han et al [12] propose five general categories of contraints: (i) *knowledge type constraints* that reflect strategic objectives of the analysis, expressed by the type of knowledge to be mined (e.g. association, classification); (ii) *data constraints* specified by the user upon the data selected for data mining; (iii) *dimension/level constraints* refering to the level of abstraction and the dimensions or properties of the data that should be studied; (iv) *rule constraints*, i.e. constraints upon the rules to be discovered by data mining; (v) *interestingness constraints* upon the discovered patterns, meant to express the usefulness or interestingness of the patterns. Such constraints can be based upon quality measures for clustering, association and classification; an extensive overview of such

measures can be found in [14]. With respect to this categorization, our approach concentrates on the last three types of constraints: Our constraint model capture restrictions on properties of the data within a cluster and properties of the clusters themselves.

When studying conventional user constraints for clustering, we can further distinguish between instance-level and cluster-level constraints. The former refer to concrete data records. They are motivated by the fact that many applications deliver some labeled data that can be exploited for the clustering of non-labeled data (cf. "semi-supervised clustering"). Some specific types of constraints proposed in the literature under this categorization are:

- *Must-Link constraints* [15] specify that a pair of points x and y must be assigned to the same cluster.
- *Cannot-Link constraints* [15] specify that a pair of points x and y cannot be assigned to the same cluster.
- δ*-constraints* (also called minimum separation) [7] specify that for any two clusters C_i and C_j and for any two points $x \in C_i$ and $y \in C_j$, $Distance(x, y) \geq \delta$.
- ϵ*-constraints* [7] specify that for any cluster C_i it holds that $\forall x \in C_i, \exists y \in C_i$, and $Distance(x, y) \leq \epsilon$.
- τ_n*-constraint* [3]: This constraint refers to a cluster as a whole and specifies that this cluster must have at least n members/records.

The τ_n constraint is a cluster-level constraint, intended to prevent locally optimal solutions that involve empty or very small clusters. *Must-Link* and *Cannot-Link* are instance-level constraints. The types δ-constraint and ϵ-constraint are so-called *conditional constraints*. They can be used to transform cluster-level constraints to instance-level constraints. This means that multiple constraints are generated using a δ-constraint as the minimum separation between clusters and an ϵ-constraint as the maximum dispersion in a cluster.

A crucial issue on (instance-level) constraint-based clustering is the feasibility of a solution that satisfies all constraints. Davidson et al. [7] prove that satisfying all types of constraints is an NP-Complete problem for K-Means. However, it is a P-Complete problem for hierarchical clustering [6].

2.2 Algorithms and Frameworks Using Constraints

The COP-KMeans proposed in [16] is a K-Means variant that incorporates background knowledge in the form of instance-level constraints, concretely Must-Link and Cannot-Link. The algorithm takes as input a dataset and a set of constraints and returns a data partitioning that satisfies all specified constraints. The algorithm progresses iteratively by checking whether constraints are violated and stopping when a partitioning that satisfies all constraints has been built.

As pointed out in [7], the convergence of such an algorithm is not guaranteed. They propose a K-means variant that minimizes the vector quantization error (the distortion), while attempting to satisfy as many constraints as possible. The idea behind this proposal is that satisfying a constraint may increase the error, so they define a new constrained vector quantization error based on penalties over constraints that cannot be satisfied [7].

In [4, 2], constraints are used to learn the distance function that determines the partitioning. In "semi-supervised learning", constraints are derived from labeled data and are used to cluster unlabeled data. In this context, Halkidi et al [10] propose a clustering framework, where quality measures for clustering (as in [11]) serve as so-called "objective criteria" and are combined with Must-Link and Cannot-Link constraints as so-called "subjective criteria" derived from the labeled data.

Those approaches consider constraints over static data. In the following, we present our approach on constraint modeling and enforcement over stream data.

3 Modeling and Checking Constraints across a Data Stream

When dealing with an accummulating dataset or a data stream, the notion of "constraint" must be adjusted: Constraints on individual data records cannot be sustained in a stream, because old data are gradually forgotten. In a stream, we are rather interested in constraints on a cluster's cardinality (cf. constraint τ_n [3]) or support within a clustering, on the dominance of some attribute values inside a cluster and, as a special case, on the cluster's derived "label", i.e. the ad hoc class of the data records it contains.

We enhance the notion of "user constraint" in two ways. First, instead of assuming that a constraint is either satisfied or violated, we allow that a constraint may be active or obsolete; active constraints may be satisfied or violated, obsolete ones are not checked at all. Second, the notion of crisp constraint is replaced by a "weighted constraint" whose enforcement depends on its weight.

3.1 Modeling User Constraints over Accummulating Data

Similarly to constraint-based clustering, constraints are derived from a set of labeled examples and then exploited to cluster the unlabeled data [5]. However, instead of assuming the existence of labeled and unlabeled data at the same timepoint, we assume quite naturally that all data upon which the first clustering ζ_1 at timepoint t_1 is created are now labeled and can be used to derive constraints. In subsequent timepoints $t_2, \ldots, t_n$, we need to check whether the clusterings $\zeta_2, \ldots, \zeta_n$ over the new data have experienced constraint violations. We model a user constraint as a predicate $p \in \mathcal{P}$ and consider the following types of predicates:

1. *Constraint on pattern:* Let ξ be a pattern over the data and a be a property of this pattern. Further, let $V(a)$ be the valuerange of a and $X \subset V(a)$ a subset of it. A "constraint on pattern" has the form $pattern(\xi, a, X)$, stating that the value of a for ξ must belong to subset X. We denote the set of those constraints as $\mathcal{P}_{pattern}$.
   For example, assume that we want to partition the IRIS dataset (cf. UCI repository) with a clustering algorithm and let ζ be a derived clustering. A constraint $pattern(\zeta, cardinality, \{3\})$ means that ζ must consist of exactly 3 clusters. A constraint $pattern(C, support, (0, 0.33])$ states that the cluster $C \in \zeta$ (which is also a pattern) should not accommodate more than one third of the dataset.
2. *Constraint on attribute:* Let $C \in \zeta$ be a cluster. Let A be a data attribute, $V(A)$ be its valuerange and let $X \subset V(A)$. A "constraint on attribute" has the form $attr(C, A, X)$,

stating that all values of attribute A for the records in C should belong to X. We denote the set of those constraints as $\mathcal{P}_{attr}$.
Let $\zeta_{good} = \{C_x, C_y, C_z\}$ be a clustering that satisfies the example constraints on pattern above. The constraint $attr(C_x, petalWidth, X)$ states that the petal width for all flowers in C_x should be within the valuerange X. Further, for $X' := (-\infty, +\infty) \setminus X$, the additional constraints $attr(C_y, petalWidth, X')$ and $attr(C_z, petalWidth, X')$ would ensure that flowers with petal width in X appear only in the cluster C_x.

3. *Constraint on co-appearance:* Let $C \in \zeta$ be a cluster and let $d \subseteq D$ be a subset of the dataset. A "constraint on co-appearance" $inside(C, d)$ states that all members of d should belong to C. We denote the set of those constraints as $\mathcal{P}_{in}$.
4. *Constraint on separation:* Let $C \in \zeta$ be a cluster and let $d \subseteq D$ be a subset of the dataset. A "constraint on separation" $outside(C, d)$ states that none of the members of d may belong to C. We denote the set of those constraints as $\mathcal{P}_{out}$.

The type "constraint on co-appearance" subsumes the constraint type *must_link*, while the type "constraint on separation" subsumes *cannot_link* between pairs of records, except that our new types refer to a predefined cluster C. This is intuitive, since constraints in our context refer to already identified clusters.

3.2 States of a Constraint

User constraints are defined upon the original data encountered at the first timepoint t_1 and grouped into the initial clustering ζ_1. Intuitively, constraints that explicitly refer to specific data records become irrelevant when the records are forgotten. Hence, we specify that a constraint can be either *active* or *obsolete*. If a constraint is active, then it is checked for violation and is found to be either *satisfied* or *violated*.

A constraint p refering to a pattern ξ is obsolete if one of the following conditions is satisfied: (i) ξ is a cluster, $p()$ is a constraint on co-appearance or separation (i.e. $p()$ has the form $inside(\xi, d)$ or $outside(\xi, d)$) and the dataset d is empty because the records in it are forgotten. (ii) ξ is a cluster that does not exist any more because the old records in it are forgotten and no new ones have been inserted. This condition affects constraints on attribute, co-appearance and separation. (iii) ξ is a cluster or clustering that has been replaced because of reclustering. This condition concerns constraints on pattern.

Obsolete constraints are not checked against violation but are not removed either, because new records may lead to their re-activation. Constraint violation checking and constraint enforcement (through re-clustering) are performed for active constraints only.

3.3 Updating Constraints

Let $\mathcal{P}$ be a set of constraints. For conventional constraint violation checking, it is sufficient to juxtapose each constraint to the dataset. In our expanded model, constraint checking must be preceded by an "updating step" that identifies obsolete constraints. In Fig. 1 we show a simple greedy algorithm `ConstraintUPD` to this purpose. [1]

[1] In the verbatim listings of the algorithms in Fig. 1 and Fig. 2, subscripts of the form $Y_{something}$ could not be reproduced. We have used the notation $Y\_something$ instead.

```
1  O ← ∅                                                    // Set of obsolete constraints - Initialization
2  Z_attr ← ∪{C ∩ D_i | p ≡ attr(C, A, X) ∈ P_attr}         // Records in constraints on attribute
3  Z_in ← ∪{d ∩ D_i | p ≡ inside(C, d) ∈ P_in}              // Records in constraints on co-appearance
4  Z_out ← ∪{d ∩ D_i | p ≡ outside(C, d) ∈ P_out}           // Records in constraints on separation
5  if Z_attr = ∅ then O ← P_attr
6  if Z_in = ∅ then O ← ∪ P_in
7  if Z_out = ∅ then O ← ∪ P_out
8  if Z_attr ≠ ∅ then
9     for each p ≡ attr(C, A, X) ∈ P_attr
10       if C ∩ D_i = ∅ then O ← O ∪ {p}
11    endfor
12 endif
13 if Z_in ≠ ∅ then
14    for each p ≡ inside(C, d) ∈ P_in
15       if C ∩ D_i = ∅ or d ∩ D_i = ∅ then O ← O ∪ {p} else d ← d ∩ D_i
16    endfor
17 endif
18 if Z_out ≠ ∅ then
19    for each p ≡ outside(C, d) ∈ P_out
20       if C ∩ D_i = ∅ or d ∩ D_i = ∅ then O ← O ∪ {p} else d ← d ∩ D_i
21    endfor
22 endif
23 for each p ≡ pattern(C, a, X) ∈ P_pattern
24    if C ∩ D_i = ∅ then O ← O ∪ {p}          //Constraint on pattern upon an empty cluster
25 endfor
26 if O ≠ ∅ then raise an alert
27 P_pattern ← P_pattern \ O; P_attr ← P_attr \ O
28 P_in ← P_in \ O; P_out ← P_out \ O
29 return P_pattern, P_attr, P_in, P_out
```

Fig. 1. Algorithm `ConstraintUPD` for the updating of constraints and associated datasets

At each timepoint t_i, `ConstraintUPD` operates upon the dataset D_i and the current state of the clustering ζ_i. D_i consists of the records inserted between t_{i-1} and t_i and those old records whose decay value is still larger than zero. `ConstraintUPD` takes as input the four sets of constraints (one per constraint type), identifies obsolete ones and raises an alert, if some constraints are no more active (cf. line 26 of Fig. 1).

In lines 2–4, `ConstraintUPD` initializes the sets of records involved in constraints on attribute, co-appearance and separation, denoted as Z-sets hereafter. For the constraints on co-appearance, Z_{in} contains the records involved in constraints of $\mathcal{P}_{in}$ *and* appearing in D_i (line 3). Z_{out} for constraints on separation is computed similarly (line 4). Z_{attr} is the intersection of D_i with all clusters involved in constraints of $\mathcal{P}_{attr}$.

If a Z-set is empty, then all constraints of the corresponding type are marked as obsolete (lines 5–7). If Z_{attr} is not empty, then each constraint on attribute p is checked (lines 8–12): If it refers to a cluster whose records are no more in D_i, then ps is obsolete.

If Z_{in} is not empty, then the corresponding test is applied upon the constraints on co-appearance (line 13–17): For such a constraint $p \equiv inside(C, d)$, the associated dataset is actualized by removing records that are no more in D_i (line 15). The same procedure is performed for constraints on separation (lines 17–22). Finally, constraints on pattern are checked (lines 23–25), detecting those applied on clusters of decayed records.

3.4 Detecting Constraint Violations

Once obsolete constraints are identified and filtered out, active constraints can be checked and re-enforced, if necessary. Constraint checking is not necessary for all constraints though: Constraints on co-appearnace and separation, once satisfied at timepoint t_0, cannot be violated unless a re-clustering takes place. Hence, constraint violation checking needs to be invoked for constraints on attribute and constraint on pattern only.

An active constraint on attribute $attr(C, A, X)$ may be violated by the newly inserted data, i.e. whenever a record x is inserted such that $x.A \notin X$. The detection of violations for these constraints can be incorporated in `ConstraintUPD` of Fig. 1, line 10.

Active constraints on pattern may also be violated, since the decay of records affects the cardinality, mean, standard deviation and further properties of clusters. A constraint $pattern(C, a, X)$ refering to a cluster C can be checked when testing whether C still has members (lines 23–25 of Fig. 1). Constraints that refer to clustering ζ as a whole are always active and can be checked after the constraint updating process. It is noted that constraints on the cardinality of ζ may be violated if some clusters are empty.

3.5 Priorization of Constraints for Re-Clustering

Constraint violation triggers alerts and data re-clustering. As pointed out in [5], the enforcement of some constraints may be intractable. Hence, constraint-based algorithms do not guarantee that all constraints are satisfied. We partially alleviate this problem as follows: First, we introduce a *weighting* of constraints based on the age of the records they refer to. Next, at each timepoint t_i we *instantiate* constraints on attribute into constraints on co-appearance upon the dataset D_i. Third, we use the weights of the constraints to compute ranks for constraint groups. Finally, we extend the constraint-based K-means of [16] to deal with *prioritized* lists of constraints on co-appearance and separation. We discuss those steps in the following.

Weights of Constraints. The weight of a constraint on co-appearance $p = inside(C, d)$ or separation $p = outside(C, d)$ is the maximum weight among the data in d:

$$weight(p) := \max_{x \in d}\{weight(x)\}$$

where a record's weight depends on its decay value. If data decay is implemented with a sliding window, then all records inside the window have the weight 1, while all records outside it have the weight 0. If decay is implemented as a continuous function $age() \in [0, 1]$ with value one for the newest records, then for each record x, $weight(x) := age(x)$.

The weight of a constraint on attribute $p = attr(C, A, X)$ is the maximum weight among the relevant data in C: $weight(p) := \max_{x \in C}\{weight(x) | x.A \in X\}$. Obviously, decayed records do not contribute to the weight of p, since their weight is zero. Finally, constraints on patterns have always the weight 1.

Instantiating Constraints on Attribute. For the dataset D_i at t_i, we instantiate each constraint on attribute $attr(C, A, X) \in \mathcal{P}_{attr}$ into a co-appearance constraint by stating that all records $x \in C$ such that $x.A \in X$ must be assigned to the same cluster. Hence, we derive from $attr(C, A, X)$ the constraint $inside(C, d_{A,X})$ with $d_{A,X} = \{x \in D_i | x.A \in X\}$. We denote this set of derived constraints on co-appearance as $Derived_{in}$.

It is noted that a derived constraint is extensionally but not intentionally equivalent to the original one: The derived constraint is not violated if a new record y with $y.A \in X$ is not in C. Hence, we use derived constraints only during re-clustering. It is also noted that the weight of the derived constraint is the same as the weight of the original one, since they both consider the same non-decayed data records.

Ranking Constraints. For constraint ranking, we derive n adjacent intervals of weight values $rank_1, \ldots, rank_n$, such that $rank_i > rank_{i+1}$. We associate with each interval $rank_i$ two sets of constraints: The set c_i consists of the original and derived active constraints on co-appearance that have weights in $rank_i$, while s_i accommodates the active constraints on separation with weights in $rank_i$.

Enforcing a Prioritized List of Constraints. In Fig. 2 we present a prioritized variation of the constraint-based K-means proposed in [16]. Our algorithm takes as input a lower boundary τ over the ranks of constraints and builds clusterings that satisfy most constraints downto τ by performing up to m attempts/iterations. It deals with constraints on co-appearance (original and derived ones) and on separation. When specifying K, constraints on this number (i.e. constraints on pattern) are also respected. Further constraints on pattern are only checked after the re-clustering.

Our prioritized constraint-based K-means consists of two phases: In Phase I (lines 1–15), it considers the constraints on co-apperance, starting with the highest rank and building one clustering per rank if possible. In Phase II (lines 16–22), it considers the constraints on separation, again starting with the highest rank. It returns the clustering ξ_0 that satisfies most co-apperance constraints (line 23) and the clustering ξ_1 that satisfies all constraints on separation downto a minimum rank $rank_x$ (line 24). Clusterings that violate individual constraints on separation are also retained for inspection (line 19).

Our algorithm starts by building a set of proto-clusters, similar to the connected components of the original algorithm in [16]: The proto-clusters satisfy constraints on co-apperance of $rank_i$ (lines 2,3). For each proto-cluster, the average of the data records is computed and used as centroid thereafter (line 4). We assume that the number of clusters K does not change, so that the number of clusters involved in constraints $\widehat{k}$ cannot exceed K. The remaining centroids $K - \widehat{k}$ are generated randomly. Clusterings are then built progressively, ensuring that more and more constraints of lower weights are satisfied (line 12), until some constraint violation occurs (line 13). Thereafter, separation constraints are considered against the retained clusterings (lines 16-22).

4 Conclusions and Outlook

We have presented a model for user constraints, which encompasses constraint checking, updating and enforcement over a data stream. Our model is motivated by the observation that user constraints over a data stream are of different nature than constraints

```
1  for each i = 1, ..., n and while rank_i > τ do
2     for each cluster C appearing in constraints of c_i do
3        protoC ← ∪{d|p ≡ inside(C, d) ∈ c_i}
4        compute the centroid of protoC as the average of the records in it
5     endfor                                   // k̂ centroids generated from proto-clusters
6     generate K − k̂ cluster centroids for clustering ζ_i
7     iterate at most m times
8        assign each record x ∈ D_i to the closest feasible cluster
9        recalculate the centroids of ζ_i
10    end-iterate
11    if ζ_i has converged
12       then retain ζ_i; continue for i ← i + 1
13       else break
14    endif
15 endfor
16 for each j = i, ..., 1 such that ζ_j exists do
17    for each i = n, ..., 1 and while rank_i > τ do
18       if there is a p ∈ s_i that is violated in ζ_j
19          then retain (ζ_j, rank_i, p)
20          else retain (ζ_j, rank_i, ∅) // ζ_j satisfies all separation constraints of rank_i
21       endif
22    endfor
23 ξ_0 ← ζ_i where i is minimum // clustering satisfying most co-appearance constraints
24 ξ_1 ← ζ_u such that (ζ_u, rank_x, ∅) is retained and rank_x is minimum
25 return ξ_0, ξ_1
```

Fig. 2. Prioritized variation of constraint-based K-means (cf. [16]) for constraints over a stream

over static data. Hence, next to constraints over individual records in the clusters, we consider constraints on the statistics of the clusters and on the dominant attribute values of the clusters. Our constraint updating algorithm eliminates constraints refering to decayed data and assigns weights to retained constraints - computed upon the age of the associated data records.

The violation of active constraints calls for constraint-sensitive re-clustering. To this purpose, we extend the constraint-based K-means algorithm of [16]. Our algorithm exploits the weights of the constraints by attempting to enforce as many high-weight constraints as possible.

Our approach is only an initial conceptual effort in modeling and enforcing constraints upon streams. We intend to study extensions of constraint-enforcing algorithms with convergence guarantees, as those proposed in [7], and design them for different types of constraints on streams. Further, we are interested in *early indicators* of a *forthcoming* constraint violation and intend to study how functions that depict the statistics of clusters and clusterings can serve to this purpose.

References

1. S. S. Anand, D. A. Bell, and J. G. Hughes. The Role of Domain Knowledge in Data Mining. In *CIKM '95: Proceedings of the Fourth International Conference on Information and Knowledge Management*, pages 37–43, New York, NY, USA, 1995. ACM Press.
2. S. Basu, M. Bilenko, and R. J. Mooney. A Probabilistic Framework for Semi-supervised Clustering. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68, New York, NY, USA, 2004. ACM Press.
3. K. Bennett, P. Bradley, and A. Demiriz. Constrained K-Means Clustering. Technical report, Microsoft Research, 2000. MSR-TR-2000-65.
4. M. Bilenko, S. Basu, and R. J. Mooney. Integrating Constraints and Metric Learning in Semi-supervised Clustering. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 11, New York, NY, USA, 2004. ACM Press.
5. I. Davidson and S. Basu. Clustering with Constraints. In *Tutorial at the The 5th IEEE International Conference on Data Mining, Houston, Texas, USA, 27-30 November*, 2005.
6. I. Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *PKDD'05: Principles of Knowledge Discovery from Databases*, pages 59–70, 2005.
7. I. Davidson and S. S. Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *SIAM'05: Society for Industrial and Applied Mathematics International Conference on Data Mining International Conference in Data Mining, Newport Beach, California, USA, 21-23 April*, 2005.
8. M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005.
9. D. Gunopulos, M. Vazirgiannis, and M. Halkidi. From Unsupervised to Semi-supervised Learning: Algorithms and Evaluation Approaches. In *SIAM'06: Tutorial at Society for Industrial and Applied Mathematics International Conference on Data Mining, Bechesoa, Marylan, USA, 20-22 April*, 2006.
10. M. Halkidi, D. Gunopulos, N. Kumar, M. Vazirgiannis, and C. Domeniconi. A Framework for Semi-Supervised Learning Based on Subjective and Objective Clustering Criteria. In *ICDM'2005: IEEE International Conference on Data Mining*, pages 637–640, 2005.
11. M. Halkidi and M. Vazirgiannis. Clustering Validity Assessment: Finding the Optimal Partitioning of a Data Set. In *ICDM'2001: IEEE International Conference on Data Mining*, pages 187–194, 2001.
12. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-based, multidimensional data mining. *Computer, IEEE Computer Society Press*, 32(8):46–50, 1999.
13. I. Kopanas, N. M. Avouris, and S. Daskalaki. The Role of Domain Knowledge in a Large Scale Data Mining Projects. In I. P. Vlahavas and C. D. Spyropoulos, editors, *Methods and Applications of Artificial Intelligence, Second Hellenic Conference on AI, SETN 2002*, volume 2308 of *Lecture Notes in Computer Science*. Springer, 2002.
14. M. Vazirgiannis, M. Halkidi, and D. Gunopoulos. *Quality Assessment and Uncertainty Handling in Data Mining*. Springer-Verlag, LNAI Series, 2003.
15. K. Wagstaff and C. Cardie. Clustering with Instance-level Constraints. In *ICML'00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
16. K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means Clustering with Background Knowledge. In *ICML'01: Proceedings of 18th International Conference on Machine Learning*, pages 577–584, 2001.

StreamSamp

DataStream Clustering Over Tilted Windows Through Sampling

Baptiste Csernel[1], Fabrice Clerot[2], and Georges Hébrail[3]

[1] École Nationale Supérieure des Télécommunications, 46 rue Barrault, 75013 Paris
Département Informatique et Réseaux. csernel@enst.fr
[2] France Télécom R&D, 2 avenue P. Marzin, 22307 Lannion
fabrice.clerot@orange-ft.com
[3] École Nationale Supérieure des Télécommunications, 46 rue Barrault, 75013 Paris
Département Informatique et Réseaux. hebrail@enst.fr

Abstract. This article presents StreamSamp, a new algorithm for data stream summarizing. The approach proposed here is simply based on the fundamental technique of sampling the entering stream followed by an intelligent storage of the generated samples thus allowing for the study of the entire stream as well as a part of it. This algorithm is of course one pass and benefits from its capability to process large amounts of high speed data independently of its dimensionality. The versatility of this summarizing algorithm as a pre-processing bmock is contrasted with other more dedicated state-of-the-art algorithms and its performances are illustrated on a clustering task by a comparison with the performances of CluStream, a reference algorithm in the field for this task.

1 Introduction

In the last few years, a growing number of applications or industrial systems have had to deal with data streams that need to be controlled or measured in one way or another. A data stream is defined here as an infinite sequence of elements generated continuously at a fast rate in regard to the treatment and storage capacity available.

Examples of such data streams can be found in applications as different as web site logs ([1]), mobile or main phone telecommunication tickets ([2]), or in sensor data, for traffic sensors for instance, but also for stock indexes ([3]) or meteorology ([4]). It thus seems reasonable to assume that, in the long run, most networked economies will generate, for control or marketing purpose, important quantities of data streams.

Depending on the application type, the treatments that are required on data streams range from the design of SQL-like systems adapted to data streams to the detection of specific patterns within the streams and this area ("stream-mining") is becoming very active with contributions from the data-mining, data-base or machine learning communities.

In this article, we introduce a new versatile data stream summary, StreamSamp and illustrate its performance on the unsupervised data stream clustering task.

1.1 Problem Definition

The classic unsupervised clutering problem is, for a given set of elements, to partition them into a given number of classes in order to regroup in the same class similar elements, where this similarity is defined in regard to a given metric or an objective function chosen according to the problem at hand. This problem has been given a lot of attention in the data mining community for years due to its many uses in all kind of applications.

When it comes to data streams, the problem remains the same but a number of additional constraints have to be taken into account which prevent the use of previously developed methods. Namely, considering the nature of a data stream, algorithms are allowed at most one pass over the data, and, despite the stream being endless, must run using a fixed amount of memory while taking into account the evolution of the streaming data over time. Last but not least, since the data rate in streams is expected to be high, the treatment time per element has to be accordingly low.

1.2 Related Work

The data stream mining community has already put forward a number of clustering algorithms for data streams. Among these, only two will be considered here. The first one called Smaller Space and developed by S. Guha et. al. in [5] has a concept slightly similar to the one described here. The other one, proposed by C. Aggarwal et al. in [6] is certainly one of the most successful recent algorithms, judging both by its performance and how it was received by the community.

Smaller Space is based on a divide and conquer approach. The algorithm operates by clustering successive chunks of data, each time keeping only the centroids resulting from the clustering. When a sufficient number of chunks have been read, all the so far calculated centroids are clustered into higher level clusters. This time again, only the centroids are kept. The process is then repeated at each level of the hierarchy. This algorithm is efficient, however, it can only provide results for the whole stream, and not for any given portion of it as more recent algorithms can. Besides, it only gives the center of each class and no information regarding the radius or the population of a class reducing the value of the resulting information.

CluStream is mainly based on two ideas. The first one, inspired by the classic clustering algorithm BIRCH [7] is to use micro classes to summarize the data stream in central memory by keeping only a small sketch called a Cluster Feature Vector (CFV) for each of them. A CFV is a structure containing for a given micro class, its number of elements, and for each variable the sum of its values on the elements contained by the micro class, as well as the sum of all their values squared. In the case of CluStream, these two sums are also kept for the

timestamps of each element which are treated as just another variable. Upon receiving a new stream element, the algorithm assigns it to its closest micro class and updates this class' CFV information accordingly. If no micro class is found close enough to the new element, a new micro class is created centered on that element, while at the same time two older micro clusters are merged or one is destroyed to preserve a constant number of micro classes while still remaining representative of the stream. The evolution of all the micro classes is tracked down through the use of unique identifiers which enable to follow the evolution of each micro cluster in CluStream even through several fusions.

The second idea is to take snapshots of the state of the algorithm at predefined times by saving on disk the CFV of all micro classes. Those snapshots are then saved according to a geometric time frame designed to hold more snapshots for times closer to the current time and less and less of them as they age (this is referred to as tilted windows). The mathematical properties of the CFVs, the fact that they include timestamps information about the clusters, and the records of the identifiers of merged clusters allow for substraction operations to be executed between two snapshots to obtain the contents that went through the stream between the dates at which they were taken and thus keep track of the evolution of the micro clusters.

Then, the final clustering step is achieved by clustering the micro classes through a classic clustering algorithm using the CFV as elements weighted by the number of elements they represent. Such clusters built from carefully selected snapshots so as to represent a given portion of the stream are really representative of the information contained.

This principle of a two step clustering based first on an online part using some micro clusters and a system of snapshots and an offline part that clusters the micro cluster to produce the final result has appealed the community with its flexibility and the many post-treatments which can be applied on the stored snapshots. A number of variations have been published with different summary structures or clustering algorithms [8–10].

2 StreamSamp

The work presented in this paper is based on a two step approach inspired from CluStream, but whereas CluStream is based on BIRCH and the kmean algorithm, StreamSamp is based on sampling.

2.1 Principle

The algorithm which builds the StreamSamp summary is quite simple. As elements of the stream go by, they are sampled in a purely random way at a fixed sampling rate α and kept in a sample. When that sample has reached a given size T it is stored with the dates marking its starting and ending point and it receives the order 0. Another sample is then started to replace the previous one in the stream's treatment (see fig. 1 on page 5).

It is of course impossible to store all the samples thus created because of the unending nature of the stream. To solve this problem, we design the summary structure so as to make samples cover periods of time of varying length, the older the sample, the longer the covered period of the stream. In practice, when the number of samples of a given order o has reached a given limit L, the two oldest samples of this order are fused into a single sample of size T and of order $o+1$. This new sample covers the same period of time as it's two parent samples put end to end (see fig. 2 on the next page) and is built by randomly keeping $T/2$ elements of each of its parent samples.

A sample on any given part of the stream or the whole stream can then be created by fusing together a number of samples (fusing all the stored samples providing a sample for the entire stream). If samples of different orders are to be fused, they must be weighted in order to retrain an equal representativity for each element of the *final sample*. This is achieved by giving a weight of 2^o to all the elements of a sample of order o. The *final sample* thus constructed then remains representative of the data it summaries.

It is on this *final sample* built according to the portion of the stream which has to be analyzed that all the real analysis will be made. Since this sample is a normal weighted sample, all classic techniques of data analysis can be directly applied to it. In this article, only the clustering process has been evaluated by using the kmeans algorithm on the samples, however, any algorithm able to deal with weighted samples could be applied to this summary, which makes it a very versatile tool.

Finally, to evaluate the quality of the resulting clustering, a score equal to the sum of the squared distances of all the elements of the reference sample to their cluster centroid is calculated. This score is known as the intra cluster inertia (also SSQ, for Sum of SQuare in [6]). The kmeans algorithm is thus repeatedly launched 10 times, and each time, the intra cluster inertia is calculated using the *final sample* as a reference. The final classes selected are those produced by the kmeans launch that resulted in the highest score and represent the final result of the algorithm. Each of these classes is representative of the classes of the streamed data and can be used to give estimates of the number of elements, the center, and the radius of the original classes.

2.2 Tests

Series of tests have been run so as to assess the performance of StreamSamp and compare it with CluStream, and the tests settings will be presented here. The data file is read at a randomly variable speed to simulate changes in the stream's speed in a real word type of application. The data is then processed by the algorithms, both of them parametrized so as to use the same amount of memory. Once the processing of the stream is done, the actual tests are run on the resulting summaries, samples for StreamSamp, and snapshots of the micro cluster's CFV for CluStream. The file concerning the horizon of the stream on which the accuracy of the clustering needs to be tested is then built according to each algorithm's process and the resulting final file is clustered 10 times in both

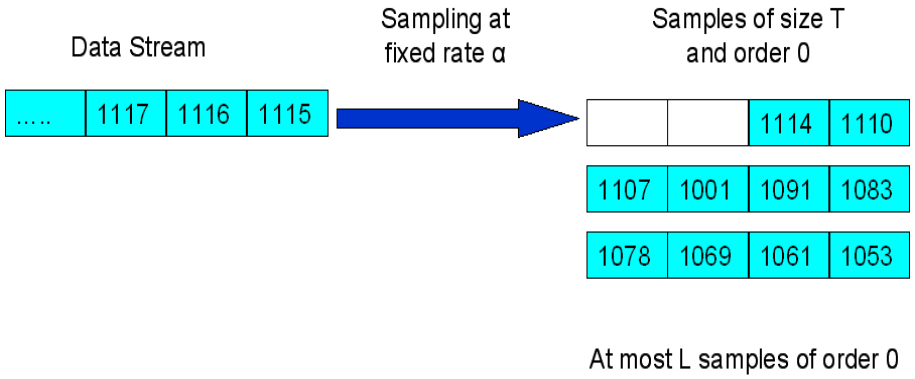


Fig. 1. Archiving of a new sample for $L = 3$

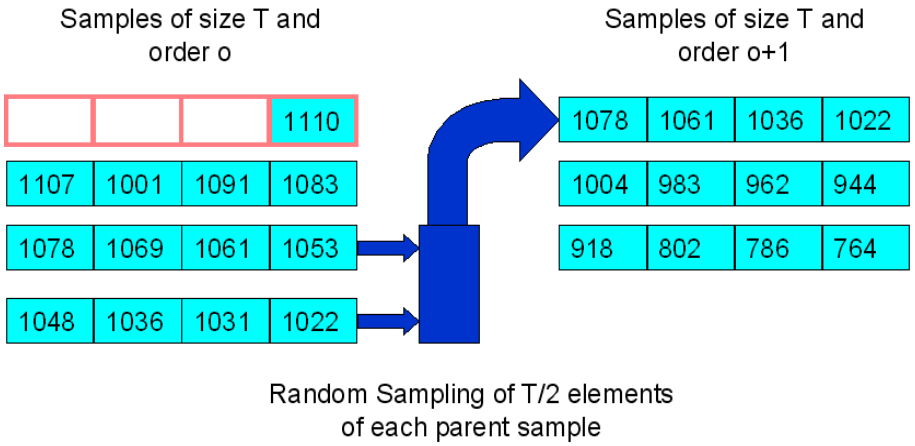


Fig. 2. Archiving of a new sample for $L = 3$

cases using the k-means algorithm and the same number of expected clusters; ($k = 5$). The best pass is then selected by *estimating* it's intra cluster inertia, using the *final sample* file (StreamSamp) or the centroids of all the micro clusters extracted from the CFV (CluStream) as a reference file.

The quality of the final clusterings is then assessed by *calculating* the intra-cluster inertia using the original data file of the stream as a reference file.

For the purpose of this test, we used the data intrusion set given at the 1999 KDD cup and used by Aggarwal to evaluate the performance of his algorithm in [6]. The parameter setting for CluStream were the same has the ones that were used in that article. For StreamSamp, the size of the samples and the maximum order limit were chosen so as to make it use as much memory as CluStream ($T = 200L = 4\alpha = 0.5$).

In order to evaluate the performance of both algorithms, tests have been run on various portions of the streams. In one series of analysis have been performed, in one fig. 3 on the facing page, the horizon of the clustered elements ranges from the last element of the stream received to encompass an ever larger number of elements, each time including more of the past streamed data in the analysis. In an other fig. 4 on the next page, the horizon starts with the first element and then grows as well to include newer elements. Finally, in the last one, one of the experiences described by Aggarwal in [6] has been reproduced, using the published score for CluStream's performance. In that test, the horizon starts at various places in the stream and each time encompasses 50000 elements.

In all tests it appears that StreamSamp's gives much better results than Clustream with a performance increase ranging between 10 and 15%. It does however appear that performance on the oldest data does not increase nearly as much as on more recent elements.

The speed of StreamSamp has also been assessed during those tests; the results are shown in fig. 4 on the facing page. The speed reported here for CluStream comes from [6] as our own java implementation of CluStream didn't prove as fast. Still, the increase of performance of StreamSamp over CluStream is of an order of magnitude.

2.3 Discussion

We have chosen to test CluSamp and CluStream over large time horizons encompassing portions of the stream lying both close and far in the stream history so as to assess how CluSamp and CluStream deal with old and new stream data and how they can be used to run analysis on precise chunks of data having occured far in time in the stream history. Since both algorithms only take into account relevant data, the performance increase observed for CluSamp over CluStream isn't as dramatic as the one observed for CluStream over Stream in [6]. Furthermore, the results indicate that the performance of StreamSamp might start to decrease slightly when very old elements are included in the horizon. This can be explained by the increasing weight of old samples for a fixed sample size , which slowly decreases the representativity of the data. However, StreamSamp

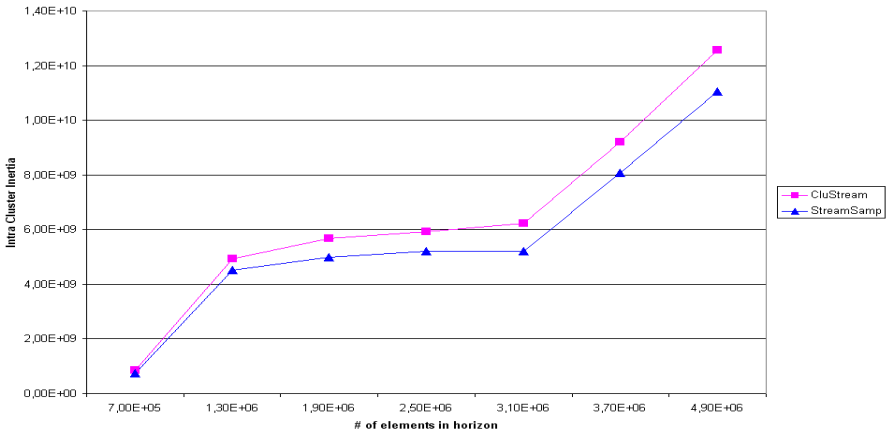


Fig. 3. Result Curve for an Increasing Quantity of Stream Data Starting with the Last Element

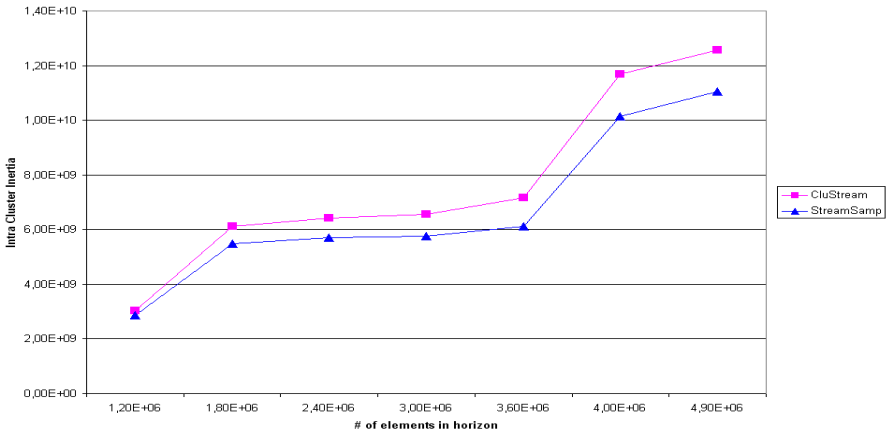


Fig. 4. Result Curve for an Increasing Quantity of Stream Data Starting with the First Element

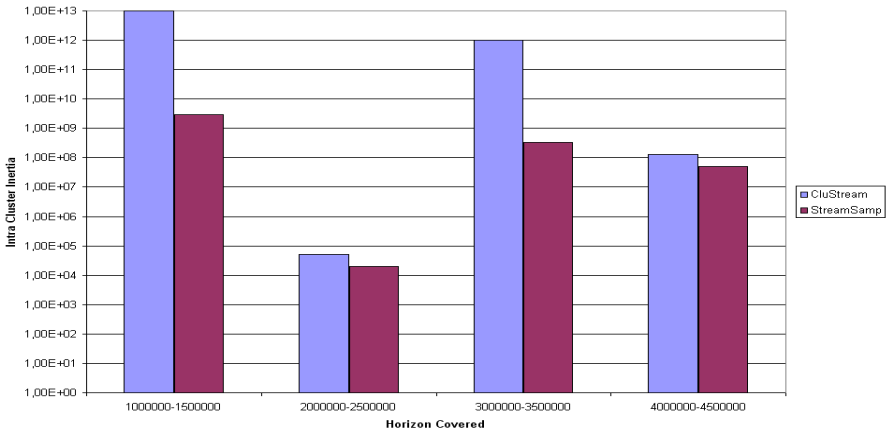


Fig. 5. Result Scores for Various Chunks of the Stream

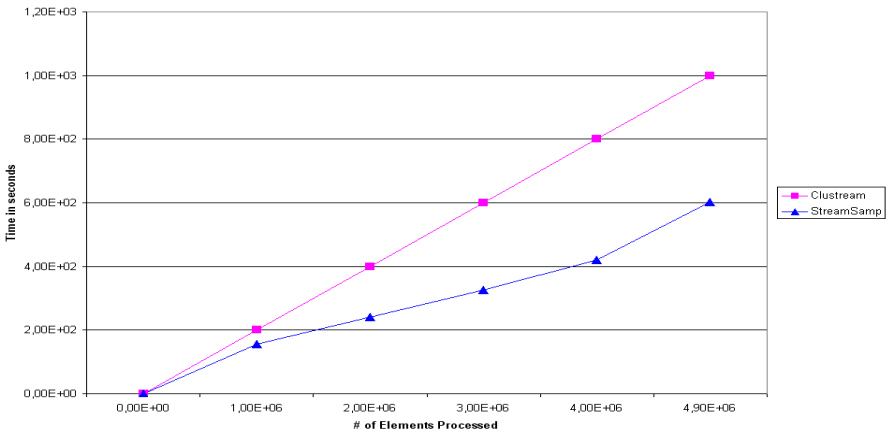


Fig. 6. Result Curve for Stream Processing Speed

outperforms CluStream in all the tests run, proving that samples are a more effective representation than micro clusters even for clustering tasks.

Besides, if only the case of clustering has been studied here, the Streamsamp summaries can be used for any type of analysis to obtain information on the stream's data distribution, making it an extremely flexible and reusable tool.

Moreover, StreamSamp's execution speed is not affected by the dimensionality of the stream elements and is only slightly affected by the length of the stream since the number of fusions to realize at the arrival of a new element increases linearly with the number of stored samples, as shown in fig. 6 on the preceding page. This is especially interesting when dealing with complex data, and, compared to an algorithm like CluStream which requires an important amount of online computation at the arrival of each new stream element, StreamSamp has a very straightforward online part which allows it to deal with very high speed streams.

Another feature of StreamSamp lies in the fact that since the granularity of the samples is a function of the stream speed and not of a fixed algorithm parameter or a function of time; the amount of samples stored is thus more pertinent. At a time when the stream has a speed burst, meaning that more information and probably of a different kind is going through, more samples will be taken than at times when the stream runs at an average speed. This insures that even for old samples, active parts of the stream will be more accurately represented, which counters to some point the problem of aging samples.

Finally StreamSamp has a very small number of parameters that need to be set beforehand for the on-line part. Only the sample rate, the maximum size of each sample and the maximum number of samples of each order is to be set according to the expected speed of the sampled stream and the size of the available memory storage. This is in strong contrast to most other existing algorithms whose on-line part have to be parametrized with a larger number of parameters, most of them depending both on the stream's speed and, more importantly, on the nature of the data.

If it is rather to be compared with other stream sampling algorithms such as for instance reservoir sampling [11] [12], StreamSamp offers a radically different and more flexible approach. While former algorithm adapted to the content of the stream by modifying the chance of an element in the sample to be updated while maintaining the same weight for all the elements in the sample, StreamSamp does the opposite by maintaining a constant sampling rate while changing the weight of the sampled elements with time. This allows StreamSamp to produce a sample of the whole stream or of only a selected portion of the stream whereas former algorithms could only maintain a sample of the whole stream.

3 Conclusion

In this paper, StreamSamp, a new algorithm for data stream clustering and stream data analysis has been presented. Its performance have been assessed for clustering tasks and it has been shown to outperform the CluStream algorithm

in terms of both clustering accuracy and execution speed. Thus proving it to be a particularly well suited stream clustering algorithm for high speed data streams of large elements, offering an attractive alternative to other existing stream clustering algorithms. Furthermore, the summaries obtained are highly versatile and can be used off-line for any other kind of statistical processing which can be applied to a weighted sample.

References

1. Gilbert, A., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Quicksand: Quick summary and analysis of network data. Dimacs technical report, Department of Computer Science, Brown University (2001)
2. Cortes, C., Fisher, K., Pregibon, D., Rogers, A.: Hancock: a language for extracting signatures from data streams. In: Proceedings of the 2000 KDD Conference. (2000) 9–17
3. http://www.traderbot.com: (Traderbot home page)
4. Carney, D., Cetinternel, U., Cherniack, M., Convey, C., Lee, S., , Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring streams - a new class of dbms applications. Technical Report CS-02-01, Department of Computer Science, Brown University (2002)
5. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: IEEE Symposium on Foundations of Computer Science, IEEE (2000)
6. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proceedings of the 29th VLDB Conference, Berlin, Germany (2003)
7. Zhang, T., Ramakrishnan, R., Livny, M.: Birch an efficient data clustering method for very large databases. In: SIGMOD, Montreal, Canada, ACM (1996)
8. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L., Meyerson, A.: Streaming-data algorithms for high quality clustering. In: Proceedings of the 18th ICDE 2002 Conference, IEEE (2002)
9. Jin, H., Wong, M., Leung, K.: Scalable model based clustering by working on data summaries. In: Proceedings of the 3rd ICDM 2003 Conference, IEEE (2003)
10. Park, N.H., Lee, W.S.: Statistical grid based clustering over data streams. In: SIGMOD Record. Volume 33., ACM (2004)
11. Vitter, J.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1) (1985) 37–57
12. Babcock, B., Datar, M., Motwani, R.: Sampling from a moving window over streaming data. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, ACM SIAM, ACM Press (2002) 633–634

Structural Analysis of the Web

Pratyus Patnaik[1], Sudip Sanyal[1]

[1] Indian Institute of Information Technology, Allahabad, India
pratyus@ug.iiita.ac.in, ssanyal@iiita.ac.in

Abstract. World Wide Web has evolved exponentially since its inception. Today, it has become important for the algorithms of the web applications like searching, web-crawling, community discovery to exploit the information hidden in the hyperlink graph of the Web. This is the main driving force of the webmining community. We present in this paper an extensive analysis of the web, purely, based on the graph analysis algorithms. Prior works in the graph based analysis of the Web have been based on certain criteria and themes. But, we believe, for the web, which has evolved stochastically, only a pure graph based analysis can give us the true insights. We have carried out our analysis on isomorphic subgraphs of the Web to arrive at our conclusions. We reaffirm that the Web, is indeed a Fractal. Each structurally isomorphic subgraph shows the same characteristics as the Web and follows the classical Bow-tie model.

Keywords: Web Mining, Subgraph Isomorphism, Graphs, Fractals.

1 Introduction

The Web is an ever growing repository of large amount of information, spread across several servers in a complicated network. With the advent of Peer2Peer, web publishing every user is a disseminator of information. With every new webpage, a new node and with every link, a new directed edge is added to the web graph.

This growth in the size of the web presents an important task of mining and extracting relevant information from the hyperlink graph which can be exploited by simple searching and crawling algorithms as well as by advanced web applications such as web-scale data mining, community extraction, construction of indices, taxonomies, and vertical portals. In the recent past many application have surfaced which exploit the knowledge of the hyperlink structure of the web. Few such applications are the advanced search applications [4, 5, 6], browsing and information foraging [7, 8], community extraction [9], taxonomy construction [10].

Substantial amount of work has been carried out in the recent past [1, 2, 3]. [1] is very theoretical, and proposes stochastic models to explain the hyperlink structure of the Web. [3] talks about Small World Network and Scale Invariance in the structure of the Webgraph. It also proposes the classical bow-tie structure based on the various graph parameters (discussed in the next section). In [2], Kumar et. al. have extended the above model and have shown the self-similarity in the Web i.e., each thematically unified region displays the same characteristics as the Web at large. They have

characterized sub graphs as collections of Web pages that share a common attribute like keyword, content, location and some randomly generated subgraphs.

But, we believe to capture the true insights on the structure of the web, which has evolved stochastically over the period of its existence, we need to make use of pure graph based sub graph isomorphism algorithms. We applied iterative subgraph isomorphism algorithm on the webgraph to get the subgraphs. We then calculated various graph analysis parameter for those subgraphs. We found that each structurally similar subregion shows the same characteristic as the web and this holds for a number of parameters.

In the subsequent sections we have described our experiments, our results and finally the conclusions. But before delving deeper into the experiment, in the next section we briefly discuss the hyperlink webgraph, the graph parameters and the subgraph isomorphism algorithm.

2 Terminologies and Algorithm

2.1 Web Graph

Our view of the Web as a graph is same as [2] i.e. we ignore the text and other content in pages, focusing instead on the links between pages. In the terminology of graph theory [11], we refer to pages as nodes, and to links as edges. In this framework, the Web is a large graph containing over a billion nodes, and a few billion edges.

2.2 Graph Terminologies

A directed graph consists of a set of nodes, denoted as V and a set of edges, denoted as E. Each edge is an ordered pair of nodes (u, v) representing a directed connection from u to v. The outdegree of a node u is the number of distinct edges (u, v1), ... , (u, vn) (i.e., the number of links from u), and the indegree is the number of distinct edges (v1, u), . . . , (vn, u) (i.e. the number of links to u). A path from node u to node v is a sequence of edges (u, u1), (u1, u2), . . . , (un, v). As the graph is directed, a path from u to v does not imply vice-versa. The distance from u to v is n+1, for the smallest value of n. If no path exists, the distance from u to v is infinity. If (u, v) is an edge, then the distance from u to v is 1.

2.3 Graph Analysis Parameters

A brief description of the parameters we have used in the analysis of the Web graph:

Characteristic Path Length and Diameter. The characteristic path length defines the typical distance from every node to every other node. The diameter represents the maximum possible distance between all the pair of reachable nodes. The Characteristic path length is calculated by finding the median of the means of the shortest paths from each node to every other node.

Clustering Coefficient. It is defined as the mean of the clustering indices of all the nodes in the graph. To find it, we find the neighbors of the node and then find the number of existing links amongst them. The ratio of the number of existing links to the number of possible links gives the clustering index of the node.

Centrality and Centralization. The degree centrality for a node is defined as:

$$C'_D(p_k) = \frac{\sum_{i=1}^{n}(a(p_i, p_k))}{n-1}$$

where a (pi, pk) is 1 iff pi and pk are directly connected in the direction from pi to pk. The degree centrality of a point is useful as an index of a potential communication ability.

Degree Centralization. The centralization of a network is calculated as the ratio of the centrality of each node of the network with a star network of the same size.

Betweenness Centrality. It is based upon the frequency with which a point falls between pairs of other points on the shortest or geodesic paths connecting them.

Closeness Centrality. It is related to the control of communication in a somewhat different manner. A point is viewed as central to the extent that it can avoid the control potential of others.

2.4 Web Graph Characteristics

As mentioned earlier Small World Network and Scale Invariance are two important characteristics reported in earlier works [1, 2, 3].

Small World Network. It is a complex network in which the distribution of connectivity is not confined to a certain scale, and where every node can be reached from every other by a small number of hops or steps. The Web was shown to exhibit this characteristic first by [3], since then many have reinforced this assertion.

Scale Free Networks. Scale-free Networks, are the outcome of random construction processes. One of their common property is that the vertex connectivities follow a scale free power-law distribution. Power-law distribution states that for a positive integer, the probability of the value i is directly proportional to i^(-k) for a small positive number k. Scale free Networks are generic and are preserved under random degree preserving rewiring. They are Self Similar and Domain Independent [12].

Scale-free networks usually contain centrally located and interconnected high degree nodes, which influence the way the network operates. For example, random node failures have very little effect on a scale-free network's connectivity or effectiveness; but deliberate attacks on such a node can lead to a complete break down [12].

2.5 Subgraph isomorphism Algorithm

Although, graph isomorphism is a classical problem, this NP hard problem has no foolproof algorithm yet. Algorithms that we considered for our analysis were FSG [13], gFSG [14], gSPAN [15], GREW [16] and SUBDUE [17]. All these graph algorithms, except the last one, cannot handle graphs of more than 1000 nodes.

SUBDUE is heuristics based and hence is able to work on large unlabeled graphs. But, it gives an approximate result. Input to the SUBDUE system was the single web graph. SUBDUE outputs substructures that best compress the input dataset according to the Minimum Description Length (MDL) [18] principle. MDL has the fundamental idea that any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally [17]. Since we wanted to select the hypothesis that captures the most regularity in the data, we looked for the hypothesis with which the best compression can be achieved.

SUBDUE performs a computationally-constrained beam search which begins from substructures consisting of all vertices with unique labels. The substructures are extended to generate candidate substructures. Candidate substructures are then evaluated according to how well they compress the Description Length (DL) of the dataset. Compression takes place by replacing all the subgraph instances by a single vertex. The DL of the input dataset G using substructure S can be calculated using the following formula,

$$I(S) + I(G|S)$$

where, S is the substructure used to compress the dataset G. I(S) and I(G|S) represent the number of bits required to encode S and dataset G after S compresses G. This procedure repeats until all substructures are considered or user-imposed computational constraints are exceeded. At the end of the procedure SUBDUE reports the best compressing substructures. This can also be carried out iteratively.

3 Experimental Details

We crawled webdata using WebMine [19]. We collected a hyperlink graph of websites of 1 million nodes and nearly 2 million edges. We have restricted our analysis to website graph because of the limitation of the SUBDUE algorithm in handling very large graphs. The graph was partitioned to make sample graphs of size ranging from 30K nodes to 100K nodes. Then SUBDUE algorithm was used to generate subgraphs of the sample graphs and also the complete webgraph. SUBDUE iteratively ran thrice for each sample to generate subgraphs at three levels of compression. WebMine tool was again used to compute various graph analysis parameters for the web graph.

4 Results and Interpretation.

We present our results in the tables below:

Table 1. Results for indegree and outdegree exp. coefficients for the power law equation.

Sample	Number of Nodes	Indegree ex. Coeff. (k)			Outdegree ex. Coeff.(k)		
		I1	I2	I3	I1	I2	I3
Sample1	100K	2.17	2.16	2.16	2..23	2.21	2.18
Sample2	80K	2.09	2.06	2.04	2.11	2.07	2.07
Sample3	70K	2.11	2.08	2.06	2.15	2.13	2.12
Sample4	50K	2.06	2.06	2.05	2.12	2.11	2.11
Sample5	30K	2.13	2.13	2.10	2.13	2.12	2.12

Table 1 shows the coefficient of the power law for indegree and outdegree. I1, I2 and I3 represents the iteration level of SUBDUE. We see that the sample graphs and their subgraphs adhere to the power law, which is a property of Scale free Graph. And the value of the Degree coefficient is near about 2.1, conforming to the earlier calculations [2, 3].

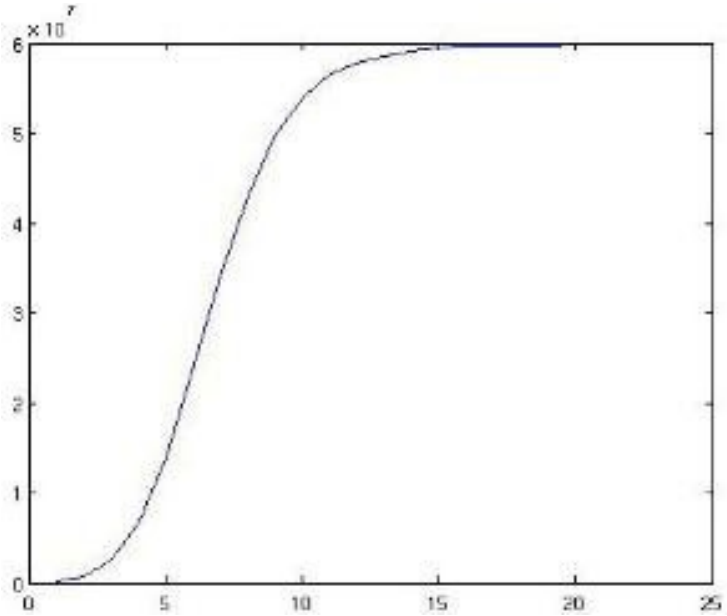


Fig. 1. Graph diameter Vs Number of nodes

Figure 1 shows the graph for Diameter Vs Number of nodes in the graph. Diameter was calculated for the sample graphs and their respective subgraphs. We can see it follows nearly a logarithmic increment. For the Web graph of 100K nodes it was 18. The diameter saturates near the value 17. This again reaffirms the scale free nature of the web.

Table 2 gives the range of values of parameters for all the samples' three iterative subgraphs, formed after the compression in SUBDUE, gave the following values for:

Table 2. Graph Parameter values.

Parameter	Range
Clustering Coefficient	0.13-0.15
Betweenness Centralization	0.03-0.04
Closeness Centralization	0.36-0.41

The range of value of clustering index clearly indicates that the samples' subgraphs are much clustered and, so, exhibit small world network. The value range of Betweenness Centralization indicates that there are a few nodes in the sample graphs that lie in the path of most of the pairs and, hence have a significant influence over the communication of the other nodes. This indicates the presence of a core inside the subgraphs. The value range of closeness centralization indicates that there are a few nodes that are close to most of the other nodes in the subgraphs and, thus again, indicates towards the existence of a core in the sample graph.

Another interesting parameter that we evaluated was the compression quotient after every iteration level of SUBDUE.

$$Q = (Vf+Ef)/(Vi+Ei)$$

Where, Vf and Ef are number of Vertices and Edges in the compressed graph and Vi and Ei are the respective numbers in the original graphs.

Table 3. Results for compression Quotient

Sample	Number of nodes	Q1	Q2	Q3
Sample1	100K	0.931	0.847	0.863
Sample2	80K	0.891	0.871	0.875
Sample3	70K	0.953	0.941	0.949
Sample4	50K	0.924	0.911	0.918
Sample5	30K	0.956	0.892	0.922

We notice that the compression is more at the last level of SUBDUE iteration, which also supports the scale invariance in the sample webgraphs.

5 Conclusion.

We have carried out a pure graph based analysis of the web. And we have concluded from an entirely structural point of view that the Web is a fractal - It has cohesive sub-regions, at various scales, which exhibit the similar characteristics as the web for a lot of parameters. Each isomorphic subgraph nearly follows the classical Bow-Tie structure, with a robust core. This scalefree structural self similarity in the Web holds the key to building the theoretical models for understanding the evolution of the World Wide Web [2]. And further, this knowledge can be exploited while addressing the issues like security and routing measures for data streams, searching the internet aand also e-marketing.

References

1. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, “Stochastic models for the web graph”, In Proc. 41st FOCS, pages 57–65, 2000.
2. S. Dill, R. Kumar, K.S. Mccurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins, “Self-similarity in the web”, ACM Trans. Inter. Tech., 2(3):205--223, 2002.
3. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener, “Graph structure in the web”, In Proc. 9th WWW, pages 309–320, 2000.
4. S. Chakarbarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan and S. Rajagopalan, “Automatic resource compilation by analyzing hyperlink structure and associated text”, In Proceedings of the 7th WWW/Comput. Netw. 30, 1–7, 65–74, 1998.

5. S. Chakarbarti, B. Dom, D. Gibson, S. Ravi Kumar, P. Raghavan, S. Rajagopalan and A Tomkins, “Experiments in topic distillation”, In SIGIRWorkshop on Hypertext Information Retrieval on the Web, 1998.
6. K.Bharat and M. Henzinger, “Improved algorithms for topic distillation in hyperlinked environments”, In Proceedings of the 21st SIGIR, 104–111, 1998.
7. R.A. Botafogo and B. Shneiderman, “Identifying aggregates in hypertext structures”, In Proceedings of the 3rd Hypertext Conference, 63–74, 1991.
8. J.Carriere, and R. Kazman, “WebQuery: Searching and visualizing the web through connectivity”, In Proceedings of the 6th WWW 29, 8–13, 1257–1267, 1997.
9. R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins, “Trawling the web for cyber communities”, In Proceedings of the 8th WWW/Comput. Netw. 31, 11-16, 1481–1493, 1999.
10. R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins, “Extracting large scale knowledge bases from the web”, In Proceedings of the Conference on Very Large Data Bases, 639–650, 1999.
11. F. Harary, “Graph Theory”, Addison Wesley, 1975.
12. L. Li, D. Alderson, R. Tanaka, J.C. Doyle, W. Willinger, ”Towards a Theory of Scale-Free Graphs: Definition, Properties, and Implications”, 2005.
13. M. Kuramochi, and G. Karypis, “Discovering Frequent Subgraphs”, 2001
14. M. Kuramochi, and G. Karypis, “Discovering Frequent Geometric Subgraphs”, 2002
15. X. Yan and J. Han, ”gSpan: Graph-Based Substructure Pattern Mining”, 2002.
16. M. Kuramochi, and G. Karypis, ”GREW—A Scalable Frequent Subgraph Discovery Algorithm”, 2003.
17. Cook Holder, et. al., “Subdue: Compression-based Frequent Pattern Discovery in Graph Data”, Proceedings of the ACM KDD Workshop on Open-Source Data Mining, 2005.
18. P. Grunwald, “Tutorial Introduction to the Minimum Description Length Principle”.
19. S. Suman and S. Aggarwal, “WebMine: A tool to uncover the web”, 2005.

Author Index