Atelier CIDN
Classification Incrémentale et Détection de Nouveauté

Organisateurs :

Pascal CUXAC, Jean-Charles LAMIREL, Vincent LEMAIRE, Thomas GUYET

PRÉFACE

Le développement de méthodes d'analyse dynamique de l'information, comme le clustering incrémental et les méthodes de détection de nouveauté, devient une préoccupation centrale dans un grand nombre d'applications dont le but principal est de traiter de larges volumes d'information variant au cours du temps. Ces applications se rapportent à des domaines très variés et hautement stratégiques, tels que l'exploration du Web et la recherche d'information, l'analyse du comportement des utilisateurs et les systèmes de recommandation, la veille technologique et scientifique, ou encore, l'analyse de l'information génomique en bioinformatique...

Pour ne prendre en exemple qu'un type d'application sur des données textuelles, force est de constater que les publications sur des méthodes permettant de détecter les ruptures technologiques, les thématiques novatrices, sont très présentes dans les congrès et revues. Cet intérêt est souligné par la mise en place par la Commission Européenne du programme NEST (New and Emerging Science and Technology) dans le cadre du FP6 et du programme FET (Future and Emerging Technologies) dans le FP7.

Lors des deux précédentes conférences EGC, se sont déroulés les premiers ateliers CIDN. Devant l'intérêt du public pour cette thématique nous organisons la troisième édition de cet atelier lors de la conférence EGC'13 à Toulouse.

L'objectif de cet atelier, commun EGC-AFIA, est de réunir des chercheurs confirmés, ainsi que des jeunes chercheurs, autour des problématiques et des applications de la "classification incrémentale", et de la "détection de nouveauté" sur des types de données variées, afin d'échanger nos réflexions sur les travaux en cours ainsi que sur les points bloquants.

Première instance de rencontre entre les associations Extraction et Gestion des Connaissances et Association Française pour l'Intelligence Artificielle, cet atelier est conjointement supporté et organisé par EGC et l'AFIA. Au sein des deux communautés, les thèmes de l'atelier ont été identifiés comme des thèmes porteurs notamment pour créer des liens avec les industriels.

P. CUXAC — INIST-CNRS
J.C. LAMIREL — Synalp-LORIA
V. LEMAIRE — Orange
T. GUYET — IRISA

Membres du comité de lecture

Le Comité de Lecture est constitué de:

Alexis Bondu (EDF R&D),
Fabrice Clérot (Orange Labs),
Pascal Cuxac (INIST-CNRS),
Bernard Dousset (IRIT),
Claire François (INIST-CNRS),
Thomas Guyet (IRISA),
Hatem Hamza (Orange),
Pascale Kuntz-Cosperec (Polytech'Nantes),
Jean-Charles Lamirel (Synalp LORIA),
Vincent Lemaire (Orange Labs),
Gaelle Loosli (Polytech'Clermont-Ferrand),
Jean Rohmer (ESILV),
Christophe Salperwyck (Orange Labs),
Fabien Torre (Université Lille 3)

TABLE DES MATIÈRES

Conférence invitée

Similarity Matching in Streaming Time Series
Alice Marascu 1

Articles sélectionnés

Incremental Novelty Detection applied to Diachronic Scientometrics
Aneesh Sreevallabh Chivukula, Jean-Charles Lamirel 3

Growing Self-organizing Trees for Knowledge Discovery from Data
Nhat-Quang Doan, Hanane Azzag, Mustapha Lebbah 21

Incremental mining of frequent sequences from a window sliding over a stream of itemsets
Thomas Guyet, René Quiniou 39

Index des auteurs **57**

Similarity Matching in Streaming Time Series

Alice Marascu

IBM Research, Ireland

Nowadays, the installation of a sensor that sends data continuously and at variable rates became a very easy and affordable task. This allows complex monitoring tasks to be performed and more information on ongoing processes to be discovered in real time. A multitude of sensor data are surrounding us and we are more and more interested in choosing the useful data from this large amount of data, and as fast and as well as possible. Different monitoring algorithms are proposed every year; though, due to the increasing number of sensors and sensors qualities, better methods are necessary continuously. Monitoring sensor network, medical devices, transportation sensors or video data are just several important applications in our days. One important problem is the real time recognition of specific sequence-patterns in streaming time series. The challenges of this problem and a novel and scalable solution will be presented.

Incremental Novelty Detection applied to Complex Text Classification

Aneesh Sreevallabh Chivukula*, Jean-Charles Lamirel**

*Center For Data Engineering,
International Institute of Information Technology, Hyderabad, India - 500 032.
aneesh@research.iiit.ac.in,
http://www.iiit.ac.in/
**Equipe SYNALP - LORIA, INRIA Nancy - Grand Est,
Campus scientifique, 615 Rue du Jardin Botanique, Vandoeuvre-les-Nancy, France - 54600.
jean-charles.lamirel@loria.fr
http://www.loria.fr/

Abstract. This paper presents an original approach to summarize the novelty presented by clustering information in a supervised incremental learning process. The approach relies on the novelty detection paradigm to allow binary discrimination over the information that is similarly labeled in the dataset. Novelty detection filters are used to rank and profile the output of incremental clustering experiments in terms of representative matrices. By discovering normalized features, resulting matrices are able to represent the empirical correlations within clusters without relying on a complex parameter estimation process. An incremental classification strategy is shown to implicitly overcome noisy, imbalanced, highly multidimensional and sparse textual data with high level of similarity between classes. The experimental evaluation on static data indicates that the proposed classifier is comparable to standard classifiers with default parameters. The experimental dataset has 7000 papers related to the classes of a patents classification scheme in the domain of pharmacology.

1 Introduction

Information filtering is concerned with dynamically adapting the distribution of information where both evolving user's interests and new incoming information are taken into account. The aim is being able to detect new subjects following the flow of arrival of new data with a sufficiently good reactivity, which amounts to defining methods which offer an optimal compromise between plasticity and stability Prudent and Ennaji (2004). In such context, user's information interests need to be defined with the combination elaborate mechanisms of query completion, sample-based interrogation and topic driven interrogation. Hence, depending both on the information learned from the flow of user's decisions and from his original query, user's information search can be typified in very various ways such as precise research, exploratory research, thematic research or connotative research. A wide range of machine learning algorithms and information retrieval techniques have been applied to the filtering task, including

Rocchio's linear classifier, k-nearest neighbours, Bayesian classifiers, neural networks, support vector machines and boosting T. and Yang (2001), Schapire et al. (1998), Dumais et al. (1998), (Shankar and George, 2000). However the existing techniques show their limit in the case of the management of unbalanced and sparse session data Lamirel and Créhange (1994).

Inspired by the systemic theory of communication, the novelty detection method Kohonen (1989), (Markou and Singh, 2003) trains a neural network such that the current output of each neuron is a linear combination of its current input and feedback from past output. Once training is complete, if a combination of training data is applied to the filter input, the novelty output will be zero. In filtering context, the novelty detection principle has been applied to select those documents that are similar to a model learned from examples relevant to user's need. By associating several filters to different types of decisions (represented by class labels), the novelty detection paradigm allows thus to propose a session memory component in the learningLamirel and Créhange (1994). Compared to traditional relevance feedback Rocchio (1971), the memory mechanism processes both the relevant and irrelevant user decisions in a homogeneous way. Moreover, detected novelty is used to control the amount of redundancy in filtering results according to the type of user's information need. In this paper we show that incremental extension of the novelty detection technique proposed for information filtering can be fruitfully exploited in the framework of the supervised classification of complex and noisy textual data. For that purpose, the case of multiple class classification is mapped to a series of one class classification problems that work on skewed datasets. Our input datasets are highly dimensional, extremely sparse in features, very imbalanced in class labels and contains several classes of similar profileRaskutti and Kowalczyk (2004).

The reminder of the paper is organized as follows: In Section 2 we present our model. Section 3 reports our experimental investigations. Our paper concludes in Section 4 followed by Bibliography.

2 Classification Model

Orthogonalization Operators The novelty filter algorithm Kohonen and Oja (1976) relies on the properties of orthogonal projection operators. Let there be d distinct Euclidean vectors, denoted by $x_1, x_2, ..., x_d$ which span a subspace $V \subset R^n$ The complement space of V, denoted by $V^{\perp}$, is spanned by all vectors in R^n which are orthogonal to V. Then, using orthogonal projection operators, any arbitrary vector $x \in R^n$ can be uniquely decomposed into the sum of two vectors $\hat{x}$ and $\tilde{x}$ such that $||\tilde{x}|| = min_{||\dot{x}||} ||\ddot{x}||$ for all possible $x = \dot{x} + \ddot{x}$. Thus, for $X \in R^{n*d}$ with x_i vectors as its columns and X^+ as the pseudo-inverse, we derive the orthogonal matrix projection operator XX^+ such that:

$$\tilde{x} = XX^+x \tag{1}$$

and

$$\hat{x} = (I - XX^+)x \tag{2}$$

where I is the identity matrix. Therefore, the component $\tilde{x}$ can be regarded as the residual contribution in x that is left when x is input to a novelty detection filter (NDF) with state matrix XX^+. The matrix XX^+ is also called the transfer function of NDF model.

Learning Rule The NDF model is implemented with a recurrent network of neuron like adaptive elements with internal connections such that every element of the output $\tilde{x}_i$ is assumed to receive a feedback from the other elements $\tilde{x}_j$ through connection weights m_{ij}. Thus, the network output signals $\tilde{x}_i$ are assumed to be linear combinations of the input signals x_i and the feedback signals as:

$$\tilde{x}_i = x_i + \Sigma_j m_{ij} \tilde{x}_j \tag{3}$$

or in matrix notation as

$$\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{M}\tilde{\mathbf{x}} \tag{4}$$

The weights of the feedback connections m_ij associated with the feedback connections characterize the variable internal state of the network. They are initialized to zero and then updated during the training phase. The updates are applied according to the anti-Hebbian learning rule:

$$\frac{d\mathbf{M}}{dt} = -\alpha \tilde{\mathbf{x}} \tilde{\mathbf{x}}^T \tag{5}$$

where α is a small positive parameter that is adaptively modified during the training phase. The feedback weights are iteratively trained to suppress redundant input features by decorrelating strongly correlated output neurons. The decorrelation ensures the detection of data that significantly deviates from the training data.
The above module can now be expressed in terms of the differential equation for $\Phi \in R^{d*d}$ as follows:

$$\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{M}\tilde{\mathbf{x}} = (I - M)^{-1} x = \Phi x \tag{6a}$$

$$\frac{d\mathbf{M}}{dt} = -\alpha \tilde{\mathbf{x}} \tilde{\mathbf{x}}^T \tag{6b}$$

The differential equation for Φ is a Bernoulli equation solved with the Greville's theorem to give a recursive expression to estimate the transfer function of the NDF as follows:

$$\Phi_k = \Phi_{k-1} - \frac{\tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^T}{||\tilde{\mathbf{x}}_k||^2} \tag{7}$$

By introducing the identity matrix in the update rule, for considering separately all the training samples and features in the input dataset, we get the final update rule for ILoNDF as:

$$\Phi_k = I + \Phi_{k-1} - \frac{\tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^T}{||\tilde{\mathbf{x}}_k||^2} \tag{8}$$

where

$$\tilde{\mathbf{x}}_k = (I + \Phi_{k-1}) \mathbf{x}_k \tag{9a}$$

$$\Phi_0 = [0]_{d*d} \tag{9b}$$

The matrices Φ_k and Φ_{k-1} have same eigenvectors with eigenvalues differing by 1 Kassab and Lamirel (2007). Introducing the identity matrix transforms the eigenvectors into a sequence of positive-definite matrices $I + \Phi_{k-1}$. As a result, the orthogonality conditions are no longer satisfied between the space spanned by the filter and that of the reference data. The dimensionality of the space spanned by the model continues to be of the same order as the original description space.

Training The basic idea of novelty detectionKohonen (1989),Markou and Singh (2003) is to learn a new model of the domain dataset being monitored and then to automatically detect novel data that deviates from the normal model. NDF can operate in an online mode without repeated parametric training. However NDF requires that the input data be noise free. Further NDF cannot model high dimensional data that changes over time. Unlike NDF, Incremental data-driven Learning of Novelty Detector Filter (ILoNDF) Kassab and Lamirel (2007) update rule ensures gradual discarding of knowledge regarding old data while considering all features of input throughout training phase. Training ILoNDF on a relevant(or positive) dataset leads to development of the transfer function of the filter under a matrix representation called state matrix or correlation matrix. We can interpret the state matrix of ILoNDF filter as representing the occurrence frequency of each of the features and their relationships in the training data. If a new example is then presented as input, a vector will appear at the output which represents the new features extracted from the input. Compared to classical NDF that exhibits absolute learning problem by being biased towards first available positive class label in dataset, ILoNDF learns features shared between positive and negative classes. Assuming the number of samples in minority classes is sufficient to complete filter training, filter transfer functions are used to build the clustering profiles. These profiles are used to perform multifold stratified cross validation on datasets having both positive and negative class labels. Filters trained on negative labels are not used in ranking, thresholding class or cluster profiles. Figure 1, 2, 3 give the pseudocode for training phase of the classifier.

Algorithm Sparse Stratification for Multifold Cross Validation

Input:
$X = \{x_1, x_2, ..., x_n\}$: Input dataset of multiple class labels;
$C = \{c_1, c_2, ..., c_n\}$: Nominal class label set for input dataset;
$nfolds$: Number of train and test folds.

Output:
Labelled train and test datasets for cross validation in same description space.
function RANDOM SELECTION(X, C)
Parse sparse arff input and labels matrix
Delete sorted instances with missing class labels
Randomize data with number generator
Generate the stratified cross validation folds

end function
for fold in $nfolds$ **do**
Begin
Incrementally write train dataset to disk
Write test dataset to disk in batch mode
End

FIG. 1 – *Partitioning*

Profiling For the binary discrimination problem we generate two ranking proportions based on scalar and vector projections of the input sample features on the filter transfer function. The principle of ILoNDF to capture correlations in the features of training input is used to form the ranks.

Algorithm Count Normalization for a given sparse train fold

Input:
D : The indices to documents in docset;
K : The indices to terms in docset;
WC : The term counts of K in docset D;
$WCSparse$: Term frequency sparse matrix over docset initialized to 0 in Dictionary Of Keys memory format

Output:
$WCSparse$: Updated term frequency sparse matrix.

▷ Term count assignment:

for $k \in K$ **do**
  Begin
    for $d \in D$ **do**
      Begin
        $WCSparse[d,k] \leftarrow WC[d,k]$
      End
  End

▷ Term frequency feature set normalization:

for $k \in K$ **do**
  Begin
    $WCSparse[:,k] \leftarrow \log \frac{|D|}{WCSparse[d,k]}$
  End

▷ Term frequency document set normalization:

for $d \in D$ **do**
  Begin
    $WCSparse[d,:] \leftarrow \frac{WCSparse[d,:]}{||WCSparse[d,:]||}$
  End

FIG. 2 – *Normalization*

– The Direct-Projection Method (DPM) : The 'novelty proportion' quantifies novelty in current input with respect to the data learnt during training. It is defined as:

$$N_{x_i} = \frac{||\tilde{x}_i||}{n\,||x_i||} \tag{10}$$

where n is number of positive training data.
The 'habituation proportion' quantifies the similarity(or redundancy) of the positive test sample with the positive data of previously learnt train samples as:

$$H_{x_i} = 1 - N_{x_i} = 1 - \frac{||\tilde{x}_i||}{n\,||x_i||} \tag{11}$$

The habituation proportion is considered a classifications score of the data x_i indicating the likelihood that the example belongs to the positive class.

– The Vector-Based Projection Method (V-PM) : For high dimensional dataset, we define a lower dimensional profile vector to represent the train dataset in place of the transfer function. Then, the novelty in each feature of the current input sample is characterized

Algorithm Incremental novelty detector training on positive samples for each of the class labels in a given train fold

Input:

$X = \{x_1, x_2, ..., x_n\}$: Labelled train dataset in Compressed Sparse Row sparse memory format for a given fold;

$C = \{c_1, c_2, ..., c_m\}$: Nominal class label set for train dataset. $\mathbf{\Phi}_0 = [0]_{d*d}$.

Output:

$\mathbf{\Phi}_{c_i}$ for $i = \{1, 2, ..., m\}$: Positive filter transfer function in Compressed Sparse Row sparse memory format for each of the class labels in a given train fold.

Partition train sample set X according to class labels C as $X_C = \{X_{c_i}\}$ for $i = \{1, 2, ..., m\}$

for $c \in C$ **do**

  Begin

    $PositiveSamples \leftarrow X_c$

    for each $\mathbf{x}_k \in PositiveSamples$ **do**

      Begin

        $\tilde{\mathbf{x}}_k \leftarrow (I + \mathbf{\Phi}_{k-1})\mathbf{x}_k$

        $\mathbf{\Phi}_k \leftarrow (I + \mathbf{\Phi}_{k-1}) - \frac{\tilde{\mathbf{x}}_k \tilde{\mathbf{x}}_k^T}{||\tilde{\mathbf{x}}_k||^2}$

      End

    $\Phi_c \leftarrow \Phi_{|PositiveSamples|}$

  End

return $\mathbf{\Phi}_{c_i}$ for $i = \{1, 2, ..., m\}$

FIG. 3 – *Training*

by projecting onto the profile vector. To find the profile vector, unit vector representing presence of each feature in dataset is cumulatively projected onto the filter space that is orthogonal to original data space. In a given novelty detector, the periodic variations in the diagonal of state matrix represents train dataset dimensions learnt. In an ultra sparse input sample set, dimensions that are never learnt lead to instability in the filter because normalization does not affect the non-diagonal elements in ILoNDF update rule. In case ILoNDF normalization is not specific enough to capture rare information in dataset, we may use NDF update rule in place of ILoNDF update rule. Therefore to further improve performance of ILoNDF update rule, we propose that the projection of both input magnitude and phase on filter is to be considered in normalization computation of both scalar and vector ranking metrics. Here we assume that projection is a better normalization operation than classical cosine similarity to capture correlations in the input dataset. The updated V-PM is defined as:

$$V_{x_i} = \frac{\tilde{x}_i P_v}{||x_i||\,||P_v||}, \tag{12a}$$

$$P_v = \Sigma_{f \in F} H_f u_f, \tag{12b}$$

$$H_f = 1 - \frac{\Phi u_f}{n u_f} \tag{12c}$$

$$\tag{12d}$$

where u_f is the unit vector associated with the direction of a feature f in the description space.

– Weighting : DPM is a precise proportion that matches the similarity of current sample to the sample that the filter learnt in the past. Whereas the V-PM is a diversity proportion that measures the similarity of current input feature values with respect to the feature information the filter acquired in the past. In general habituation space is a combination and rotation of original description space. To form a proper pattern matching proportion we require a habituation proportion that is a linear combination of both DPM and V-PM proportions. Projection similarity is used to form the V-PM proportions of sparse samples. The DPM scalar ranking is determined by the habituation proportion that measures the input sample component that remains after the novelty in said sample is learnt. The weighted combination of the classification ranks defines a global score calculated as :

$$CS(x_i) = (1 - \lambda)H_{x_i} + \lambda V_{x_i}, \tag{13a}$$

$$\lambda = \frac{standarddeviationofP_v}{maxvalue - minvalueoftheP_vcomponents} \tag{13b}$$

λ is based on the variations in P_v across features because we can view the variance of vector components as indicative of usefulness in forming classification ranks. Figure 4 and Algorithm 1 give the pseudocode for profiling phase of the classifier.

Testing Each test sample input to a novelty detection filter is projected into a search space that is orthogonal to the original description space. When applying ILoNDF to test data of the positive class, a classification score indicating the likelihood that the example is similar to the positive class is computed. ILoNDF decision thresholding process tends to perform well on datasets with more linear than non linear features. The performance of the ILoNDF is quite effective in the case of single-label datasets where each document belongs exactly to one category. However, it fails in the multi-label case where a document may be belong to more than one category. The latter case is relatively more difficult to process for all the existing novelty detection approaches, since high correlation between relevant and irrelevant documents is strongly probable. Algorithm 2 gives the pseudocode for testing phase of the classifier.

Evaluation The Equation 9 is used to find the profile vector for a given input train fold dataset. Without the need for repeated parameter tuning, ILoNDF is robust in dealing with high-dimensional noisy data that is investigated in our experiments. The characteristic vector of a filter's correlation matrix is used in profiling the class labels. Since ILoNDF trains on samples with the positive label, we check the rate of acceptance of positive samples and rate of rejection of negative samples during testing for each positive label. The cumulative accuracy of the classification scheme is then the summation of accuracies obtained for each positive label. The time complexity of the iterative algorithm depends on number of dimensions and samples in the training set. Being based in novelty detection filters, the proposed classifier is less sensitive to outliers. Algorithm 3 gives the pseudocode for evaluation phase of the classifier.

Algorithm 1 Dissemination Threshold value set for each positive class label in a given train fold

Input:

$X = \{x_1, x_2, ..., x_n\}$: Labelled train dataset in Compressed Sparse Row sparse memory format for a given fold;

$C = \{c_1, c_2, ..., c_m\}$: Nominal class label set for train dataset;

$\mathbf{P}_{\mathbf{v}c_i}$ for $i = \{1, 2, ..., m\}$: Representative profile vector in Compressed Sparse Row sparse memory format for each state matrix of novelty detector corresponding to positive label for each of the class labels in a given train fold;

λ_{c_i} for $i = \{1, 2, ..., m\}$: Ranking weight for each profile vector representing a positive class label.

Output:

$threshold_{c_i}$ for $i = \{1, 2, ..., m\}$: Ranking threshold value for each state matrix representing a positive class label in a given train fold..

Partition train sample set X according to class labels C as $X_C = \{X_{c_i}\}$ for $i = \{1, 2, ..., m\}$

for $c \in C$ **do**

Begin

$cindex \leftarrow$ index of $c \in C$

$H_c \leftarrow [0]_{|X_c| \times |C|}$

$P_c \leftarrow [0]_{|X_c| \times |C|}$

$CS_c \leftarrow [0]_{|X_c| \times |C|}$

for all $\mathbf{x} \in X_c$ **do**

Begin

$xindex \leftarrow$ index of $\mathbf{x} \in X_c$

$\tilde{\mathbf{x}} \leftarrow (I + \mathbf{\Phi}_c)\mathbf{x}$

$H_c[xindex, cindex] \leftarrow 1 - \frac{||\tilde{\mathbf{x}}||}{|X_c| \times ||\mathbf{x}||}$

$P_c[xindex, cindex] \leftarrow \frac{\tilde{\mathbf{x}}\mathbf{P}_{\mathbf{v}c_i}}{||x||\,||\mathbf{P}_{\mathbf{v}c_i}||}$

$CS_c[xindex, cindex] \leftarrow$

$(1 - \lambda_{c_i})H_c[xindex, cindex]$

$+(\lambda_{c_i})P_c[xindex, cindex]$

End

$threshold_{c_i} \leftarrow \langle CS_c[:, cindex] \rangle$

End

return $threshold_{c_i}$ for $i = \{1, 2, ..., m\}$

Algorithm 2 Incremental novelty detector testing on positive and negative samples for each of the class labels in a given test fold : Initialization

Input:

$X = \{x_1, x_2, ..., x_n\}$: Labelled test dataset in Compressed Sparse Row sparse memory format for a given fold;
$C = \{c_1, c_2, ..., c_m\}$: Nominal class label set for test dataset;
$\mathbf{\Phi}_{c_i}$ for $i = \{1, 2, ..., m\}$: Positive filter transfer function in Compressed Sparse Row sparse memory format for each of the class labels in a train fold associated with the test dataset.
$\mathbf{P}_{\mathbf{v}c_i}$ for $i = \{1, 2, ..., m\}$: Representative profile vector in Compressed Sparse Row sparse memory format for each state matrix of novelty detector corresponding to positive label for each of the class labels in a train fold associated with the test dataset;
λ_{c_i} for $i = \{1, 2, ..., m\}$: Ranking weight for each profile vector representing a positive class label for each of the class labels in a train fold associated with the test dataset;
$threshold_{c_i}$ for $i = \{1, 2, ..., m\}$: Ranking threshold value for each state matrix representing a positive class label in a train fold associated with the test dataset.

Output:

Accuracy : Figure of merit for classification performance computed on test data projected against transfer functions and representative profiles formed on the train data.

Partition test sample set X according to class labels C as $X_C = \{X_{c_i}\}$ for $i = \{1, 2, ..., m\}$
$TruePositives \leftarrow 0$
$TrueNegatives \leftarrow 0$
$TPS \leftarrow 0$
$TNS \leftarrow 0$
$FoldAccuracy \leftarrow 0$
for $c \in C$ **do**
  Begin
    $cindex \leftarrow$ index of $c \in C$
    $PS \leftarrow X_c$
    $NS \leftarrow X_C - X_c$
    $HPS_c \leftarrow [0]_{|X_c| \times |C|}$
    $PPS_c \leftarrow [0]_{|X_c| \times |C|}$
    $CSPS_c \leftarrow [0]_{|X_c| \times |C|}$
    $HNS_c \leftarrow [0]_{|X_C - X_c| \times |C|}$
    $PNS_c \leftarrow [0]_{|X_C - X_c| \times |C|}$
    $CSNS_c \leftarrow [0]_{|X_C - X_c| \times |C|}$
  End

Algorithm 3 Incremental novelty detector testing on positive and negative samples for each of the class labels in a given test fold : Accuracy

for $c \in C$ **do**
Begin
for all $\mathbf{x} \in X_c$ **do**
Begin
$xindex \leftarrow$ index of $\mathbf{x} \in X_c$
$\tilde{\mathbf{x}} \leftarrow (I + \mathbf{\Phi}_c)\mathbf{x}$
$HPS_c[xindex, cindex] \leftarrow 1 - \frac{||\tilde{\mathbf{x}}||}{|X_c| \times ||\mathbf{x}||}$
$PPS_c[xindex, cindex] \leftarrow \frac{\tilde{\mathbf{x}}\mathbf{P}_{\mathbf{v}c_i}}{||x||\left|\left|\mathbf{P}_{\mathbf{v}c_i}\right|\right|}$
$CSPS_c[xindex, cindex] \leftarrow$
$(1 - \lambda_{c_i})HPS_c[xindex, cindex]$
$+(\lambda_{c_i})PPS_c[xindex, cindex]$

If $CSPS_c[xindex, cindex]$
$\geq threshold_{c_i}$
for $i = \{1, 2, ..., m\}$ **then**
Begin
$TruePositives \leftarrow$
$TruePositives + 1$
End
End
$TPS = TPS + |X_c|$
$TNS = TNS + |X_C - X_c|$
for all $\mathbf{x} \in X_C - X_c$ **do**
Begin
$xindex \leftarrow$ index of $\mathbf{x} \in X_c$
$\tilde{\mathbf{x}} \leftarrow (I + \mathbf{\Phi}_c)\mathbf{x}$
$HNS_c[xindex, cindex]$
$\leftarrow 1 - \frac{||\tilde{\mathbf{x}}||}{|X_C - X_c| \times ||\mathbf{x}||}$
$PNS_c[xindex, cindex] \leftarrow \frac{\tilde{\mathbf{x}}\mathbf{P}_{\mathbf{v}c_i}}{||x||\left|\left|\mathbf{P}_{\mathbf{v}c_i}\right|\right|}$
$CSNS_c[xindex, cindex] \leftarrow$
$(1 - \lambda_{c_i})HNS_c[xindex, cindex]$
$+(\lambda_{c_i})PNS_c[xindex, cindex]$

If $CSNS_c[xindex, cindex]$
$\leq threshold_{c_i}$
for $i = \{1, 2, ..., m\}$ **then**
Begin
$TrueNegatives \leftarrow$
$TrueNegatives + 1$
End
End
End
$FoldAccuracy \leftarrow \frac{TruePositives + TrueNegatives}{TPS + TNS}$
return $FoldAccuracy$

Algorithm Learning profile vectors representing state matrix of novelty detector for each of the class labels in a given train fold

Input:

$X = \{x_1, x_2, ..., x_n\}$: Labelled train dataset in Compressed Sparse Row sparse memory format for a given fold;

$C = \{c_1, c_2, ..., c_m\}$: Nominal class label set for train dataset;

$\mathbf{\Phi}_{c_i}$ for $i = \{1, 2, ..., m\}$: Positive filter transfer function in Compressed Sparse Row sparse memory format for each of the class labels in a given train fold;

$F = \{f_1, f_2, ..., f_d\}$: The feature set to docset.

Output:

$\mathbf{P}_{\mathbf{v}c_i}$ for $i = \{1, 2, ..., m\}$: Representative profile vector in Compressed Sparse Row sparse memory format for each state matrix of novelty detector corresponding to positive label for each of the class labels in a given train fold ;

λ_{c_i} for $i = \{1, 2, ..., m\}$: Ranking weight for each profile vector representing a positive class label.

for $c \in C$ **do**
  Begin
    $\mathbf{P}_{\mathbf{v}c_i} \leftarrow [0]_{d-1*1}$
    $\lambda_{c_i} \leftarrow 0$
    for $f \in F$ **do**
      Begin
        $\mathbf{u}_f \leftarrow 1$ for dimension f, 0 otherwise
        $H_f \leftarrow 1 - \frac{||\mathbf{\Phi}_{c_i}\mathbf{u}_f||}{n \times ||\mathbf{u}_f||}$
        $\mathbf{P}_{\mathbf{v}c_i} \leftarrow \mathbf{P}^{\mathbf{f}}_{\mathbf{v}c_i} + H_f\mathbf{u}_f$
      End
    $\lambda_{c_i} \leftarrow \frac{\sigma \mathbf{P}_{\mathbf{v}c_i}}{\max_{\mathbf{P}_{\mathbf{v}c_i}} - \min_{\mathbf{P}_{\mathbf{v}c_i}}}$
  End
return $\mathbf{P}_{\mathbf{v}c_i}$ and λ_{c_i} for $i = \{1, 2, ..., m\}$

FIG. 4 – *Profiling*

3 Experiments

In this section, we describe our experiments for testing the performance of the novelty detector filter in text filtering environment. Our experiments are performed on a dataset of bibliographical records related to a reference dataset of patents in the domain of pharmacology issued from the QUAREO project. [1]

3.1 Data Source

The data is a collection of patent documents related to pharmacology domain. The bibliographic citations in the patents are extracted from the Medline database [2]. The source data contains 6387 patents in XML format, grouped into 15 subclasses of the A61K class

1. http://www.quaero.org
2. http://www.ncbi.nlm.nih.gov/pubmed/

(medical preparation). 25887 citations have been extracted from 6387 patents. Hajlaoui et al. (2012). Then the Medline database is queried with extracted citations for related scientific articles. The querying gives 7501 articles with 90% recall. Each article is then labeled by the class code of the citing patent. The set of labeled articles represents the final document set on which the training is performed. The final document set is unbalanced, with smallest class containing 22 articles (A61K41 class) and largest class containing 2500 articles (A61K31 class). Inter class similarity computed using cosine correlation indicates that more than 70% of classes' couples have a similarity between 0.5 and 0.9. Thus the ability of any classification model to precisely detect the right class is curtailed. A common solution to deal with unbalance in dataset is undersampling majority classes and oversampling minority classes. However sampling that introduces redundancy in dataset does not improve the performance of novelty detection training process. Instead, we recommend ensemble creation approaches such as boosting and bagging based on principles of data resampling and reestimation. Such techniques give low standard deviation between unbalanced and overlapping class labels. In this context, the training data weighting is an attempt to reach an optimal set of classesPrudent and Ennaji (2005) over cluster partitioning in unsupervised learning and feature sampling in search space. [3]. Here, by definition uniform stratification on either balanced or unbalanced dataset affects the feature correlation metrics than the class probability estimation process. So that bootstrapping of train and test data may solve problems of filtering sensibility, stability, scalability, dimensionality etc but does not improve accuracy computation over the sampled correlations. To reduce program execution time, we recommend the pruning of irrelevant features by thresholding and ranking the term frequencies.

3.2 Data Representation

- The document set is converted to a bag of words modelSalton (1971) using the TreeTagger tool Schmid (1994) developed by the Institute for Computational Linguistics of the University of Stuttgart. This tool is both a lemmatizer and a tagger. A lemmatizer associates a lemma, or a syntactic root, to each word in the text and a tagger automatically annotates text with morpho-syntactic information. In our case, the documents are firstly lemmatized and the tagging process is performed on lemmatized items (in the case when a word is unknown to the lemmatizer, its original form is conserved). The punctuation signs and the numbers identified by the tagger are deleted. The feature selection according to grammatical categories allows identifying salient features for the documents classification according to document types or opinions.
- Every document is represented as a term vector filled with keyword frequencies. The description space has dimensionality 30000. The thresholded description space has 11471. The whole text collection is then represented as a (N+1)J matrix where J is number of articles in the collection in a N-dimensional space. Each line j of this matrix is an N-dimensional bag of words vector for article j, plus its class label. The Term Frequency-Inverse Document Frequency(TFIDF) weighting scheme Salton and Buckley (1988) gives a sparse matrix representation of the text collection. The representation matrix is stored in arff format [4]. Although we use the sparse matrix representation in our

3. http://weka.sourceforge.net/doc.dev/weka/filters/supervised/attribute/AttributeSelection.html

4. http://weka.wikispaces.com/ARFF

experiments, we observe that the computational overhead of using a dense matrix representation such as Augmented normalized frequency (ANTF) weighting scheme is not useful. We require relevant terms in the text collection such that the mean and standard deviation of term frequencies varies from minimum and maximum values in document set. By flattening the term counts, the current version of ANTF formatted input reduces variance in term frequencies between balanced and unbalanced class labels across document set. However the ANTF dataset is not more linearly separable as compared to TFIDF formatted input dataset.

– The data is stored on disk in sparse arff format. Due to high dimensionality, external libraries cannot be used in place of regular expressions and string manipulation to load the dataset. An input record is loaded into memory in sparse matrix format. Dictionary Of Keys memory format is used to initialize sparse matrices. Each input record is converted to Compressed Sparse Row format for every subsequent matrix multiplication operation. The intermediate files after training, profiling, thresholding, testing processes are stored on disk with Cpickle 'old binary format' as protocol for disk io. Instead of a database server process handling disk io, if Cpickle cannot handle the sparse dataset scale, we propose the usage of HDF5,pytables library [5], [6]. During cross validation, large and small size train and test datasets are stored in batch and incremental mode. To check memory profiling, we also propose the usage of specialized sparse libraries (such as pysparse, sage, petsc4py instead of standard scipy) for storing and loading sparse matrices.

3.3 Feature selection

In order to compare the results issued from our incremental learning approach which optimized classification results, we exploit an original feature selection process based on feature maximization. Although such an optimization would not be realistic in incremental condition, it provides useful information on the influence of noise on the learning of different methods

Feature maximization is a quality metric which favours features with maximum feature F-measure Lamirel (2011). *Feature F-measure* (FF) is the harmonic mean of *feature recall* (FR) and *feature precision* (FP) which in turn are defined as [7]

$$FR_c(f) = \frac{\sum\limits_{v \in c} W_v^f}{\sum\limits_{c' \in C} \sum\limits_{v \in c'} W_v^f}, \qquad FP_c(f) = \frac{\sum\limits_{v \in c} W_v^f}{\sum\limits_{f' \in F_c, v \in c} W_v^{f'}}$$

where W_x^f represents the weight of the feature f for element x (1 or 0 in the case of our application) and F_c designates the set of features associated with the data associated to class c.

5. http://www.hdfgroup.org/HDF5/

6. http://www.pytables.org/

7. Since *feature recall* is equivalent to the conditional probability $P(c|p)$ and *feature precision* is equivalent to the conditional probability $P(p|c)$, this former strategy can be classified as an expectation maximization approach with respect to the original definition given by Dempster and al. Dempster et al. (1977)

A feature is then said to be relevant for a given class iff its feature F-measure is higher for that class than the average feature F-measure FF_m computed for all the classes

$$FF_m = \frac{\sum_{c \in C} FF_c(f)}{|C|} \tag{14}$$

All the relevant features of the classes are merged to form the pruned description space. This process resulted in reducing the size of the description space from 11471 variables to 330. Our new feature selection technique is finally compared to information gain in our experiment.

3.4 Results

Accuracy						
ILoNDF	Rochio	J48	RandomForest	AdaBoostM1	DMNBtext	SMO
0.59	0.53	0.41	0.43	0.39	0.56	0.57

TAB. 1 – *Comparing linear accuracies on thresholded dataset*

Accuracy						
ILoNDF	Rochio	J48	RandomForest	AdaBoostM1	DMNBtext	SMO
0.63	0.55	0.73	0.71	0.35	0.79	0.64

TAB. 2 – *Comparing linear accuracies on pruned dataset (feature maximization)*

Accuracy						
ILoNDF	Rochio	J48	RandomForest	AdaBoostM1	DMNBtext	SMO
0.61	0.53	0.48	0.49	0.38	0.58	0.59

TAB. 3 – *Comparing linear accuracies on pruned dataset (information gain)*

– We compare the accuracy given in Table 1, 2, 3 of our algorithm with the following algorithms. Default parameters, listed in Table 4, are used when executing weka algorithms.
  – Classical Rochio's supervised k-medoids algorithm on sparse data Rocchio (1971)
  – Weka's Decision Tree algorithm weka.classifiers.trees.J48 Quinlan (1993)
  – Weka's Random Forest algorithm weka.classifiers.trees.RandomForest Breiman (2001)
  – Weka's Boosting algorithm weka.classifiers.meta.AdaBoostM1 Freund and Schapire (1996)

Parameters for Weka Algorithms	
weka.classifiers.trees.J48	-C 0.25 -M 2
weka.classifiers.trees.RandomForest	-I 10 -K 0 -S 1 -num-slots 1
weka.classifiers.meta.AdaBoostM1	-P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump
weka.classifiers.bayes.DMNBtext	-I 1 -B false
weka.classifiers.functions.SMO	-C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K PolyKernel -C 250007 -E 1.0"

TAB. 4 – *Weka Algorithms and Default Parameters*

  - Weka's Bayesian Network algorithm weka.classifiers.bayes.DMNBtext Su et al. (2008)
  - Weka's Support Vector Machine algorithm weka.classifiers.functions.SMO Schoelkopf et al. (1998), Keerthi et al. (2001)
- We observe the following from results :
  - ILoNDF provides the best performance on initial thresholded dataset, in which a high level of noise can be observed.
  - In a given execution trial, accuracy values vary according to stratification, number of samples in each class. Profiling, ranking normalization in ILoNDF is affected by imbalance in dataset for minority class labels.
  - Information gain is quite inefficient for feature selection in the context of our dataset.
  - Feature maximization provides conversely very significant performance enhancement for most methods.
  - In addition to learning linear correlations, NDF acts as a regularization operation reducing variance across features and subspaces.
  - Rochio has best performance with L1 norm, indicating linear correlations are learnt. ILoNDF outperforms Rochio in pruned dataset using feature maximization principle. This result indicates that the feature selection process based on feature maximization has overcome nonlinearities across features in preliminary experiments.
  - Decision tree (J48) has performance comparable to RandomForest in initial thresholded dataset, indicating need for further feature selection.
  - Bayesian belief network, DMNBText, has most accuracy and least variance on pruned dataset using feature maximization, indicating stability across stratification. Boosting(AdaBoost) has lowest accuracy. We conclude that, a combination of bayesian techniques like the ones provided by a Bayesian belief network and Bayesian based feature selection technique is comparable to updated ILoNDF for learning nonlinear correlations.
  - The computed accuracy of incremental NDF has variable confidence level of 5-10%. TrueNegatives are dominating TruePositives in each fold indicating nonlinearity and label skew in dataset. Therefore, we do not attempt arbitrary comparison of the dissemination thresholds formed in orthogonal search spaces during accuracy evaluation.

4 Conclusion

In this paper we show that incremental extension of the novelty detection technique proposed for information filtering can be fruitfully exploited in the framework of the supervised classification of highly multidimensional, ultra sparse and noisy textual data, with high similarity between classes. We have especially shown that this method can outperform state of the art incremental techniques when noise and class imbalance are an inherent factor of the input flow, and thus must be considered as working constraints of the classification problem.

In theory, novelty detector learning method is an incremental parameter free method that has the ability of generative learning by integrating information relating to the relative frequencies and co-occurrence dependencies of the features that are stable in training data. Thus ILoNDF injects novelty into classification based on information in class labels. However, in sparse dataset, the accuracy variation which is observed may be studied by checking for the variation in diagonal of the novelty detectors that learn from a pruned featureset.

The performance enhancement we obtained with the use of variable selection based on feature maximization leads us also to think that the approach can thus be fruitfully combined with incremental clustering technique based on same kind of metric, like the IGNGF clustering method Lamirel (2011). The class profiles can thus be adaptively aggregated by defining cascading, voting and weighting experiments on filter profiles while constructing clusters from input samples. Hence, the use of the correlation matrices and discriminatory functions associated to cluster overlap, extension and generalization, one class novelty detection of ILoNDF can be integrated with multiclass clustering to give a ensemble classification method Duin (2002), Kuncheva (2002), Serpico et al. (1996), Giacintoa et al. (2000).

Further, we may define the loss functions and performance metrics based on relevancy feedback for smoothing the nonlinearities encountered when training the filters. We believe that such a statistical smoothing will also reduces the class imbalance in the dataset.

Finally, to build up novelty detectors over subspaces, we can also investigate the usage of random subspacing principle in the classification processDeerwester et al. (1990), Boley (1998), Zhao et al. (2011).

References

Boley, D. L. (1998). Principal direction divisive partitioning. *Data Mining and Knowledge Discovery 2(4)*, 325–344.

Breiman, L. (October 2001). Random forests. *Machine Learning 45(1)*, 5–32.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. Landauer, and R. Harshman (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 391–407.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood for incomplete data via the EM algorithm. *J. Roy. Statist. Soc. Ser. B 39*(1), 1–38.

Duin, R. (2002). The combining classifier: to train or not to train? *Proceedings. 16th International Conference on Pattern Recognition 2*, 765–770.

Dumais, S., J. Platt, D. Heckerman, and M. Sahami (1998). Inductive learning algorithms and representations for text categorization. *Proceedings of the Seventh International Conference on Information and Knowledge Management CIKM*, 148–155.

Freund, Y. and R. Schapire (1996). Experiments with a new boosting algorithm. *Proc International Conference on Machine Learning*, 148–156.

Giacintoa, G., F. Rolia, and F. Bruzzoneb (May 2000). Combination of neural and statistical algorithms for supervised classification of remote-sensing images. *Pattern Recognition Letters 21, Issue 5*, 385–397.

Hajlaoui, K., P. Cuxac, J. Lamirel, and C. Francois (2012). Enhancing patent expertise through automatic matching with scientific papers. *Discovery Science, Lecture Notes in Computer Science 7569*, 299–312.

Kassab, R. and J. Lamirel (2007). Towards a synthetic analysis of user's information need for more effective personalized filtering services. *Proceedings of the 2007 ACM symposium on Applied computing*, 852–859.

Keerthi, S., S. Shevade, C. Bhattacharyya, and K. Murthy (2001). Improvements to platt's smo algorithm for svm classifier design. *Neural Computation 13(3)*, 637–649.

Kohonen, T. (1989). *Self organisation and associative memory* (3 ed.). New York: Springer.

Kohonen, T. and E. Oja (Jan 1976). Fast adaptive formation of orthogonalizing filters and associative memory in recurrent networks of neuron-like elements. *Biol Cybern. 8;21(2)*, 85–95.

Kuncheva, L. (Feb 2002). A theoretical study on six classifier fusion strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24, Issue: 2*, 281–286.

Lamirel, J. (January 21, 2011). A new multi-viewpoint and multi-level clustering paradigm for efficient data mining tasks. *New Fundamental Technologies in Data Mining", Edited by Kimito Funatsu*.

Lamirel, J.-C. and M. Créhange (1994). *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*.

Markou, M. and S. Singh (2003). Novelty detection: a review|part 2: neural network based approaches. *Signal Processing journal 83*, 2499–2521.

Prudent, Y. and A. Ennaji (2004). Clustering incremental pour un apprentissage distribue - vers un systeme evolutif et robuste. *CAP04, Montpellier*.

Prudent, Y. and A. Ennaji (August 2005). A k nearest classifier design. *Engineering of Intelligent Systems, Electronic Letters on Computer Vision and Image Analysis, Editors: Jean-Marc Ogier, Thierry Paquet, Gemma Sanchez, Special issue on Document Analysis 5(2)*.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Raskutti, B. and A. Kowalczyk (2004). Extreme re-balancing for svms: a case study. *SIGKDD Explor. Newsl. 6(1)*, 60–69.

Rocchio, J. (1971). *Relevance feedback in information retrieval In The SMART Retrieval System: Experiments in Automatic Document Processing*. Englewood NewJersey: Prentice Hall Inc.

Salton, G. (1971). *Automatic processing of foreign language documents*. Prentice-Hill: Englewood, Cliffs, NJ.

Salton, G. and C. Buckley (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management 24(5)*, 513–523.

Schapire, R., Y. Singer, and A. Singhal (1998). Boosting and rocchio applied to text filtering. *Proceedings of the Twenty first Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 215–223.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. *Proceedings of International Conference on New Methods in Language Processing*.

Schoelkopf, B., C. Burges, and A. S. editors (1998). *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Serpico, S., L. Bruzzonea, and F. Rolib (25 November 1996). An experimental comparison of neural and statistical non-parametric algorithms for supervised classification of remote-sensing images. *Pattern Recognition Letters, Special Issue on Non-conventional Pattern Analysis in Remote Sensing 17, Issue 13*, 1331–1341.

Shankar, S. and K. George (2000). Weight adjustment schemes for a centroid based classier. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, Minnesota,. Computer Science Technical Report (TR00-035).

Su, J., H. Zhang, C. Ling, and S. Matwin (2008). Discriminative parameter learning for bayesian networks. *ICML*.

T., A. and Y. Yang (2001). knn, rocchio and metrics for information filtering. *The Tenth Text REtrieval Conference (TREC 10)*.

Zhao, Z., L. Wang, H. Liu, and J. Ye (2011). On similarity preserving feature selection. *IEEE Transactions on Knowledge and Data Engineering*.

Résumé

Cet article présente une approche originale pour synthétiser à la fois les caractéristiques de classes et la nouveauté à partir du même processus d'ap-prentissage incrémental supervisé. L'approche repose sur le paradigme de détection de nouveauté qui permet d'opérer une discrimination binaire des informations caractérisées par la même étiquette dans les données. Ce paradigme permet d'obtenir des profils de classes induits par des matrices de projection. Les matrices de projection, obtenues de manière incrémentale sont en mesure de caractériser les corrélations linéaires entre les variables représentatives des classes, sans nécessiter de processus complexe d'estimation de paramètres. Les premières études que nous menons dans ce papier montre que même dans le cas de données statiques la méthode incrémentale proposée peut surpasser les méthodes de référence, pour la classification de données déséquilibrées, bruitées, fortement multidimensionnelles et éparses avec un fort degré de similitude entre les classes. L'ensemble de données expérimentales est un ensemble de 7000 publications rattachées à des classes de brevets issues d'une classification de référence en pharmacologie.

Growing Self-organizing Trees for Knowledge Discovery from Data

Nhat-Quang Doan*, Hanane Azzag*, Mustapha Lebbah *

* Université Paris 13, LIPN UMR 7030
99, avenue Jean-Baptiste Clément
93430 Villetaneuse, France
nhat-quang.doan, hanane.azzag, lebbah@lipn.univ-paris13.fr

Résumé. In this paper, we propose a new unsupervised learning method based on growing neural gas and using self-assembly rules to build hierarchical structures. Our method named Growing Self-organizing Trees (GSoT) depicts data in topological and hierarchical organization. This makes GSoT a good tool for data clustering and knowledge discovery. Experiments conducted on real data sets demonstrate the good performance of GSoT.

1 Introduction

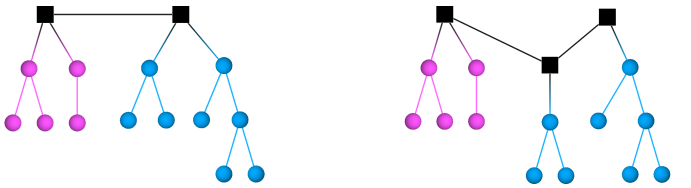
Discovering the inherent structure and its uses in large data has become one of the major challenges in data mining applications. An attractive way to assist the analysts in data exploration is to base on unsupervised approaches allowing clustering and mapping high-dimensional data in a small number of dimensions. The self-organizing map (SOM) can be used as a clustering method that addresses these issues. A variety of self-organizing models are derived from the first original model proposed by Kohonen [1]. All models are different from each other, but share the same idea : depict data set on a fixed and simple geometric relationship projected on a reduced and fixed topology (1D or 2D). Other variants such as Growing Neural Gas (GNG) by Fritzke [2] and growing hierarchical self-organizing map [3] allow to overcome sensitivity to topology by dynamically growing the grid or the network. Generally, growing algorithm is widely used for learning topological preservation, clustering, vector quantization and quick indexation of data [4, 5].

Hierarchical structures are often used to illustrate the arrangement of data. A hierarchical tree is an efficient and optimal representation facilitating cluster analysis. The identification of these hierarchical relations are an important data mining and exploration task that cannot be addressed conveniently within the growing neural gas or SOM models. Some approaches are presented in the related work section.

In this paper we propose a novel algorithm called GSoT : Growing Self-organizing Trees. The algorithm is based on the Growing Neural Gas algorithm [5] and an original autonomous hierarchical clustering method named AntTree [6]. One of the main contributions of GSoT is that the output space consists of a set of trees (hierarchical trees) arranged according to certain topology (called network) usually in one or two dimensional space (Fig. 1). Additionally, in our approach the number of trees are incremental and the tree topology is evolutive. The

particularity of these trees with respect to other existing approaches is that each node of tree represents one data object. This allows for an immediate comparison between data and visualization. It can be exploited by descending from topological level (network) to the last level of trees, which provides useful information about the clustering and the data structure and topology.

The key idea of GSoT algorithm is to successively add new trees to an initially small network by evaluating local statistical measures gathering during the previous adaptation steps. In GSoT, during one epoch, we generate a number of trees corresponding to the network size. To build trees, we have developed self-assembly rules inspired from AntTree [6]. AntTree is a hierarchical clustering method using artificial ants that are totally adaptive to self-organizing models. Data (ants) move locally toward tree structure without a centralized control policy and without using parameters. In classical hierarchical clustering methods, an object is assigned to a cluster and will not be considered again. A correction of previous misclassification is not possible. On the other hand, AntTree rules overcome this limitation by allowing a self-adaptation of structure of tree topology during training.



(a) GSoT structure : two trees linked by a topological connection

(b) New GSoT structure adapting to the insertion of new neuron

FIG. 1 – *The GSoT architecture in two successive epochs of growing process. Each tree node represents one data object.*

TAB. 1 – *Different aspects of hierarchical approach*

Algorithm	**Hierarchy of**	**Output trees**	**Node type**	**Data per Node**
GH-SOM [7]	prototype	single	vector	single
SOM-AT [8]	data	single	attribute	single
TreeSOM [9]	prototype / data	single	vector	variant
S-TREE [10]	prototype	single	vector	single
SOTree [11]	data	single	vector	single
TS-SOM [12]	prototype	single	vector	single
HOGNG [13]	prototype	single	vector	variant
TreeGNG [14]	layer	single	vector	variant
GSoT	data	multiple	vector	single

In general, data structure of GSoT is both topological and hierarchical as the example de-

picted in Fig. 1. In our representation, black square node refers to support (network node) ; circle node refers to tree node, which corresponds to an input data object. Two trees are topological neighbors if an edge is created between their supports. First, the network is initialized with just two supports (Fig. 1(a)). These supports are regarded as the tree roots. Having completed connecting data to their best match tree, we add new support to extend the network size. The next step is to compute new assignments for input data. Once data have found their new best match tree again, we disconnect and reconnect them (with their associated sub-tree) from the old to the new tree. Incrementally, we obtain several hierarchical trees while the training step still proceeds (Fig. 1(b)).

The remainder of this paper is organized as follows : we briefly describe the novelty features over existing methods in Section 2. Section 3 is devoted to our model. We present the experiments with numerical and visual results in Section 4. In the final Section 5, we draw our conclusions and perspectives.

2 GSoT versus other hierarchical methods

Self-Organizing models are very efficient approaches to solve clustering problems. Two well-known models are SOM [1] and GNG which share the same principle of topological preservation and vector quantization. There are tree-structured variations to the GNG algorithm such as HOGNG [13] or TreeGNG [14]. Because of the similar architecture, we are also interested in studying hierarchical SOM variants such as GH-SOM [7], SOM-AT [8], S-TREE [10], SOTree [11], TreeSOM [9] and TS-SOM [12]. An overview on various features of these methods and GSoT is summarized in Table 1. Here we want to differ our method from the others in several principal features such as the basic hierarchical structure, the number of output trees, the type of node and the number of data per node.

GNG and SOM have some common features, but the difference is the network size. While the size of SOM must be defined a priori, the size of GNG may vary during training process. In general, the methods in Table 1 are different from each other in one or several features. While GSoT propose simultaneously to build many trees, the others have construct only a single tree. GH-SOM, TS-SOM approaches proposed a structure where each prototype becomes one node of tree. Having the same idea, S-TREE grows a binary tree from prototype vectors. SOTree provides a 2D grid modeled by a hierarchical tree to visualize data. TreeSOM generates a hierarchical tree where only the leaf nodes may get many data elements, and other nodes none at all. SOM-AT based on introducing matching and adjusting schemes for input data attribute trees. The most optimal tree is selected to represent input data. TreeGNG tries to build a tree whose nodes contain a group of data. These data groups are determined by GNG algorithm. HOGNG aims to generate two hierarchical layers where the second one is expanded for overlapped data.

Using tree-structured representation provides remarkably more degrees of freedom and advantage to analyze input data. GSoT facilitates visualizing clusters as multi-hierarchical trees topologically connected to each other. GSoT trees describe relations between pairs of data. In fact, the structure of other hierarchical clustering methods are usually depicted by a binary tree or dendrogram. In such types of tree-like structure, only leaf nodes are regarded as data objects. The other nodes, thus, describe the proximity of objects. Once an object is assigned to a cluster, it will not be considered again. When positions of data in structure are fixed, those hie-

rarchical clustering methods are not capable of correcting possible previous misclassification. However, this problem can not be seen in our method due to the possibility of connecting or disconnecting data from the rules we have defined. Hence, this is very easy to adapt to growing networks.

3 Growing Self-organizing Tree

3.1 GSoT principle

Growing algorithm is an adaptive process in the sense that the network moves the cells to cover the data distribution. In other words, the cells follow the probability density of input data. Statistical information is used for determining appropriate place where to insert a new cell in the network [2]. Due to the nature, the network size is incremental over time, the evolutive topology of network depends on input data. The position of reference vector may vary on the random selection for training. A connection has an age variable used to decide when to remove old edges in order to keep the topology updated.

GSoT is autonomous method since we have very limited parameters. We do not have to fix the number of observations used in assignment step as the traditional GNG does. Our method progressively constructs tree-like organization over time. The only parameters needed are stopping criteria as quantization error threshold. The algorithm can aslo terminate as maximum number of trees have been reached.

3.2 GSoT batch algorithm

Algorithm 1 GSoT batch algorithm

1. intialize two supports at random positions in $\Re^d$ as well as the two respective trees.
2. initialize the data list $List$.
3. select randomly v_i from $List$.
4. $\mathbf{assignData}$: find the nearest support $s_1 \in S$
5. – if v_i is initial, $tree_{s_1} \leftarrow$ **buildTree**$(tree_{s_1}, v_i)$
   – if v_i is disconnected, $tree_{s_1} \leftarrow$ **buildTree**$(tree_{s_1}, subtree_{v_i})$
   – if v_i is connected and $s_1 \neq s_{old}$, $subtree_{v_i} \leftarrow$ disconnect v_i and all the data recursively connected to v_i from $tree_{s_{old}}$; $tree_{s_1} \leftarrow$ **buildTree**$(tree_{s_1}, subtree_{v_i})$
6. v_i is not connected, put v_i onto $List$; otherwise, remove $subtree_{v_i}$ from $List$.
7. $S \leftarrow$ **updateSupport**(S) : update supports.
8. if $List$ is not empty, go to step 3 ;
   else $S \leftarrow$ **addSupport**(S) : add new support.
9. if the stopping criterion are not yet fulfilled go to step 2.

Let $\Re^d$ be the Euclidean data space and $\mathcal{X} = \{\mathbf{x}_i; i = 1, \ldots, N\}$ a set of observations, where each observation $\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^d)$ is a vector in $\Re^d$. In our approach, each node v_i of the tree is associated with observation $\mathbf{x}_i$. The network of trees we consider consist of :

- A set S of supports (or cells). Supports don't contain any information of input data. Each support $s_i \in S$ is the root of the corresponding tree denoted $tree_{s_i}$. A tree is associated to a reference vector (prototype) $\mathbf{w}_{s_i} \in \Re^d$.
- A set E of connections among pairs of supports. These connections preserve the topological relations in the network. Connections are characterized by a variable called "age".

A node v_i (i = [1,...,N]) has only one among three status at a time :

- *initial* : the default status before training,
- *connected* : the node is currently connected to another node,
- *disconnected* : the node were connected at least once but now gets disconnected.

We denote by a set $List$ which contains all nodes. Before training, $List$ contains only *initial* nodes. Whenever a node becomes *connected*, it is immediately removed from $List$; alternatively whenever a node and its children get disconnected from a tree, we put them back onto $List$. If $List$ becomes empty, we terminate the current training epoch. We refresh $List$ to train data again if the stopping criteria haven't been met yet. Let us denote

- v_{pos} is the support (the root of tree) or the node position where v_i is located. At the beginning, v_i is located on the support and will move in the tree (toward another nodes) in order to find the best position.
- v^+ and v^- two nodes connected to v_{pos} which are respectively the most similar and dissimilar node to v_i.
- $subtree_{v_i}$ consists of the root v_i and all the nodes recursively connected to v_i.

The batch algorithm is shown in Algorithm 1. In order to stop the algorithm autonomously, we can employ a threshold of quality measure such as quantization error. Or simply, we just consider a number of trees as stopping criterion. Four important steps are presented as following.

Assignment step

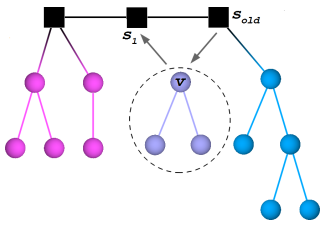


FIG. 2 – *Group assignment*

This step corresponds to the function $assignData$ used in Algorithm 1. There are two distinct types of assignment :

1. a single data object.

$$\phi(v_i) = argmin \parallel \mathbf{x}_i - \mathbf{w}_{s_1} \parallel^2$$

2. a subtree, $subtree_{v_i}$ and $\phi(subtree_{v_i}) = \phi(v_j) = \phi(v_i)\ \forall v_j \in subtree_{v_i}$.

For each training iteration, we determine the winner support s_1 such that $\mathbf{w}_{s_1}$ is the most similar (using Euclidean distance) with the input sample $\mathbf{x}_i$. Node v_i associated with $\mathbf{x}_i$, is assigned and connected to the respective tree (i.e $tree_{s_1}$). If v_i has *initial* status or does not have any node connected to it, the assignment for single node v_i is simply succeeded. Otherwise, for $subtree_{v_i}$, we repeat the same process for v_i as in the previous case. Then all node $v_j \in subtree_{v_i}$ is going to automatically follow v_i and gets assigned to $tree_{s_1}$.

An example of sub-tree assignment is shown in Figure 2. Given that $subtree_v$ constituted by three silver data is no longer connected to $tree_{s_{old}}$. Now we have to determine the new best match support s_1 (or $tree_{s_1}$) for the root of $subtree_v$ (i.e the node v). The assignment of its two child nodes follows the one of v using the statistical characteristics of tree. Even though these three nodes are now found in s_1, their hierarchy in new support always remains as the one in the old one (i.e. in s_{old}).

Tree construction

This step can be essentially realized by the function *buildTree*. The connecting and disconnecting rules are stated as following : In Algorithm 2, $T_{Dissim}(v_{pos})$ is denoted as the lowest

Algorithm 2 buildTree : rules of building tree

R 1. less than 2 data connected to v_{pos}
connect v_i to v_{pos}

R 2. more than 2 data connected to v_{pos} and for the first time
$T_{Dissim}(v_{pos}) = min(sim(v_i, v_j))$ where v_i and v_j are any pair of nodes connected to v_{pos} and $sim(v_i, v_j) = ||\mathbf{x}_i - \mathbf{x}_j||^2$
If $sim(v_i, v^+) < T_{Dissim}(v_{pos})$, disconnect v^- from v_{pos} (and recursively all child nodes of v^-) and connect v_i to v_{pos}
Else move v_i to v^+

R 3. more than 2 data connected to v_{pos} and for the sencond time
If $sim(v_i, v^+) < T_{Dissim}(v_{pos})$, connect v_i to v_{pos}
Else move v_i to v^+

similarity value which can be observed among the children of v_{pos}. v_i is connected to v_{pos} if and only if the connection of v_i decreases further this value. Since this minimum value can only be computed with at least two nodes, then the first two data are automatically connected without any test in Rule 1. This may result in "abusive" connection for the second data. Therefore the second data is removed and disconnected as soon as the third one is connected (Rule 2). For this latter one, we are certain that the similarity test has been successful. We allow to disconnect data only once from v_{pos} during training to assume the convergence of the algorithm. If we have already disconnected data from v_{pos}, the Rule 3 is employed. We remind that each time data are connected, they are immediately removed from $List$; otherwise if they are disconnected, they are put back onto $List$.

During the training, the disconnected data are put back into $List$. These data will get another assignment so that their misclassification can be corrected. In practice, there are three distinct cases of disconnection :

1. disconnect data due to an assignment.
2. disconnect data when we try to connect data in Rule 2.
3. disconnect a tree due to that the corresponding support is removed during update step.

(a) Disconnection of sub-tree (b) Reconnection of sub-tree

FIG. 3 – *Disconnecting/reconnecting sub-tree*

A simple example of disconnection/reconnection for a sub-tree of data is illustrated in Figure 3. Given $tree_{s_2}$ as in Figure 3(a), v is to disconnect from $tree_{s_2}$. All the node connected to v must be recursively disconnected too. Therefore $subtree_v$ (consists of three silver data) have *disconnected* status and are put back onto $List$. After getting new assignment, v is going to connect to $tree_{s_1}$. It leads to that the children of v (if they are still disconnected) have s_1 as their best match support too. We systematically connect this sub-tree to s_1 following the call of *buildTree* function. Eventually, the network is updated and shown in Figure 3(b). While reconnecting a sub-tree, we always want to keep its structure. This gives a benefit in time execution.

Updating supports

The function $updateSupport$ is devoted for this step which is necessary to compute new positions for supports. Topological neighbors are taken into account in order to update supports. Here in Algorithm 3 we follow the same process as in GNG.

Algorithm 3 updateSupport : updating the supports

1. add the squared distance between the nearest unit $\mathbf{w}_{s1}$ and the input data $\mathbf{x}_i$ to the local error :
$$error_{s1} = error_{s_1} + \parallel \mathbf{w}_{s_1} - \mathbf{x}_i \parallel^2$$
2. move s_1 and its direct topological neighbors (i.e. all nodes connected to s_1 by an edge) toward v_i by fractions ϵ_b and ϵ_r, respectively, of the total distance :
$$\Delta_{s1} = \epsilon_b(\mathbf{x}_i - \mathbf{w}_{s_1})$$
$$\Delta_r = \epsilon_r(\mathbf{x}_i - \mathbf{w}_r)$$
for all direct neighbors r of s_1
3. find the second nearest support s_2 of v_i. If $tree_{s_1}$ and $tree_{s_2}$ are connected by an edge, set the age of this edge to zero. Otherwise, create new edge between them.
4. remove edges with an age larger than Max_{age}. If this results in no emanating edges, remove them as well.
5. decrease all errors by multiplying them with a constant β.
6. repeat the step 1 for data which followed a group assignment with v_i

Inserting new support

The network is flexible by adding new supports, which can be seen in Algorithm 4. It can be done if only we have successfully connected all the data from $List$ to trees. When a new support is inserted into the network, we have to define the tree associated with this support. Once the tree has been initialized, $List$ must be refreshed in order to redo the assignments again. This explains why GSoT is able to avoid misclassification of previous step.

Algorithm 4 addSupport : adding new support into the network

1. find the support q with the maximum accumulated error.
$$q = argmax(error_{s_i}) \; \forall s_i \in S$$
2. insert a new support t halfway between q and its neighbor f with the largest error :
$$\mathbf{w}_p = \frac{1}{2}(\mathbf{w}_q - \mathbf{w}_f)$$
3. insert edges connecting p with q and f, and remove the edge between q and f.
4. decrease the local errors of q and f by multiplying them with a constant α, and initialize the error value of p with the new value of the error of q.

4 Experiments

4.1 Set-up parameters and quality criterion

Concerning numerical results, we're interested in comparing our model with algorithms that possess similar architecture such as GNG and MST (Minimum Spanning Tree) [15]. In this case we adopt the same parameters for GSoT and GNG : a maximum number of trees (max = 10 or 30) as stopping criterion, a maximum of age (max = 50), random initialization and scaling factors for the reduction of error of nodes ($\alpha = 5.10^{-2}$, $\beta = 5.10^{-3}$, $\epsilon_b = 0.1$, $\epsilon_r = 5.10^{-4}$). For MST, we build the proximity graph using Prim's algorithm [16]. Then the nine longest edges are removed from this proximity graph in order to obtain ten separated clusters. Unlike others, MST builds one invariable tree which resulted in the obtained clusters to be independent from the random initialization.

In order to evaluate the performance, we select three different criteria : Accuracy, Normalized Mutual Information [17] and Rand index ; each should be maximized. Given a set of N objects of L classes classified into K clusters and two partitions to compare : $X = \{x_1, .., x_N\}$ where $x_k \in [C_1..C_K]$ a random variable for cluster assignments, $Y = \{y_1, .., y_N\}$ where $y_l = [B_1..B_L]$ a random variable for the pre-existing labels. Hence, the contingency table can be expressed as in Table 2 :

TAB. 2 – *Contingency table*

B\C	C_1	C_2	$\cdots$	C_k	$\cdots$	C_K	Sum
B_1	n_{11}	n_{12}	$\cdots$	n_{1k}	$\cdots$	n_{1K}	N_{1*}
B_2	n_{21}	n_{22}	$\cdots$	n_{2k}	$\cdots$	n_{2K}	N_{2*}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
B_l	n_{l1}	n_{l2}	$\cdots$	n_{lk}	$\cdots$	n_{lK}	N_{l*}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
B_L	n_{L1}	n_{L2}	$\cdots$	n_{Lk}	$\cdots$	n_{LK}	N_{L*}
Sum	N_{*1}	N_{*2}		N_{*k}		N_{*K}	N

The Accuracy (Purity) is defined as follows :

$$Acc = \frac{\sum_k max_{l=[1..L]}(n_{lk})}{N} \tag{1}$$

The Normalized Mutual Information is given by the following expression :

$$NMI = \frac{\sum_l \sum_k n_{l_k} log_2(\frac{N.n_{l,k}}{N_{l*}N_{*k}})}{(\sum_l N_{l*} log_2(\frac{N_{l*}}{N}))(\sum_k N_{*k} log_2(\frac{N_{*k}}{N})}) \tag{2}$$

The Rand Index is defined straightforwardly as

$$Rand = (N_{00} + N_{11}) / \left(\begin{array}{c} N \\ 2 \end{array} \right) \tag{3}$$

where N_{11} is the number of pairs that are in the same cluster in both B and C ; N_{00} is the number of pairs that are in different clusters in both B and C.

4.2 Competitive performance

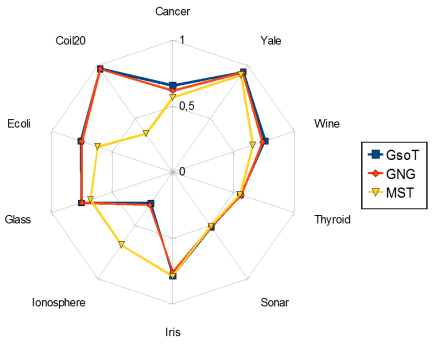
TAB. 3 – *Competitive performance*

Data-set	Method	Accuracy	NMI	Rand
Cancer	GSoT	0.962	0.479	0.656
	GNG	0.957	0.484	0.618
	MST	0.691	0.097	0.567
Coil20	GSoT	0.809	0.803	0.969
	GNG	0.782	0.805	0.968
	MST	0.181	0.353	0.360
Ecoli	GSoT	0.856	0.540	0.761
	GNG	0.848	0.542	0.753
	MST	0.639	0.363	0.622
Glass	GSoT	0.703	0.414	0.754
	GNG	0.700	0.410	0.750
	MST	0.672	0.374	0.681
Ionosphere	GSoT	0.892	0.075	0.291
	GNG	0.883	0.081	0.309
	MST	0.894	0.046	0.684
Iris	GSoT	0.954	0.641	0.789
	GNG	0.918	0.634	0.759
	MST	0.840	0.559	0.780
Sonar	GSoT	0.676	0.094	0.514
	GNG	0.673	0.096	0.513
	MST	0.634	0.148	0.508
Thyroid	GSoT	0.930	0.462	0.558
	GNG	0.907	0.458	0.566
	MST	0.772	0.241	0.553
Wine	GSoT	0.934	0.594	0.761
	GNG	0.910	0.559	0.740
	MST	0.634	0.453	0.666
Yale	GSoT	0.624	0.648	0.939
	GNG	0.593	0.613	0.933
	MST	0.600	0.612	0.911

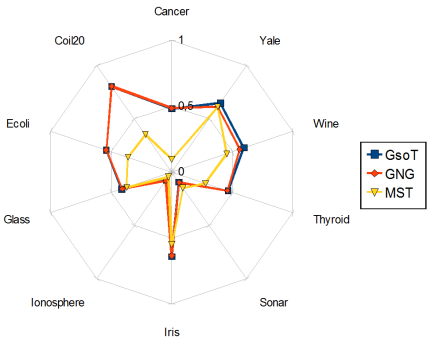
To show experimentally the efficiency of GSoT on data with known properties, we have applied three algorithms to real world databases. The real databases are available in the Machine Learning repository. Additionally, we selected two image databases : Coil20 [18] consists of a set which contains images of 20 different objects with 72 images per object ; Yale [19] contains 165 grayscale images of 15 individuals. Each data-set has been normalized before training.

All results which are presented in Table 3 and the radar charts in Fig. 4 have been averaged over 10 runs. We can observe the results given by different methods for 10 clusters and 30

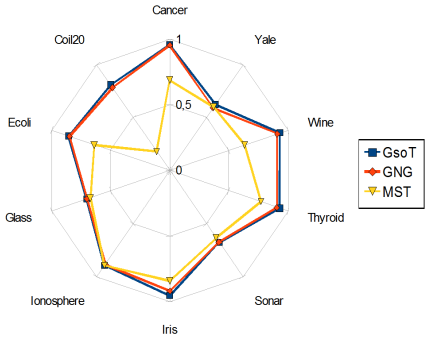
clusters for Coil20 and Yale. In practice GSoT manages to output better values of quality measures than GNG in most of selected databases. MST is always less efficient than the two others. Because of the poor performance, MST should be avoided. The most significant cases where GSoT dominates all three measures are Cancer, Iris, Wine and Yale data-set. Particularly, for Coil20, our method provides better Accuracy than the others (i.e. Acc = 0.809 vs 0.782 and 0.181). For Ecoli we have better values in Accuracy (Acc = 0.856) and Rand index (Rand = 0.761). Finally for Yale, even though GSoT and GNG are almost equal to Rand index, our algorithm obtains higher values of Accuracy (Acc = 0.624) and NMI (NMI = 0.648).



(a) Rand index



(b) Normalized mutual information



(c) Purity

FIG. 4 – *Quality criterion of each database*

4.3 Discussions : Number of assignments

In fact, the trees of GSoT are considered as an intermediary step to memorize all the assignments of the previous epoch. Fig. 5 displays the percentage of assignments of every database during training. The x-axis refers to the number of epochs and the y-axis the ratio of the number of assignments over the number of data. The whole data-set must be trained by the GNG algorithm (i.e $Ratio = 100\%$) but the GSoT algorithm uses only a small proportion of data

to train in each epoch. At the beginning, the number of assignments start from 100%. Then, the number of new assignments should be decreased epoch after epoch because only a small ratio of data must be re-assigned to new tree. Most of selected databases are in this case, the ratio goes down to 40% after 29 first epochs. However, it should be noted that data can get disconnected and assigned to another best match tree several times due to autonomous rules of connections/disconnections and the update of supports in an epoch. It explains that several curves go up sometimes such as in Cancer, Coil20 and Thyroid.

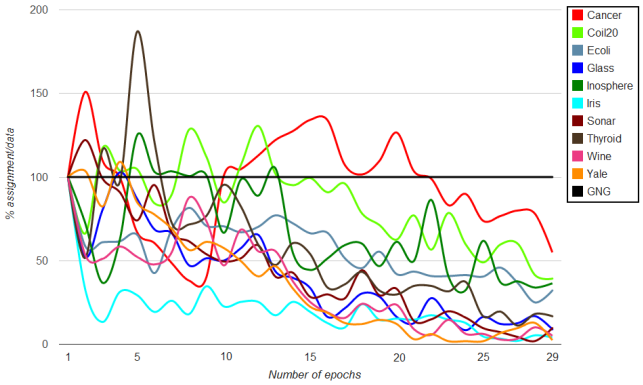


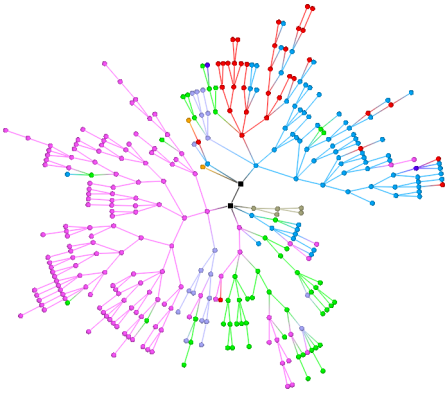
FIG. 5 – *Percentage of new assignments during training process for all chosen databases*

4.4 Visual validation

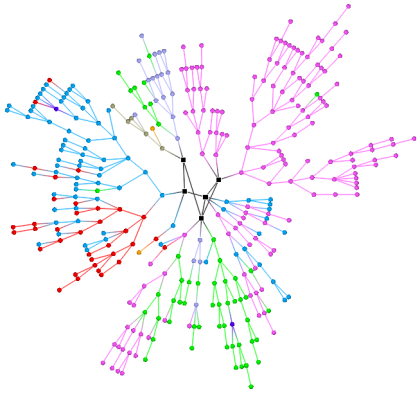
This experimental phase shows how the proposed method provides further information than other clustering approaches. The main advantage is to provide a simultaneous organization : topological and hierarchical. This simplifies data exploration by offering friendly and interactive visualization. We use Tulip [20] as the framework to visualize and analysis network. Fig. 6 and 7 respectively show the evolution of Ecoli and Coil20 data structure provided by GSoT in successive epochs. We arrange data into a new organization which is more visible and easier to visualize and analyze. The supports whose appearance are black and square are located in the center surrounded by trees. We have a tree associated with one support and the tree root is attached to this node. We provide the multi-level structure given by GSoT when the network size is [2, 5, 10] for Ecoli and [2, 10, 20] for Coil20. For each time, a view displays unique colors for different input classes.

In Fig. 6, we can certainly observe the clusters given by GSoT, most of them are presented by major tree. For Ecoli database, we have 336 data grouped into eight classes. In this data-set, there are two classes with only two data (purple and orange, these data are often misclassified with a majority vote. When network has two or five neurons (Fig. 6(a) and 6(b)), we are not able to detect all the input classes. When the size of network increases and is superior than the number of input classes, we can remark some big clusters in Fig. 6(c).

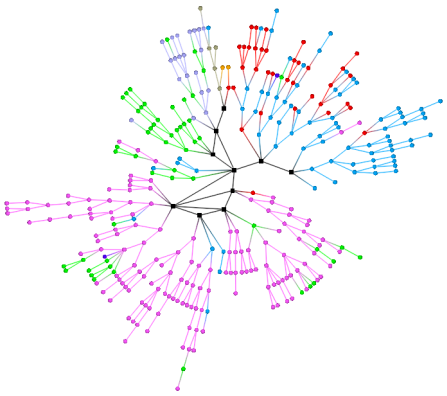
The visual result of Coil20 agrees with the previous one, we always found large trees in Fig. 7. Several pure clusters such as celeste and maroon classes can be clearly seen in Fig. 7(c). Because a vector of Coil20 describes an image, we zoom into two samples extracted from Fig.



(a) 2-tree network (Rand = 0.674 ; NMI = 0.421 ; Purity = 0.616)



(b) 5-tree network (Rand = 0.821 ; NMI = 0.550 ; Purity = 0.726)

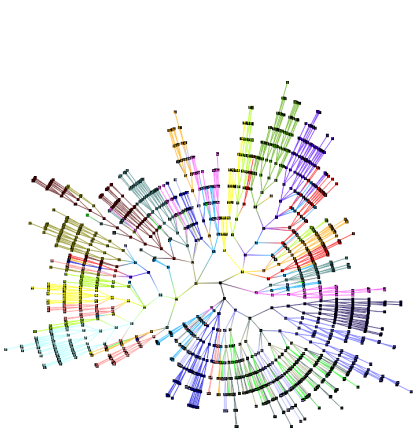


(c) 10-tree network (Rand = 0.869 ; NMI = 0.637 ; Purity = 0.794)

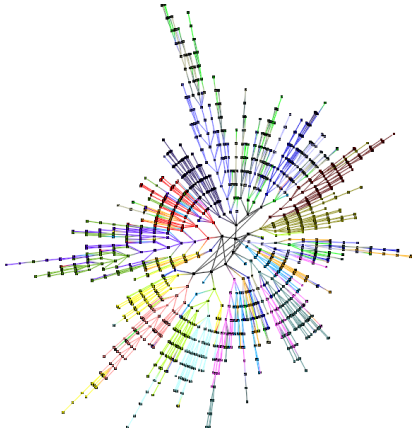
FIG. 6 – *Ecoli : data structure presented by the GSoT algorithm. Each color represents one real class of input data.*

7(c) in order study the similarity between images in hierarchical sub-trees. We just take only two first level of sub-tree shown in Fig. 8 : one from the celeste cluster (the top square in Fig. 7(c)) and another from a mixed cluster (the bottom square in Fig. 7(c)). The first zoom shows a good classification. The same type of objects (cup) are grouped together, GSoT well separated cup class where symbol is visible. For the second, even though we grouped images of different objects in the same cluster, the geometric shapes of objects are quite similar (car). In sub-treesm car objects turn in the same direction of their parent node.

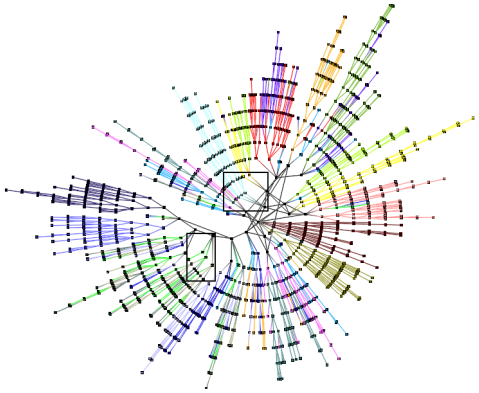
Our approach tries to improve the standard visualization by building topological and hierarchical ordered clusters. Atypical data are precisely pinpointed with this approach and can be further studied. Furthermore, we can directly visualize data in form of hierarchical and topological trees without projecting like PCA. In practice, analysts can apply our multi-structure



(a) 2-tree network (Rand = 0.685 ; NMI = 0.356 ; Purity = 0.146)



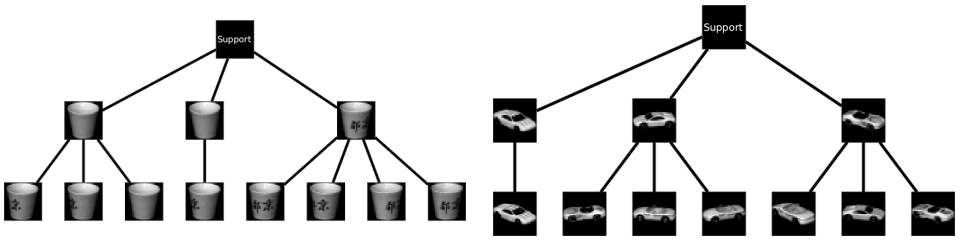
(b) 10-tree network (Rand = 0.895 ; NMI = 0.559 ; Purity = 0.368)



(c) 20-tree network (Rand = 0.945 ; NMI = 0.735 ; Purity = 0.612)

FIG. 7 – *Coil20 : data structure presented by the GSoT algorithm. Each color represents one real class of input data.*

to image indexation and exploration (search engine).



(a) First zoom sample : images belong to one class

(b) Second zoom sample : images belong to three different classes

FIG. 8 – *Zoom views of Coil20*

5 Conclusion

In this paper, we have proposed a topological and hierarchical model based on growing process and autonomous tree building technique. This model allows to generate hierarchical relations between pairs of data and topological relations between two trees. The proposed algorithm is able to appropriate clustering the data in all runs and brings advantage in reducing the number of training assignments. GSoT works well on several real world data-sets through the experiments. In addition, with the proposed structure, GSoT offers friendly and interactive visualization for input data.

As perspectives, there are a number of interesting potential avenues for future research in growing self-organizing trees. We could additionally incorporate some label into the clustering process. The objective is to optimize a semi-supervised clustering by using the constraints provided by trees structures. Another possibility would be to explore the features selections techniques by ranking features based on their spatial distribution and their trees position.

Références

[1] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[2] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[3] Dieter Merkl Michael Dittenbach, Andreas Rauber. Uncovering hierarchical structure in data using the growing hierarchical self-organizing map. *Neurocomputing*, 48(1-4) :199–216, October 2002.

[4] JosÃ© Alfredo F. Costa and Ricardo S. Oliveira. Cluster analysis using growing neural gas and graph partitioning. In *IJCNN*, pages 3051–3056. IEEE, 2007.

[5] Bernd Fritzke. Unsupervised clustering with growing cell structures. In *In Proceedings of the International Joint Conference on Neural Networks*, pages 531–536. IEEE, 1991.

[6] Hanene Azzag, Gilles Venturini, Antoine Oliver, and Christiane Guinot. A hierarchical ant based clustering algorithm and its use in three real-world applications. *European Journal of Operational Research*, 179(3) :906–922, June 2007.

[7] Michael Dittenbach, Dieter Merkl, and Andreas Rauber. The growing hierarchical self-organizing map. pages 15–19. IEEE Computer Society, 2000.

[8] Markus Peura. The self-organizing map of trees. *Neural Process. Lett.*, 8 :155–162, October 1998.

[9] Elena V. Samsonova, Joost N. Kok, and Ad P. Ijzerman. Treesom : Cluster analysis in the self-organizing map. neural networks. *American Economic Review*, 82 :1162–1176, 2006.

[10] Marcos M. Campos and Gail A. Carpenter. S-tree : self-organizing trees for data clustering and online vector quantization. *Neural Netw.*, 14 :505–525, May 2001.

[11] Hanene Azzag and Mustapha Lebbah. Self-organizing tree using artificial ants. *JITR*, 4(2) :1–16, 2011.

[12] Pasi Koikkalainen and Ismo Horppu. Handling missing data with the tree-structured self-organizing map. In *IJCNN*, pages 2289–2294, 2007.

[13] Xiang Cao and Ponnuthurai N. Suganthan. Video shot motion characterization based on hierarchical overlapped growing neural gas networks. *Multimedia Syst.*, 9(4) :378–385, 2003.

[14] Kevin Doherty, Rod Adams, and Neil Davey. TreeGNG - hierarchical topological clustering. In *ESANN*, pages 19–24, 2005.

[15] Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. Minimum spanning tree based clustering algorithms. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '06, pages 73–81, Washington, DC, USA, 2006. IEEE Computer Society.

[16] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36 :1389–1401, 1957.

[17] Alexander Strehl, Joydeep Ghosh, and Claire Cardie. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3 :583–617, 2002.

[18] Deng Cai, Chiyuan Zhang, and Xiaofei He. Unsupervised feature selection for multi-cluster data. In *16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'10)*, 2010.

[19] Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han, and Thomas Huang. Learning a spatially smooth subspace for face recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition Machine Learning (CVPR'07)*, 2007.

[20] D. Auber. Tulip : A huge graph visualisation framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Softwares*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

Summary

Dans cet article, nous présentons une nouvelle méthode d'apprentissage non supervisé basée sur le "Growing Neural Gas" et en utilisant des règles d'auto-assemblage pour construire des structures hiérarchiques. Notre méthode nomée Growing Self-organizing Trees (GSoT) représente les données avec une organization multi-niveaux: topologique et hiérarchique. Nous montrons dans les expérimentations sur des bases de données réelles que GSoT offre de nouveaux outils pour partitionner, explorer les données et extraire des connaissances.

Incremental mining of frequent sequences from a window sliding over a stream of itemsets

Thomas Guyet*,**, René Quiniou***

*AGROCAMPUS OUEST, UMR6074 IRISA, F-35042 Rennes
**Université européenne de Bretagne
***INRIA Centre de Rennes - Bretagne Atlantique

Abstract. Nous introduisons le problème de fouille des séquences fréquentes dans une fenêtre glissante sur un flux d'itemsets. Pour résoudre ce problème, nous présentons un algorithme incrémental, complet et correct, basé sur une représentation des séquences fréquentes inspiré par l'algorithme PSP et sur une méthode de comptage des occurrences minimales. Les expériences ont été menées sur des données simulées et sur des données réelles consommation instantanée d'électricité. Les résultats montrent que notre algorithme incrémental améliore de manière significative le temps de calcul par rapport à une approche non-incrémentale.

1 Introduction

Sequential pattern mining has been studied extensively in static contexts (Masseglia et al. (1998); Pei et al. (2004); Srikant and Agrawal (1996)). Most of the proposed algorithms make use of the Apriori antimonotonicity property stating that if a pattern is frequent then all its sub-patterns are also frequent. All these algorithms make use of a pattern counting method based on the number of transactions that contain the pattern without taking into account the multiple occurrences of the pattern in a transaction.

Counting the number of occurrences of a motif, or episode, in a window introduces some complexity compared to previous approaches as two occurrences of an episode can overlap. Mannila et al. (1997) introduced the algorithms Minepi and Winepi for extracting frequent episodes and counting their minimal occurrences. Since then, other counting methods have been proposed to solve this problem while preserving the antimonotonicity properties required for the effectiveness of pattern occurrences search (see Achar et al. (2010)).

In the context of data streams, where data arrive in a continuous flow, specific algorithmic solutions must be designed for the extraction of frequent sequences. A common practice consists in sliding a window over the data and then in extracting frequent episodes from the successive itemsets inside this window. But the itemset series is evolving continuously: when a new itemset arrives, the oldest itemset at the beginning of the window becomes obsolete. In existing static approaches, it is necessary to restart an entire mining process to extract the frequent episodes of the new period.

As the context of data streams imposes to process the incoming data very fast, the naïve approach above has a prohibitive computation time. Thus, an incremental method that updates

efficiently the current set of sequential patterns is needed. We propose a method based on a representation of frequent sequences inspired by PSP (Masseglia et al. (1998)), an improvement of GSP (Srikant and Agrawal (1996)) for mining frequent itemset subsequences from a database of sequences of itemsets. This representation enables an efficient update of sequential pattern occurrences due to new data and obsolete data as well.

In section 2, we present related work. In section 3, we present a formal setting of the problem of mining itemsets from a data stream. In section 4, we present an incremental algorithm that solves this problem. In section 5, we introduce the data structure used to collect the history of frequent sequences. In section 6, we present experimental results on simulated data as well as real data related to instantaneous power consumption.

2 Related work

Since the publication of the seminal paper of Agrawal et al. (1993) establishing the foundations of itemset mining, many proposals have been made. We are particularly interested in sequential pattern mining, with a special focus on incremental and progressive methods, such as those proposed in the field of data streams.

Many methods have been proposed to discover frequent subsequences or sequential patterns either from a sequence database or from a single long sequence, in a static context or in data streams. GSP (Srikant and Agrawal (1996)), PSP (Masseglia et al. (1998)), PrefixSpan (Pei et al. (2004)), Spade (Zaki (2001)), Spam (Ayres et al. (2002)), to cite a few, proposed different approaches to the problem of mining sequential patterns (*i.e.* frequent sequences of itemsets) from a static sequence (of transactions) database. There are less proposals for mining frequent recurring patterns from a single long sequence. Minepi, Winepi (Mannila et al. (1997)), WinMiner (Méger and Rigotti (2004)), for instance, count the minimal occurrences of episodes in a given sequence, while Laxman et al. Laxman et al. (2007) counts the non-overlapped occurrences of episodes. These approaches consider either serial episodes where items (events) are strictly ordered or parallel episodes where items are unordered. Recently, Tatti and Cule (2011) introduced episodes with simultaneous events where multiple events may have the same timestamp, aiming at handling quasi-simultaneous events. These works do not address compute th whole set of frequent sequential patterns and do not address the problem of forgetting obsolete data.

Data streams introduce a challenge to sequential pattern mining since data are received at high rate and are possibly infinite. To cope with such massive data, several approaches of itemset mining in data streams have proposed approximate solutions based on landmark windows (Manku and Motwani (2002)), tilted-time windows (Giannella et al. (2004)), damped windows (Chang and Lee (2003)) or sliding windows (Li and Lee (2009)). Few proposals have been made for sequential pattern mining over data streams. Chen et al. (2005) consider multiple streams recoded as a (single) sequence of itemsets grouping the items occurring at more or less the same time. Their algorithm Mile adapts PrefixSpan to mine frequent sequences that appear in a sufficient number of fixed size windows. SPEED (Raïssi et al. (2006)) considers successive batches of sequences extracted from a data stream. Frequent sequential patterns are stored in a tilted-time based structure which prunes less frequent or too old patterns.Marascu and Masseglia (2006) propose to cluster stream data to build a summary from which sequential patterns can be extracted. IncSpam (Ho et al. (2006)) uses a bit-sequence rep-

resentation of items to maintain the set of sequential patterns from itemset-sequence streams with a transaction-sensitive sliding window. Most of these approaches aim at maintaining a good approximation of the newest and most frequent sequential patterns of the data stream. Moreover, they deal with a transaction-based scheme representation of the data whereas we are interested of mining sequential patterns from a single long sequence.

Mining sequential patterns from data streams is similar to mining sequential patterns incrementally from dynamic databases. ISE (Masseglia et al. (2003)) considers the arrival of new customers and new items and extends the set of candidate patterns from the original database accordingly. IncSP (Lin and Lee (2004)) uses efficient implicit merging and counting over appended sequences. IncSpan (Cheng et al. (2004); Nguyen et al. (2005)) maintains a tree of frequent and semi-frequent patterns to avoid many multiple scans upon database updates. However, these incremental methods handle new data but not obsolete data. Recently, some proposals have been made that view sequential pattern mining as an incremental process: a window slides continuously over the data and, as time goes by, new data are added to the window and old data are removed. SSM (Ezeife. and Monwar (2007)) maintains three data structures to mine incrementally sequential patterns from a data stream. D-List maintains the support of all items in the data stream. PLWAP tree stores incrementally frequent sequences from batches. The frequent sequential pattern tree FSP is constructed incrementally from batch to batch. Pisa (Huang et al. (2008)) mines the most recent sequential patterns of a progressive sequence database. It maintains a PS-tree (progressive sequential tree) to keep the information data from a window sliding of the stream. PS-tree stores the timestamped sequence items and also efficiently accumulates the occurrence frequency of every candidate sequential pattern. Update operations remove obsolete information and add new information to the three data structures. Recently, Patnaik et al. (2012) proposed a streaming algorithm for pattern mining over an event stream. Their aim is to extract the top-k frequent episodes of an arbitrary length l from the successive batches of a window of interest defined over an event stream. Episodes are parallel with no overlapping occurrences. They use an Apriori-like method to maintain the set of top-k episodes on each successive batch. The incremental algorithm makes use of the negative border to reduce, as much as possible, the set of candidates from one batch to the other.

3 Basic concepts and problem statement

3.1 Items, itemsets and sequences

From now on, $[n]$ denotes the set of the n first integers, *i.e.* $[n] = \{1, \dots, n\}$.

Let $(\mathcal{E}, =, <)$ be the set of items and $<$ a total order (*e.g.* lexicographical) on this set. An *itemset* $A = (a^1, a^2, \dots, a^n),\ a^i \in \mathcal{E}$ is an ordered set of distinct items, *i.e.* $\forall i \in [n-1],\ a^i < a^{i+1}$ and $i \neq j \Rightarrow a^i \neq a^j$. The size of an itemset α, denoted $|\alpha|$ is the number of items it contains. An itemset $\beta = (b^1, \dots, b^m)$ is a sub-itemset of $\alpha = (a^1, \dots, a^n)$, denoted $\beta \sqsubseteq \alpha$, iff β is a subset of α.

A *sequence* S is an ordered series of itemsets $S = \langle s_1, s_2, ..., s_n \rangle$. The length of a sequence S, denoted $|S|$, is the number of itemsets that make up the sequence. The size of a sequence S, denoted $\|S\|$, is the total number of items it contains $\|S\| = \sum_{i=1}^{|S|} |s_i|$. $T = \langle t_1, t_2, ..., t_m \rangle$ is

a *sub-sequence* of $S = \langle s_1, s_2, ..., s_n \rangle$, denoted $T \preceq S$, iff there exists a sequence of integers $1 \leq i_1 < i_2 < ... < i_m \leq n$ such that $\forall k \in [m], t_k \sqsubseteq s_{i_k}$.

Example 1. *Let $\mathcal{E} = \{a, b, c\}$ with the lexicographical order ($a < b$, $b < c$) and the sequence $S = \langle a(bc)(abc)cb \rangle$. To simplify the notation, we omit the parentheses around itemsets containing a single item. The size of S is 8 and its length is 5. For instance, sequence $\langle (bc)(ac) \rangle$ is a sub-sequence of S.*

The relation $\preceq$ is a partial order on the set of sequences.

3.2 Stream of itemsets

A *stream of itemsets* $F = \{f_i\}_{i \in \mathbb{N}}$ is an infinite sequence of itemsets that evolves continuously. It is to be assumed that only a small part of it can be kept in memory.

The *window* W of size w at time t is the sequence $W = \langle f_{i_1}, f_{i_2}, ..., f_{i_n} \rangle$ such that $\forall k \in [n],\ t \leq i_k \leq t + w$ and $\forall f_i \in F \setminus W,\ i < t$ or $i > t + w$. The *current window* of size w of a stream F is the window beginning at time $t - w + 1$ where t is the position of the last itemset appeared in the stream. This window includes the most recent itemsets of the stream.

Definition 1 (Instances of a sequence in a window)**.** *The set of instances of a sequence $S = s_1, \ldots, s_n)$ of length n in a window $W = (w_1, \ldots, w_m)$, denoted $\mathcal{I}_W(S)$, is the list of n-tuples of positions (within W) corresponding to the **minimal occurrences** of S in W (see Mannila et al. (1997)).*

$$\mathcal{I}_W(S) = \big\{ (i_j)_{j \in [n]} \in [m] \mid \begin{array}{ll} \forall j \in [n], s_j \sqsubseteq w_{i_j}, & (1) \\ \forall j \in [n-1],\ i_j < i_{j+1}, & (2) \\ (w_j)_{j \in [i_1+1, i_n]} \not\preceq S, & (3) \\ (w_j)_{j \in [i_1, i_n-1]} \not\preceq S\ \} & (4) \end{array}$$

Condition (1) requires that any itemset of S is a sub-itemset of an itemset of W. Condition (2) specifies that the order of itemsets of W must be respected. In addition, any itemset of W cannot be a super-itemset of two distinct itemsets of S. This condition does not impose any time constraint between itemsets. Conditions (3) and (4) specify minimal occurrences: if an instance of S has been identified in the interval $[i_1, i_n]$, there can't be any instance of S in a strict subinterval of $[i_1, i_n]$.

Example 2. *Let $W = \langle a(bc)(abc)cb \rangle$ be a window on some stream. We focus on instances of the sequence $S = \langle (ab)b \rangle$. To find an instance of this sequence, we have to locate itemsets of S: (ab) appears only at position 3 ($(ab) \sqsubseteq (abc)$). b appears at positions 2, 3 and 5. Sequence S has only one instance: $(3, 5)$. Thus $\mathcal{I}_W(\langle (ab)b \rangle) = \{(3, 5)\}$.*

Now, let us consider the window $W = \langle acbbc \rangle$ to illustrate the conditions (3) and (4). Without these conditions, the instances of the sequence $\langle ab \rangle$ would be $\{(1, 3), (1, 4)\}$. Condition (4) prohibits the instance $(1, 4)$ because $(1, 3)$ is an instance of $\langle ab \rangle$ in the window such that $[1, 3] \subset [1, 4]$. Thus, $\mathcal{I}_W(\langle ab \rangle) = \{(1, 3)\}$.

For $W = \langle aaaa \rangle$, $\mathcal{I}_W(\langle aa \rangle) = \{(1, 2), (2, 3), (3, 4)\}$ and $\mathcal{I}_W(\langle aaa \rangle) = \{(1, 2, 3), (2, 3, 4)\}$.

Let W be a window and S a sequence of itemsets. The **support** of the sequence S in window W, denoted $supp_W(S)$ is the cardinality of $\mathcal{I}_W(S)$, *i.e.* $supp_W(S) = card\left(\mathcal{I}_W\left(S\right)\right)$.

Note that, in the general case, the support function $supp_W(\cdot)$ is not anti-monotonic on the set of sequences with associated partial order $\preceq$ (see Tatti and Cule (2012)).

Definition 2 (Mining a stream of itemsets)**.** *Given a threshold σ, we say that a sequence S is* frequent *in a window W of size w iff $supp_W(S) \geq \sigma$. Mining a stream of itemsets consists in extracting at every time instant, all the frequent sequences in the most recent sliding window.*

In approaches that consider multiple parallel streams, *e.g.* Ho et al. (2006); Marascu and Masseglia (2006); Raïssi et al. (2006), the support of a sequence is usually defined as the number of streams in which this sequence appears. In this work, we consider a single stream and the support of a sequence is the number of instances of this sequence in the current window corresponding to the most recent data.

4 Incremental algorithm for mining a stream of itemsets

In this section, we present an incremental algorithm for mining frequent sequences in a window sliding over a stream of itemsets. The aim of such an algorithm is to efficiently update the set of frequent sequences following two kinds of window transformations: the addition of an itemset at the end of the window and the removal of an itemset at the beginning of the window.

Algorithm 1 presents this general idea of incremental mining an itemset stream. The algorithmic difficulties lie in the functions DETETEFIRST and ADD which must efficiently update the set of frequent sequences computed so far. These functions will be detailed further in this section.

Input: $\mathcal{F}$: itemset stream, w: windows size, σ: threshold
1: $t \leftarrow 1$
2: **while** $t \leq w$ **do** ▷ Stream beginning
3: $\alpha \leftarrow \mathcal{F}(t)$
4: $\mathcal{A} \leftarrow$ ADD$(\alpha, \mathcal{A})$ ▷ Update $\mathcal{A}$ by adding itemset α
5: $t \leftarrow t + 1$
6: **end while**
7: **while** true **do**
8: $\alpha \leftarrow \mathcal{F}(t)$
9: $\mathcal{A} \leftarrow$ DELETEFIRST$(\mathcal{A})$ ▷ Update $\mathcal{A}$ by removing references to the first itemset
10: $\mathcal{A} \leftarrow$ ADD$(\alpha, \mathcal{A})$
11: $t \leftarrow t + 1$
12: **end while**

FIG. 1 – *Incremental mining of a stream of itemset $\mathcal{F}$.*

The algorithm relies on representing the set of frequent sequences in a tree, the structure of which is inspired by the prefixing method of PSP of Masseglia et al. (1998). While GSP represents the set of frequent sequences of itemsets as a tree where the edges mean sequentiality, PSP represents a set of frequent sequences as a tree with two types of edges: the edges representing sequentiality between itemsets and the edges representing the composition of itemsets. Masseglia et al. showed that this representation is equivalent to GSP while requiring less memory.

4.1 Representing sequences by a PSP tree

The set of frequent sequences is represented by a prefix tree, the nodes of which have the structure defined below.

Definition 3 (Node)**.** *A node N is a 4-tuple $\langle \alpha, \mathcal{I}, \mathcal{S}, \mathcal{C} \rangle$ where*

- *$\alpha = (a_1, \ldots, a_n)$ is a sequence of size n,*
- *$\mathcal{I} = \mathcal{I}_W(\alpha)$, the instance list of sequence α in W,*
- *$\mathcal{S}$ is the set of descendant nodes which represent sequences $\beta = (b_1, \ldots, b_{n+1})$ of size $\|\alpha\| + 1$ such that $\forall i \in [n],\ a_i = b_i$, (i.e. b_{n+1} is a strict successor of a_n),*
- *$\mathcal{C}$ is the set of descendant nodes which represent sequences $\beta = (b_1, \ldots, b_n)$ of size $\|\alpha\| + 1$ such that $\forall i \in [n-1],\ a_i = b_i$, $a_n \sqsubseteq b_n$ and $\forall j < |a_n|,\ a_n^j < b_n^{|a_n|+1}$, (i.e. itemset b_n extends itemset a_n with the item $b_n^{|a_n|+1}$).*

Definition 4 (Tree of frequent sequences)**.** *$\mathcal{A}_\sigma(W)$ denotes the tree that represents all sequences of W having a support greater than σ. The root node of a prefix tree is a node of the form $\langle \{\}, \emptyset, \mathcal{S}, \mathcal{C} \rangle$.*

Let N be a node of $\mathcal{A}_\sigma(W)$. The subtree rooted at node N represents the tree composed of all descendants of N (including N).

Despite the non-general anti-monoticity property of the support, it may be proved that the support is anti-monotone wrt the PSP-tree relations. If a node has a support greater than or equal to σ then all its ancestors are frequent sequences in W. In addition, each node – apart from the root – has a single parent. This ensures that recursive processing the PSP tree is complete and non-redundant.

Example 3. *Let $W = \langle a(bc)(abc)cb \rangle$ and $\sigma = 2$. Figure 2 shows the tree $\mathcal{A}_\sigma(W)$. Solid lines indicate membership in the set $\mathcal{S}$ (Succession in the sequence), while the dotted lines indicate membership in the set $\mathcal{C}$ (Composition with the last itemset). The node $(bc)b$, highlighted in gray, has the sequence node (bc) as parent, since $(bc)b$ is obtained by concatenating b to (bc). The parent node of (bc) is (b) and is obtained by itemset composition (dotted line). At each node of Figure 2, the instance list of the sequence is displayed in the index. For example, the sequence $(bc)c$ has two instances: $\mathcal{I}(\langle (bc)c \rangle) = \{(2,3),\ (3,5)\}$.*

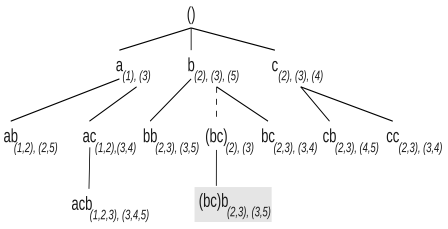


FIG. 2 – *Example of a tree of frequent sequences ($\sigma = 2$)*

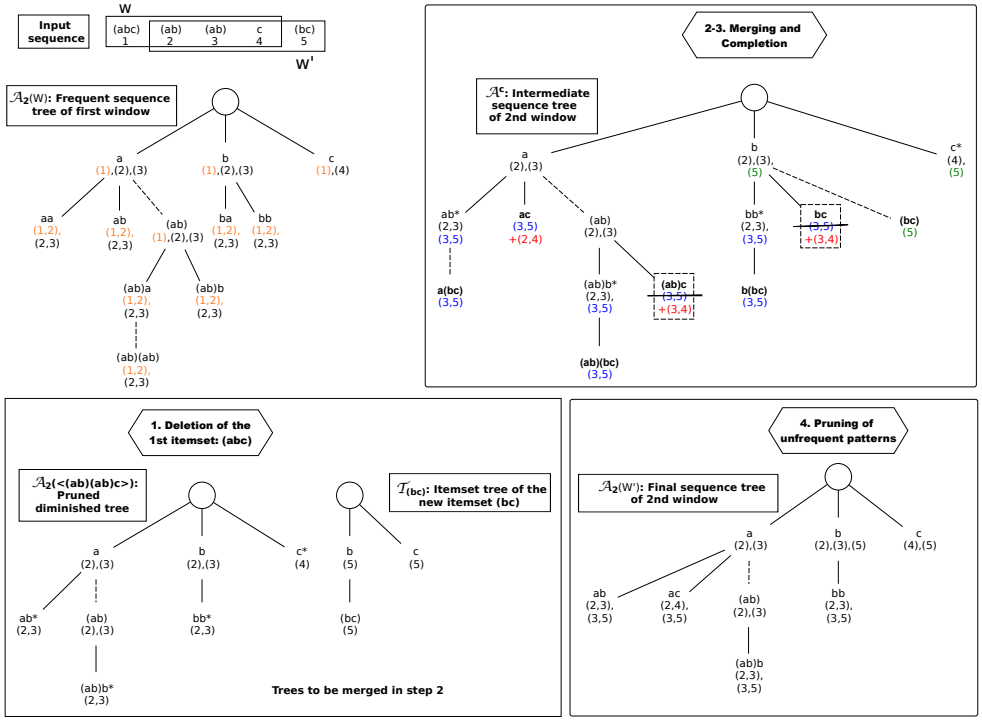


FIG. 3 – *Successive steps for updating the sequence tree upon the arrival of itemset* (bc) *in the window* $W = \langle(abc)(ab)(ab)c\rangle$.

4.2 Illustration of the method

This section illustrates the principle of the proposed method. For space reasons, the algorithm details have been omitted. The incremental process aims at updating the frequent sequence tree from data in the most recent window of the stream and determining whether the sequences are frequent. The arrival of a new itemset in the stream triggers two steps: (1) the deletion of instances related to the first itemset in the window, (2) the addition of sequences and instances related to the new incoming itemset (see Figure 1). The addition step carries contributes to most of the computational load. It involves three substeps: merging sub-itemsets of the new itemset into the current tree, completing the instance lists and pruning non-frequent sequence nodes.

The deletion step is performed before the addition of a new itemset in order to reduce the size of the tree before the time-consuming merging and completion substeps.

Let us consider the window $W = \langle(abc)(ab)(ab)c\rangle$ of length 4, at position 1 of the stream. Assume that $\mathcal{A}_2(W)$, *i.e.* the sequence tree with support greater than 2, has been already built. Figure 3 describes the successive steps for transforming the frequent sequence tree $\mathcal{A}_2(W)$ into the tree $\mathcal{A}_2(W')$ at the arrival of the new itemset (bc).

1. Deletion of the first itemset: all instances starting at the first (oldest) position of the

window (orange instances at position 1, in the example) are deleted. Then, sequences having a number of instances less than $\sigma = 2$ are deleted from the tree. The result is the tree $\mathcal{A}_2(\langle(ab)(ab)c\rangle)$ where a, (ab), b are frequent. Quasi-frequent sequences (marked with a $\star$ in the example) are not frequent but may become frequent as they have have a frequence equal to $\sigma - 1$ and they are ended by an item present in the new itemset, (bc) here. Such nodes are kept in the frequent tree with their occurence list because no completion (see below) will be necessary for them.

2. Merging the new current itemset (bc) with every node of the sequence tree: this step generates all the new candidate sequences of the new window. Intuitively, a sequence is a new candidate (*i.e.* potentially frequent) only if it is the concatenation of a sub-itemset of (bc) to a frequent sequence of $\langle(ab)(ab)c\rangle$. In the frequent sequence tree, this concatenation can be seen as extending each node of $\mathcal{A}_2(\langle(ab)(ab)c\rangle)$ with the itemset tree $\mathcal{T}_{(bc)}$ representing all sub-itemsets of (bc).

In Figure 3, the tree $\mathcal{T}_{(bc)}$ is merged with the four nodes of $\mathcal{A}_2(\langle(ab)(ab)c\rangle)$ not marked with a *:

- with the root node (green instances): all subsequences of (bc) become potentially frequent.
- with the nodes a, (ab), b (blue instances): all sequences starting with one of this three sequences (frequent in $\langle(ab)(ab)c\rangle$) and followed by a sub-itemset of (bc) become potentially frequent.

We talk about "tree merging" because if a node already exists in the tree (*e.g.* node (b)), the instance related to the new itemset is added to the list of existing instances. The list of instances of (b) becomes $\{(2), (3), (5)\}$. We know that each of these nodes holds all the instances of the associated sequence in W'. New nodes are noted in bold face in the frequent tree after the merging step in Figure 3. Each of these new nodes of $\mathcal{A}^f$, *e.g.* the node (bc), has an occurrence list consisting of only one instance of a sub-itemset of (bc). Quasi-frequent nodes (nodes marked with a *) are not merged with the itemset tree $\mathcal{T}_{(bc)}$. Their occurrence lists are simply updated, if needed.

3. Completion of instance lists: For new candidate nodes but only for those, it is necessary to scan the window W' once again to build the complete list of instances of a sequence. For example, the node ab is associated with the instance list $\{(3, 5)\}$. This list must be completed with the list of instances of ab in the previous window where it was unfrequent, *i.e.* $\{(2, 3)\}$. Red instances of the tree $\mathcal{A}^c$ in Figure 3 show the instances added by completion.

4. Pruning non-frequent sequences: $\mathcal{A}^c$, the tree obtained after completion, contains new candidate sequences with complete instance lists. The last step removes sequences with an instance list of size strictly lower than $\sigma = 2$ yielding the tree $\mathcal{A}_2(W')$.

4.3 Merging an itemset tree into a frequent sequence tree

Now, we detail the merging step which integrates the itemset tree $\mathcal{T}$ into the sequence tree $\mathcal{A}$. Then, we explain instance list completion.

Merging the itemset tree $\mathcal{T}$ with every node of the frequent sequence tree $\mathcal{A}$ consists of two main steps (see Figure 4):

- prefixing the itemset tree $\mathcal{T}$ with the sequence of node N,
- recursively merging the prefixed $\mathcal{T}$ with descendants of node N (*cf.* Algorithm 5).

```
1: function MERGING(A, T)
2:     T' ← T
3:     for N ∈ A do
4:         for n ∈ T' do                                   ▷ Prefixing T'
5:             n.α = N.α ⊕ n.α                             ▷ Prefixing the sequence with N.α
6:             for all I ∈ n.I do      ▷ Prefixing the instances with the last element of N.I, noted d
7:                 I = d ∪ I
8:             end for
9:         end for
10:        RECMERGE(T', N)                                 ▷ Recursive merging of T' with nodes N of A
11:    end for
12:    return A
13: end function
```

FIG. 4 – MERGING*: merging the itemset tree $\mathcal{T}$ with every node of the sequence tree $\mathcal{A}$.*

Let $N.\alpha$ denote the sequence associated with node N from the sequence tree $\mathcal{A}$ and $N.\mathcal{I}$ denote the instance list associated with the same node N. For each node N of $\mathcal{A}$, the itemset tree $\mathcal{T}$ is first prefixed by N: on the one hand, the sequences of each node of $\mathcal{T}$ are prefixed by $N.\alpha$; on the other hand, all instances of $\mathcal{T}$ are prefixed by the last instance of $N.\mathcal{I}$. Using the last instance of $N.\mathcal{I}$ enforces the third property of Definition 1.

Input: n: itemset node tree, N: node of the sequence tree to be merged with n and such that $n.\alpha = N.\alpha$

```
1: function RECMERGE(n, N)
2:     N.I ← N.I ∪ n.I                                     ▷ Merging instance lists
3:     for s_N ∈ N.S ∪ N.C do                              ▷ Recursion
4:         for s_n ∈ n.S ∪ n.C do
5:             if s_N.α = s_n.α then
6:                 found ← True
7:                 RECMERGE(s_n, s_N)
8:             end if
9:         end for
10:        if not found then
11:            if s_n ∈ n.S then
12:                N.S ← N.S ∪ {COPY(s_n)}
13:            else
14:                N.C ← N.C ∪ {COPY(s_n)}
15:            end if
16:        end if
17:    end for
18: end function
```

FIG. 5 – RECMERGE*: recursively merging the prefixed itemset tree $\mathcal{T}$ with a node of $\mathcal{A}$*

In a second step, the algorithm recursively merges the root of the itemset tree $\mathcal{T}$ prefixed by N. Algorithm 5 details this merging operation. We must first make sure that $n.\alpha = N.\alpha$ to verify that the two nodes represent the same sequence. At line 2, instance lists of nodes n and N are merged. By construction of the new instance, the conditions of Definition 1 are satisfied.

Then, the descendants of n are processed recursively. For each node of $n.\mathcal{S}$ (resp. $n.\mathcal{C}$), we search a node s_n in $N.\mathcal{S}$ (resp. $N.\mathcal{C}$) such that these nodes represent the same sequence. If such a node is found, then the function `RecMerge` is recursively applied. Otherwise, a copy of the entire subtree of s_n is added to $n.\mathcal{S}$ (resp. $n.\mathcal{C}$).

4.4 Instance lists completion

One of the difficult task is merging the instance lists of nodes n into the one of N. When a new sequence is introduced in the tree, other instances of this sequence may be present in the previous window (corresponding to the beginning of the current window) but unfrequent and, so, they were not stored in the tree (except quasi-frequent sequences). For example, in Figure 3, the sequence $\langle bc \rangle$ (node surrounded by a dotted line square) is not frequent in W and is not present in the frequent sequence tree $\mathcal{A}_2(W)$. However, after the arrival of itemset (bc) the sequence $\langle bc \rangle$ may become frequent in W. Thus, it is necessary to scan W' to find all instances of $\langle bc \rangle$ to compute its frequency.

For thesake of completeness, the instance list must be completed. To make the completion efficient, the algorithm uses the projection principle of PrefixSpan to reduce the number of scans over the sequence W. For succession nodes, the completion scans only the sub-sequence of W' composed of the itemsets between $i_{|\beta|} + 1$ and $j_{|\beta|-1}$, where $J = (j_1, ..., j_{|\beta|})$ is the instance after I in the list of instances of β. Properties (2) and (4) of Definition 1 ensure that the completed instance lists are correct and complete. Thus, the overall algorithm is complete and correct.

4.5 Time complexity analysis

To the best of our knowledge, there exists no results about the theoretical complexity of the minimal occurrence mining task. Ding et al. (Ding et al. (2009)) have shown that testing whether the support of a sequence is equal to k is NP for counting non-overlapping instances and NP-complete for counting strong non-overlapping instances. However, these two counting methods are not equivalent to minimal occurrences counting.

Two instances $(i_j)_{j\in[m]}$ and $(i'_j)_{j\in[m]}$ of a sequence S of size m are overlapping if and only if $\exists 1 \leq l < m : i_l = j_l$. Two instances are strong-overlapping if and only if $\exists 1 \leq l < m$ and $1 \leq l' < m : i_l = j'_l$.

In this section, we estimate the time complexity of updating the frequent tree. For this purpose, we consider a sequence of items, *i.e.* consisting of itemsets of size 1 only.

Proposition 1. *Let W be a window of size w, ql the size of the vocabulary and σ the frequency threshold. Considering a sequence S of size n, we denote by $N_{ql,n,\sigma}$ the number of nodes in the tree $\mathcal{A}_\sigma(S)$.*

The time complexity of updating a frequent sequence tree can be decomposed as follows:

1. *Deletion of obsolete instances: $\mathcal{O}(N_{ql,w,\sigma})$,*
2. *Merging the current itemset tree with the current frequent sequence tree: $\mathcal{O}\left(N_{ql,w-1,\sigma}\right)$,*
3. *Completion of instance lists (in the worst case): $\mathcal{O}\left(N_{ql,w-1,\sigma} \times w\right)$*

In the worst case, the time complexity of updating a frequent squence tree is
$\mathcal{O}\left(N_{ql,w,\sigma} \times (1 + w) + N_{ql,w-1,\sigma}\right)$

Proof. 1. The deletion of the obsolete itemsets is performed by processing each node of the tree. Considering that we cope with minimal occurrences only and that instance lists are ordered from the oldest instance to the newest, it is sufficient to process the first instance of instance lists. Pruning is performed during the same operation. Thus, the time complexity of itemset deletion is $\mathcal{O}(N_{ql,w,\sigma})$. This step yields the pruned diminished sequence tree.

2. In case of a sequence of items $\mathcal{T}$, the tree associated with the new itemset, has only one node. In this case, the merging operation consists in processing each node of the frequent sequence tree sequentially by adding an instance or by adding a new node. The overall time complexity is then $\mathcal{O}(N_{ql,w-1,\sigma})$ where $N_{ql,w-1,\sigma}$ is the size of the frequent sequence tree corresponding to the old window where the first itemset has been deleted. This step yields the merged sequence tree.

3. In the worst case, *i.e.* when the new item is not in the merged sequence tree, the number of created nodes is equal to the number of nodes in the pruned diminished sequence tree, thus it is lower or equal to $N_{ql,w-1,\sigma}$. For each node, the algorithm has to complete the instance list. In the worst case, the algorithm browses the whole sequence W.

□

5 Frequent sequences history over the stream

In real applications, it is interesting to highlight the mode changes of the observed system by analyzing its behavior through frequent sequences. In this section, we describe the construction of an history of frequent itemsets without trivial matches (Lin et al. (2002)). The history data structure is computed while processing a stream to give a quick view of the distribution of frequent patterns over the stream. We do not address the issue of history compression such as some proposals, *e.g.* Tilted-Time Windows (Giannella et al. (2003)) or REGLO (Marascu and Masseglia (2006)). The memory space required by this data structure grows linearly with time.

Definition 5 (History of frequent sequences)**.** *Let S be a sequence. $\mathcal{H}^{\sigma,w}(S)$ denotes the history of the sequence S over a stream and is defined by*

$$\mathcal{H}^{\sigma,w}(S) = \left([l_1, u_1], \ldots, [l_m, u_m]\right),$$

with $l_i, u_i \in \mathbb{N}$ such that $\forall i \in [m],\ 0 < l_i \leq u_i$ and $\forall i \in [m-1],\ u_i < l_{i+1}$.

This means that the sequence S is frequent, according to the threshold σ, in the windows of size w starting at time t in the stream for all $t \in [l_i, u_i]$ with $i \in [m]$.

Algorithm of Figure 6 describes the function that updates the sequence history with the current frequent sequences. This function may be inserted in the algorithm of the Figure 1 line 11 in order to construct the sequence history on the stream. The principle of the algorithm is to update the history at the end of the processing of the arrival of a new itemset in the sequence. For each sequence frequent in the window beginning at time t, if the sequence was already frequent at time $t-1$ the last interval is enlarged with the date of the current window ; if the sequence was unfrequent at $t-1$ a new interval is created.

Input: t: current window date, $\mathcal{A}$: frequent sequences PSP-tree, $\mathcal{H}^{\sigma,w}$: sequences history

```
1:  function HISTORYUPDATE(t)
2:      for all N ∈ A do                          ▷ For all frequent sequence
3:          ([l_i, u_i])_{i∈[m]} ← H^{σ,w}(N.α)
4:          if u_m = t − 1 then
5:              u_m ← t − 1
6:          else
7:              H^{σ,w} ← H^{σ,w} × [t, t]
8:          end if
9:      end for
10:     return H^{σ,w}
11: end function
```

FIG. 6 – HISTORYUPDATE*: updating the sequence history.*

Example 4 (History of frequent sequences)**.** *Given the stream* $\mathcal{F} = \langle a(bc)ab(abc)abc(ab)a\rangle$*, the history of frequent sequences for* $\sigma = 2$ *and* $w = 4$ *is illustrated by Figure 7. Using the proposed interval-based structure, the history for sequences* a *and* ba *are respectively* $\mathcal{H}^{\sigma,w}(a) = \{[1,7]\}$ *and* $\mathcal{H}^{\sigma,w}(ba) = \{[2,3],[7,7]\}$.

In this Figure 7, a black cell indicates that the sequential pattern for this line is frequent at the stream position in column. For instance, the sequential pattern $\langle ab\rangle$ *were frequent only in the first window of the stream while sequencial patterns* a *and* b *were frequent all over the stream.*

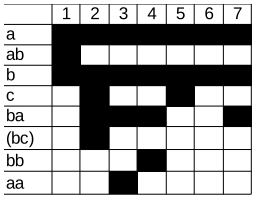


FIG. 7 – *History of frequent sequences.*

6 Experiments and results

Insofar, as there is no pre-existing method that performs the same task as our algorithm named SEQ, we developed a second naïve algorithm, from now on named BAT. BAT performs the same task as SEQ, *i.e.* extracts the frequent sequences from a window sliding on a data stream, but non incrementally. This algorithm, based on PrefixSpan and using the PSP tree structure, rebuilds the entire tree $\mathcal{A}_\sigma(W)$ for each consecutive window of size ws on the data stream.

The algorithms were developed in C++ and executed on a processor at 2.2 GHz, with 2GB of RAM.

6.1 Experiments on simulated data

In this section, we compare the results obtained by SEQ with those of algorithm BAT on experiments over simulated data produced by the IBM data generator. This generator is usually used to generate transactions, but it can generate a single long transaction as well. The transaction will simulate a data stream input to algorithms. The results were obtained from 1200 executions performed by varying the parameters ws (window size), σ (minimal support) and ql (item vocabulary size, $card(\mathcal{E})$).

The curves below were obtained by averaging the results over all the experiments. For example, the point on the curve SEQ of Figure 8 - (a) for $\sigma = 5$ is obtained by averaging all the results of experiments running SEQ with $\sigma = 5$ when the other parameters vary freely.

We can note first that the memory usage of SEQ is slightly higher than BAT. This comes from the fact that SEQ makes use of merged trees ($\mathcal{A}^c$ in Figure 3) the size of which is a little larger than the basic tree of frequent sequences. The tree size decreases exponentially when the vocabulary size grows or the support threshold increases. We can observe this same trend for memory usage. Finally, we note in Figure 8-(f) that, on average, the window size ws has a low influence on the required memory.

Figures 8-(a), (c) and (e) show that the computation time of both algorithms is exponential with the window size ws, but it grows faster than exponential as σ or ql decreases. In such cases, the number of frequent sequences increases and trees are larger.

On average, the computation time of SEQ is 80 % lower than the computation time of BAT. Figure 8-(a) shows that the more σ decreases, the larger is the performance gap between the two algorithms. By contrast, Figure 8-(e) shows that this improvement decreases with the window size (77 % for $ws = 25$).

6.2 Experiments on smart electrical meter data

Smart electrical meters record the power consumption of an individual or company in intervals of 30 mn and communicate that "instant" information to the electricity provider for monitoring and billing purposes. The aim of smart meters is to better anticipate the high consumption of a distribution sector by awarding a consumption profile to each meter, that is to say, a dynamic model of changes in consumption. However, consumption profiles are not stable over time. Depending on the period of the year (seasons, holidays), of the week (weekdays, weekends) or of the day, consumption patterns change in a non-predictable manner. Consequently, there is no meter profile to predict medium to long-term consumption. Our algorithm can be used to extract, online, profiles of short-term consumption. Similar individual household electric power consumption may be downloaded from the UCI repository (see Frank and Asuncion (2010)).

The annual series of instantaneous consumption is a flow of about 18,000 values. We use the SAX algorithm (Lin et al. (2003)) to discretize the set of consumption values. A vocabulary size of $|\mathcal{E}| = 14$ and a PAA aggregation window $W = 24$ have been chosen. The consumption profile of a smart meter at time t is the set of frequent consumption sequences during the period $[t - w, t]$ (sliding window of predefined size $w = 28$ itemsets, *i.e.* 2 weeks).

Figure 9 shows the results for 40 meters. The results obtained on these real data are close to those obtained on simulated data. On the one hand, memory usage is slightly higher for the

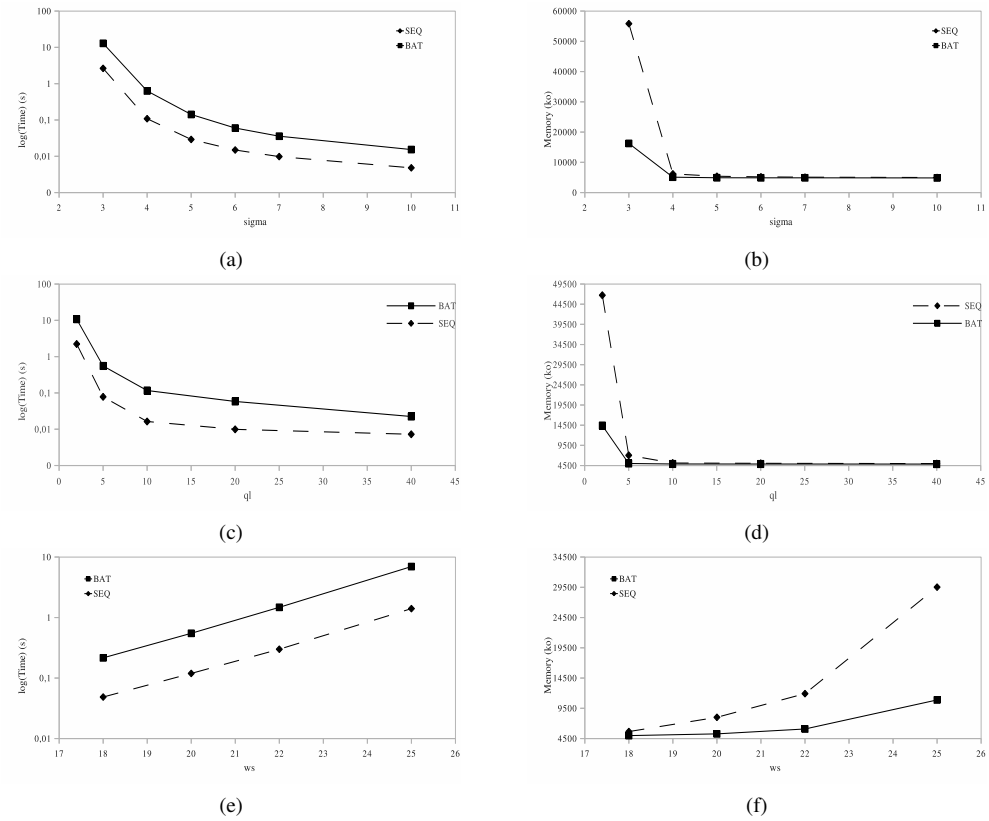


FIG. 8 – *Comparison of processing time (logarithmic scale) and memory usage with respect to the support threshold σ (with $ws < 25$), the number of symbols ql (with $\sigma > 3$) and the size of the sliding window ws.*

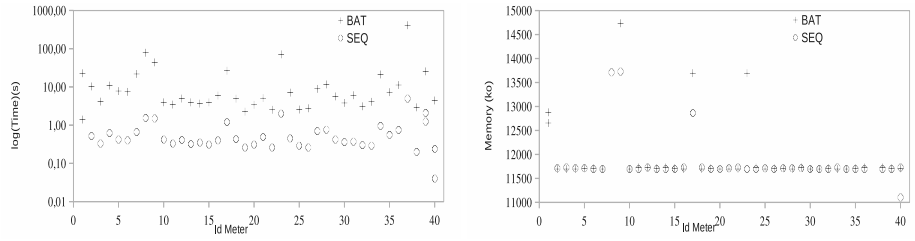


FIG. 9 – *Comparison of computation time (left) and memory usage (right) for the mining power consumption streams.*

incremental algorithm but on the other hand, the processing time of SEQ is improved by 89%, on average, compared to BAT.

For some meters processing time is very long (about few minutes) while for most of the meters processing times are around seconds. This disparity is explained by the observed variability of consumption. The sequences that are difficult to process are quite constant (*e.g.* industrial consumption). These sequences include many symbol repetitions leading to many frequent sequences of repeated symbols.

7 Conclusion and perspectives

We have presented the problem of mining frequent sequences in a sliding window over a stream of itemsets. To address this problem, we have proposed an incremental algorithm that efficiently updates a tree data structure inspired by PSP. Our algorithm is based on counting minimal occurrences of a sequence in a window. The proposed algorithms are complete and correct.

Experiments conducted on simulated data and real instantaneous power consumption data show that our algorithm significantly improves the execution time of an algorithm based on a non-incremental naïve approach. For both algorithms, the time complexity is exponential with the size of the sliding windows and beyond exponential with respect to the number of items or the support threshold. These execution time performance were obtained with a memory usage close to the one of the naïve approach.

Future work will consider incremental mining of multiple data streams. In particular, the proposed tree representation of frequent sequences can be extended to design an algorithm that can extract frequent sequences in multiple itemsets streams and take into account the repetitions in each stream. Also, a lot research has been done on condensed representations of patterns e.g. closed patterns. We want to investigate such condensed representations in the context of incremental mining.

The frequent sequence history built from the stream of itemset proposes a new view on the stream. It would be insteresting to mine it in order to highlight concept changes. The adaptation of algorithms for mining interval-based sequences, such as the method proposed in Guyet and Quiniou (2011), to streams would be a possible solution but constitutes a great challenge.

References

Achar, A., S. Laxman, and P. S. Sastry (2010). A unified view of automata-based algorithms for frequent episode discovery. *CoRR abs/1007.0690*.

Agrawal, R., T. Imielinski, and A. Swami (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 207–216.

Ayres, J., J. Flannick, J. Gehrke, and T. Yiu (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 429–435. ACM.

Chang, J. H. and W. S. Lee (2003). Finding recent frequent itemsets adaptively over online data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 487–492.

Chen, G., X. Wu, and X. Zhu (2005). Sequential pattern mining in multiple streams. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 585–588.

Cheng, H., X. Yan, and J. Han (2004). IncSpan: incremental mining of sequential patterns in large database. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 527–532.

Ding, B., D. Lo, J. Han, and S.-C. Khoo (2009). Efficient mining of closed repetitive gapped subsequences from a sequence database. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pp. 1024–1035.

Ezeife., C. and M. Monwar (2007). A PLWAP-based algorithm for mining frequent sequential stream patterns. *International Journal of Information Technology and Intelligent Computing 2*(1), 89–116.

Frank, A. and A. Asuncion (2010). UCI machine learning repository.

Giannella, C., J. Han, J. Pei, X. Yan, and P. S. Yu (2003). Mining frequent patterns in data streams at multiple time granularities. In K. S. H. Kargupta, A. Joshi and Y. Yesha (Eds.), *Proceedings of the Next Generation Data Mining Workshop*.

Giannella, C., J. Han, J. Pei, X. Yan, and P. S. Yu (2004). *Next generation data mining*, Chapter Mining frequent patterns in data streams at multiple time granularities. AAAI/MIT Press.

Guyet, T. and R. Quiniou (2011). Extracting temporal patterns from interval-based sequences. In *Proceedings of International Join Conference on Artificial Intelligence*, pp. 1306–1311.

Ho, C.-C., H.-F. Li, F.-F. Kuo, and S.-Y. Lee (2006). Incremental mining of sequential patterns over a stream sliding window. In *IWMESD Workshop at ICDM*, pp. 677 –681.

Huang, J.-W., C.-Y. Tseng, J.-C. Ou, and M.-S. Chen (2008). A general model for sequential pattern mining with a progressive database. *IEEE Transactions on Knowledge and Data Engineering 20*(9), 1153 –1167.

Laxman, S., P. S. Sastry, and K. P. Unnikrishnan (2007). A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 410–419.

Li, H.-F. and S.-Y. Lee (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. *Journal of Expert Systems with Applications 36*(2), 1466–1477.

Lin, J., E. Keogh, S. Lonardi, and B. Chiu (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery*.

Lin, J., E. Keogh, S. Lonardi, and P. Patel (2002). Finding motifs in time series. In *Proceedings of the 2nd Workshop on Temporal Data Mining*, pp. 53–68.

Lin, M.-Y. and S.-Y. Lee (2004). Incremental update on sequential patterns in large databases by implicit merging and efficient counting. *Journal of Information Systems 29*, 385–404.

Manku, G. S. and R. Motwani (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pp. 346–357.

Mannila, H., H. Toivonen, and A. I. Verkamo (1997). Discovering frequent episodes in event sequences. *Journal of Data Mining and Knowledge Discovery 1*(3), 210–215.

Marascu, A.-M. and F. Masseglia (2006). Mining sequential patterns from data streams: a centroid approach. *Journal for Intelligent Information Systems 27*, 291–307.

Masseglia, F., F. Cathala, and P. Poncelet (1998). The PSP approach for mining sequential patterns. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, pp. 176–184.

Masseglia, F., P. Poncelet, and M. Teisseire (2003). Incremental mining of sequential patterns in large databases. *Journal of Data and Knowledge Engineering 46*, 97–121.

Méger, N. and C. Rigotti (2004). Constraint-based mining of episode rules and optimal window sizes. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 313–324.

Nguyen, S. N., X. Sun, and M. E. Orlowska (2005). Improvements of IncSpan: Incremental mining of sequential patterns in large database. In *Proceedings of the 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining*, pp. 442–451.

Patnaik, D., N. Ramakrishnan, S. Laxman, and B. Chandramouli (2012). Streaming algorithms for pattern discovery over dynamically changing event sequences. *CoRR abs/1205.4477*.

Pei, J., J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu (2004). Mining sequential patterns by pattern-growth: the prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering 16*(11), 1424–1440.

Raïssi, C., P. Poncelet, and M.Teisseire (2006). Need for SPEED: Mining sequential pattens in data streams. In *Proceedings of Data Warehousing and Knowledge Discovery*.

Srikant, R. and R. Agrawal (1996). Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pp. 3–17.

Tatti, N. and B. Cule (2011). Mining closed episodes with simultaneous events. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1172–1180.

Tatti, N. and B. Cule (2012). Mining closed strict episodes. *Data Mining and Knowledge Discovery 25*(1), 34–66.

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Journal of Machine Learning 42*(1/2), 31–60.

Résumé

We introduce the problem of mining frequent sequences of itemsets in a window sliding over a stream of itemsets. To address this problem, we present a complete and correct incremental algorithm based on a representation of frequent sequences inspired by the PSP algorithm and a method for counting the minimal occurrences of a sequence. The experiments were conducted on simulated data and on real instantaneous power consumption data. The results show that our incremental algorithm improves significantly the computation time compared to a non-incremental approach.

Index

A

Azzag, Hanane 21

C

Chivukula, Aneesh 2

D

Doan, Nhat-Quang 21

G

Guyet, Thomas 38

L

Lamirel, Jean-Charles 2
Lebbah, Mustapha 21

M

Marascu, Alice 1

Q

Quiniou, René 38