Applied Intelligence (2021) 51:506–526
https://doi.org/10.1007/s10489-020-01806-0

A multi-valued and sequential-labeled decision tree method for recommending sequential patterns in cold-start situations

Chang-Ling Hsu[1]

Published online: 19 August 2020

Abstract

We plan to recommend some initial suitable single-itemed sequences like a flight itinerary based on a preference pattern in the form of personalized sequential pattern to each cold-start user. However, sequential pattern mining has never treated a conventional sequential pattern as a personalized pattern. Besides, as a cold-start user lacks the personalized sequential pattern, collaborative filtering cannot recommend one any single-itemed sequences. Thus, we first design such a preference pattern, namely representative sequential pattern, which reflects one's main frequently recurring buying behavior mined from the item-sequences during a time period. After sampling a training-set from non-cold-start users who prefer similar items, we propose an auxiliary algorithm to mine the representative sequential pattern as the sequential class labels of each training instance. A multi-label classifier seems therefore be trained to predict the sequential-label for each cold-start user based on one's features. However, most multi-label classification methods are designed to classify data whose class labels are non-sequential. Besides, some of the predictor attributes would be multi-valued in the real world. Aiming to handle such data, we have developed a novel algorithm, named MSDT (Multi-valued and Sequential-labeled Decision Tree). Experimental results indicate it outperforms all the baseline multi-label algorithms in accuracy even if three of them are deep learning algorithms.

Keywords Classification · Sequential pattern mining · Personalized recommendation · Data mining · Machine learning

1 Introduction

Businesses have popularly used recommender systems to identify interesting product/service items for their customers [28]. Related recommender technologies such as sequential pattern mining and classification methods also have been widely applied to analyze sequence data. Sequence data can be a chronological list of product/service items. A sequential pattern can be an item-set pattern, a string pattern (e.g., words or DNA) [8], or a trajectory pattern [44]. Although each element of an item-set pattern usually contains more than a single item in sequential pattern mining, each element of a string pattern and a trajectory pattern is only a single item. Some applications intend to handle some spatial-temporal or state-temporal sequence data, which reflect that a user cannot visit more than one location or one course at the same time. A single-itemed sequence (hereinafter referred to as item-sequence) can thus be a sequence of locations like a flight/tour itinerary or states like a customer purchasing/engagement trend.

Furthermore, some researches have focused on the study of the cold-start user problem in the personalized recommendation. Cold-start users are those who have bought nothing or not enough items so that recommender systems cannot recommend them any items or item-sequences online or offline. To tackle the cold-start problem of the item-sequences, one approach is to recommend an initial few item-sequences to a cold-start user and use the feedback to learn a user profile with a preference pattern. The learned profile can then be used to recommend some item-sequences to the cold user. The consequence as the item recommendation that [3] commented: in the absence of a good user profile, the recommendations are like random probes, but if not chosen judiciously, bad or too many recommendations may turn off a user.

Therefore, the priority is to recommend initial suitable item-sequences to each cold-user. Related recommender technologies concern about how to elicit a user's preference patterns from item-sequences. Two types of approaches

✉ Chang-Ling Hsu
johnny@mail.mcu.edu.tw

[1] Department of Information Management, Ming Chuan University, 5 De Ming Rd., Gui Shan District, Taoyuan City, 333, Taiwan

might be considered. One is sequential pattern mining, and the other is collaborative filtering (CF). Sequential pattern mining can be used to mine baskets of buying sequential pattern from item-sequences of non-cold-users. However, the subject of aggregation for the supports of the learned patterns are not aimed to calculate for each individual user; i.e. they are not user-centric. The aggregation is for the users, who bought items appeared in a basket across baskets sequentially. It neglects who the user is and what one's preferences are. Contrarily, the subject whom CF aims to predict (filter) by analyzing preferences of items for is each individual user; i.e. they are user-centric. It predicts items of one user using the opinions in the form of items of others [31]. However, CF neglects whether items were rated or bought sequentially.

Although a conventional sequential pattern is not user-centric, we still can represent a preference pattern in the form of personalized sequential pattern for the following reasons. First, we knew that individual user's purchasing behavior sometimes shows personalized sequential patterns, i.e. frequently recurring sub-sequences, which can be discovered from one's item-sequences. Second, [11] deem that the maximal sequential patterns are representative of all the corresponding sequential patterns. It is because the maximal sequential pattern is a sequential pattern not included in another sequential pattern [12]. However, each user often has more than one maximal sequential pattern. Thus, we design such a preference pattern, namely representative sequential pattern, which represents all the maximal sequential patterns of the user, to reflect one's main frequently recurring buying behavior. For instance, a tourist has a representative sequential pattern, <Taipei Bangkok Zurich Hongkong Taipei>, mined from one's three flight itineraries.

Now, we propose to predict a representative sequential pattern based on the features of a cold-start user. We sample a training dataset with features and item-sequences from the non-cold-start users who prefer similar items; and, the item-sequences are collected in non-real-time data streams. Our strategy is to combine the technologies of sequential pattern mining and classifying as follows. First, for each training data instance, we use a maximal sequential pattern analyzer to mine a personal representative sequential pattern for each non-cold-start user from its past behavior during a user-specified time period. Second, given some features of each user can cause one's item-sequences, we use a supervised classification method to learn a classifier from the training-set, each of which has the mined representative sequential pattern, labeled as a sequence of class labels (namely a sequential-label). Third, the classifier is then used to predict each cold-start user an initial representative sequential pattern based on one's features. Finally, the predicted patterns are used for the recommendation.

However, the question then arises on how to get such a supervised classification method. We can see that a sequential-label is not only sequential but also multi-labeled. A multi-label classification method can handle data with a multiple-classed label [5], but they assume the representative sequential pattern as a class label to be non-sequential. Thus, a requirement arises immediately is how to design a new classification algorithm that can handle each data instance with a sequential-label. In addition, the data's predictor attributes would be multi-valued in the real world. Besides, some applications require the classifier to be interpretable. One example is for marketing plan. managements sometimes require profiling what common features of some non-cold-start users show which pattern of a flight itinerary, a tour itinerary or a logistic delivery route. As a decision tree classifier has the interpretability, we propose to learn such a classifier from a multi-valued and sequential-labeled training set. An example of the training set is illustrated in Table 1. The training set has four predictor attributes, one attribute with multiple item-sequences, and one class-label attribute containing a sequential-label. Except the predictor attribute, *hobby*, is multi-valued, the other predictor attributes are single-valued.

Aiming to handle such data, this research first has used a maximal sequential pattern algorithm to acquire the sequential-label of each training instance from the item-sequences. The item-sequences were collected in advance from non-cold-start users within some successive sliding windows during a user-specified time period. Second, we have developed a novel multi-valued decision tree method, which has a sequential pattern analyzer used in each tree-node growing, named MSDT (Multi-valued and Sequential-labeled Decision Tree).

The remainder of this paper is organized as follows. Section 2 reviews the related research work. Section 3 gives notations and preliminaries. Section 4 describes the algorithms. Section 5 designs experiments. Section 6 gives the detailed experimental results and discussion. Finally, Section 7 draws conclusions.

2 Related work

The current work relates to approaches of sequential pattern analyzers, decision tree classification and sequence classification. We review them selectively in this section to provide a context for this work.

2.1 Approaches of sequential pattern analyzers

Most of the current sequential pattern algorithms can discover frequent sequential patterns, SPAM [2] especially,

Table 1 A training set with 15 customers

User id	Education level	Income	Gender	Hobby	Sequences	Sequential $-label*$
1	A	100	Female	arts	$< 123 >, < 1253 >$	123
2	B	880	Male	arts	$< 423 >, < 231 >$	23
3	A	370	Female	arts, shopping	$< 12 >, < 13 >$	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
15	B	520	Female	arts, sports, shopping	$< 31 >, < 3 >$	3

The sequential-label attribute contains the representative sequential pattern, mined from the "sequences" attribute under the constraint of the minimum support count, 2

and the maximal sequential patterns, VMSP [12, 13] especially. They both can save memory space and get time efficiency. An experimental study on five real datasets shows that VMSP is up to two orders of magnitude faster than the MaxSP algorithm [11]. Both of them are the type of vertical format-based algorithm, the vertical structure of which has been used to store each item-set of a transaction database in the SPADE algorithm [42]. This approach makes the storage and searching space smaller, and the mining process more quickly in memory.

Our work differs from the existing studies in two aspects: (1) Initially, we have applied VMSP to mining some maximal sequential patterns for each user. However, as there could be several maximal sequential patterns, which should be further pre-processed to choose only one representative sequential pattern for MSDT. (2) While growing each node of a decision tree, MSDT needs to discover both of frequent and infrequent sequential patterns among sequential-labels in order to get distinguishable among various sequential-labels. VMSP removes infrequent items in advance from the sequences during the early stage of the growing phase. Fortunately, SPAM does not remove infrequent items during the tree-growing phase. However, SPAM can only calculate the support count rather than support. We therefore need to revise it to enable the calculation.

2.2 Approaches of decision tree classification

Given a decision tree classifier, $C : x \rightarrow l$, where x is a sequence of feature-conditions of internal nodes, and l is the class label of a leaf node. According to the type of data, we review the current classification methods through the three categories as: (a) x's features are single-valued and l is single-labeled: The methods are such as ID3 [23, 24], IC [1], C4.5 [25], CART [32], ExtraTree [14], RandomForest [4] and ExtraTrees [14]. The measures for selecting the single-valued splitting attribute are such as *information gain* [23], *gain ratio* [24] and *gini* [32]. They are based on entropy/impurity and scoring only among single-labels. (b) x's features can be multi-valued and l is multi-labeled: The methods are such as our previous works, MMC [5] and MMDT [7]. MMC proposed each multi-label as a label-set initially. The measures for selecting the multi-valued splitting attribute are such as *weighted-similarity* of MMC [5] and *similarity ratio* of MMDT [7]. Additionally, the combination of the discretization algorithm, MMD [36], with MMDT (namely MMD+MMDT here) handling multi-intervals discretization of continuous attributes can refine MMDT. However, the setting of MMD only focused on the associations between the attributes and the non-sequential-labels. (c) x's features are single-valued and l is multi-labeled: The methods such as iSOUP-Tree [20] and the other methods implemented by the Scikit-learn Python package [21, 30] include DecisionTreeClassifier (namely CART-ML here) extending CART, ExtraTreeClassifier (namely ExtraTree-ML here) extending ExtraTree, RandomForestClassifier (namely RandomForest-ML here) extending RandomForest and ExtraTreesClassifier (namely ExtraTrees-ML here) extending ExtraTrees. The iSOUP-Tree method uses the measure, the mix of the ICVarR heuristic and the Hoeffding bound. All the other methods use the measures of *information gain* or *gini*.

Different from the existing studies, the selecting measure for splitting attributes of our work needs to be redesigned. We state the reason as follows. The calculations of all the measures are mainly based on the support of each class label l in each growing node. However, the counting for its support counts is different from that by both of the single-labeled methods and the multi-labeled methods. As the single-labeled methods consider each multi-label or sequential-label as a label identity, they would treat the various but similar labels as totally different and mutual exclusive labels. However, there exists similarity among the various labels because they may have common sub-sequences. As the multi-labeled methods consider each sequential-label as a label-set without any chronological order meaning, they would treat two sequential-labels like "123" and "321" as the same.

2.3 Approaches of sequence classification

Sequence classification handling data with a sequence of labels is termed sequence labeling or labeling sequence [41]. In the following, we will show two limits of current sequence labeling approaches while solving the classification problem of this study.

First, the current methods neglect classifying data according to an actor's explicit features and consulting this actor's sequence of labels caused by those features. [41] deem that sequence classification is different from the conventional classification task on feature vectors. It is because the sequences handled by the former do not have explicit features, extracted from the nature of an actor. For example, in the field of technical analysis of stock trends, the price of a stock is predicted according to its implicit features (e.g. annual price trend), extracted from historical price sequences, using an extrapolation of the price pattern [15] or using correlations between time series in technical analysis [29]. Both do not consider the explicit features such as the general or the financial attributes of the stock company.

Second, the definition of the labels are different from that of our sequential-label in the following two points. The first point is that the labels of the former do not denote a representative sequential pattern. The second point is that the labels of the former are not a strong sequence of class labels. A strong sequence of class labels is denoted as $l^+ \in L^+$; where L^+ is the set of all non-null sequences of elements of L, and L is a set of class labels [18]. In other words, a strong sequence of class labels is any one of the set of all the possible sequences of the class labels in L. A sequential-label in this study is also a strong sequence of class labels. Nevertheless, the current sequence labeling methods have only focused on handling a non-strong sequence decomposition, which only takes account of contiguous sub-sequences and thus neglects parts of all the possible sub-sequences. For instance, given that a sequence, $< 123 >$, has total of 7 sub-sequences, containing $< 1 >$, $< 2 >$, $< 3 >$, $< 12 >$, $< 13 >$, $< 23 >$ and $< 123 >$. Since event "1" and "3" are not neighbors, a non-strong sequence decomposition only takes six of the 7 sub-sequences containing $< 1 >$, $< 2 >$, $< 3 >$, $< 12 >$, $< 23 >$ and $< 123 >$ and neglects $< 13 >$.

3 Preliminaries

Before describing the MSDT algorithm, we will formally define some preliminaries related to the classification lifecycle, including some notations, an auxiliary function and an auxiliary algorithm. The details are described according to the context of the classification lifecycle, the data preparing phase and the training phase respectively as follows.

3.1 The context of classification lifecycle

Initially, we plan to acquire the data source, R, which has each user data from both of non-cold-start users and cold-start users. And each data instance of R represents a data state of the user j, $R_j = (j, x_j, S_j)$, where x_j is the feature vector; S_j is the set of item-sequences acquired within user-specified ω successive sliding windows in a non-real-time data stream of items during a user-specified time period. Additionally, the length of S_j always have an upper bound requirement specified by users in some applications. For instance, a tourist has three flight itineraries, each of which is collected within a sliding window with size six to constrain six airports at most during the most recent quarter. As the successive sliding windows are non-overlapped one another, each sliding window, W_i, is defined as a window with a user-specified upper bound of the window size, α, where $i = 1..\omega$.

Definition 1 A *set of item-sequences* of the user j within ω successive sliding windows during a user-specified time period is defined as: $S_j = \{S_{ji} | i = 1..p$, where S_{ji} is an item-sequence$\}$. The maximal length of all S_j in the data source, R, within a sliding window is equal to the upper bound of window size, α. Each $S_{ji} =< e_1 \cdots e_i \cdots e_k >$, is a chronological list elements, where each element is a transaction containing only a single-item, $e_i \in E = \{E_t | E_t$ is a single-item, where $t = 1..v\}$. The length of each S_{ji} is the number of single-items. In R, each transaction record of the user j consists of the vector, (*user-id*, *time-stamp* and a single event), done by the user. An event here is defined as a single-item or a transaction location happened at the time-stamp. We should collect each S_{ji} of S_j from a non-real-time data stream of the user transactions, $stream_j$, within each sliding window, W_i.

Example 1 An example with two users is shown in Fig. 1. Let each single-item belongs to $E = \{1, 2, 3, 4, 5\}$, and $\alpha = 4$. As $\alpha = 4$, the maximal length of all item-sequences for each user is 4. The first user, *buyer1*, has four transaction records ordered by time-stamp: (*buyer1*, t_1, 1), (*buyer1*, t_2, 2), (*buyer1*, t_3, 3) and (*buyer1*, t_4, 2), so that *buyer1* has an item-sequence, $< 1232 >$ within the sliding window, W_1. As for the other user, *buyer2* has three such transaction records: (*buyer2*, t_1, 2), (*buyer2*, t_2, 4) and (*buyer2*, t_4, 5) that *buyer2* has $< 245 >$ in the sliding window, W_1. To acquire the set of item-sequences, let $\omega = 3$ to collect the transaction records in three successive sliding windows. Besides $< 1232 >$, we can see *buyer1* has two subsequent

item-sequences, $< 2435 >$ and $< 3214 >$. Finally, we have gotten the set of item-sequences, $\{< 1232 >, < 2435 >, < 3214 >\}$, for *buyer1*.

To clarify the ambiguity among different type of users, we further define the following terms.

Definition 2 A *user* here is defined as a seller or a buyer. A seller is a user who operates the algorithms in this paper. A buyer is a website user who initially registers one's profile for membership online. And then, the buyer can buy items online, offline or both in the field of O2O (Online to Offline) e-commerce [40]. Therefore, to collect those transaction data of each user completely, we further name them as online buyers, offline buyers and online&offline buyers. They can be recommended online, offline and online&offline respectively. Buyers can be categorized into two user types: cold-start users and non-cold-start users.

Definition 3 A *representative sequential pattern*, RS_j, of user j is defined as: the maximal sequential pattern that contains single-items with the largest support in a set of the maximal sequential patterns, MP_j. If there are more than one pattern with the same largest support, choose the pattern with the longest item-sequence; otherwise, just randomly choose one of the patterns with the same longest item-sequences.

Definition 4 A *cold-start user* is a previously-unseen, rarely-doing or rarely-buying user, who still has not owned sufficient item-sequences for the VMSP4MSDT algorithm to generate a representative sequential pattern. A previously-unseen user is defined as that whose item-sequences, $S_j = \emptyset$. A rarely-doing or rarely-buying user is defined as that whose $S_j \neq \emptyset$ but representative sequential pattern, $RS_j = \emptyset$.

3.2 The data preparing phase

Definition 5 A *sequential-label*, L_j, of user j is defined as a sequence of class labels, used to denote a representative sequential pattern with a string format. $L_j =$ "$e_{j1} \cdots e_{ji} \cdots e_{jk}$", where each item $e_{ji} \in E$; $E = \{E_i | E_i$ is an item as well as a class label, where $i = 1..v\}$; and, $L_j \in L$, where L is a set of all non-empty sequential-labels, and $L = \{L_j | j = 1..n\} - \{$""$\}$.

To prepare a training set and a test set from the data stream of items of each data instance, we design the data-prepare function as shown in Function data-prepare(R, α, ω). In this function, Steps 3-4 are critical points. Step 3 calls the VMSP4MSDT algorithm as shown in Algorithm 1 to acquire a representative sequential pattern, RS_j, for each data instance. RS_j is then transformed into a sequential-label, L_j, in Step 4 to label each data instance. Before describing VMSP4MSDT, we first define a set of sequential patterns, $SP_j = \{SP_{ij} | i = 1..m$, where SP_{ij} is defined as a sequential pattern, discovered from $S_j\}$. Next, we define a set of the maximal sequential patterns, $MPj = \{MP_{ij} | i = 1..n$, where MP_{ij} is defined as the maximal sequential pattern, which is a sequential pattern in SP_j not included in any other sequential patterns in $SP_j\}$.

Function `data-prepare(`R, α, ω`)` *used in Algorithm 2*

Input: the data source, $R = \{R_j | R_j = (j, x_j, stream_j, S_j)$, where j: the user id, x_j: the feature vector, $stream_j$: data stream of items, and S_j: the item-sequences$\}$, α: the user-specified upper bound of each sliding window size, and ω: the maximal number of sliding windows per classification lifecycle.

Output: the training set, D, and the test set, *test-set*.

1: **for** *each* R_j *with a stream of items,* $stream_j$, *within each sliding window,* W_i, *where* $i = 1..\omega$ *with size* α *during a user-specified time period* **do**

2: to acquire each S_j from $stream_j$ of R_j within each W_i

3: $RS_j \leftarrow$ call the VMSP4MSDT(S_j) algorithm to get the representative sequential pattern

4: transform RS_j into the sequential-label, L_j

5: **if** $(RS_j \neq \emptyset)$ **then** `/* only sample the instance of non-cold-start users */`

6: choose R_j as the sample instance, $Y_j = (j, x_j, L_j)$

7: partition the sample set, Y, into the training set, D, and the test set, *test-set*

8: **return** D, *test-set*

Algorithm 1 specifies how the VMSP4MSDT algorithm mines a representative sequential pattern for the data instance of each user. It starts by calling the VMSP algorithm to return a set of the maximal sequential patterns, MP_j, for each user j. As there could be several maximal sequential patterns in MP_j, which have then be processed according to Definition 3 to choose one of them to represent MP_j as the representative sequential pattern, RS_j.

Algorithm 1 VMSP4MSDT(S_j) used in the data-prepare function.

Input: the set of item-sequences, S_j.
Output: the representative sequential pattern, RS_j.
1: $MP_j \leftarrow VMSP \triangleright S_j \triangleleft$, where MP_j is a set of the maximal sequential patterns.
2: **if** MP_j *is null* **then**
3: | **return** *null*
else
4: | $Q_j \leftarrow$ choose those maximal sequential patterns with the largest support from MP_j
5: **if** Q_j *has only one pattern* **then**
6: | $RS_j \leftarrow$ the element of Q_j
7: **else if** *all the patterns in* Q_j *have only one longest sequence* **then**
8: | $RS_j \leftarrow$ the pattern with the longest sequence of Q_j
9: **else**
| $RS_j \leftarrow$ randomly choose one of the patterns with the same longest sequence
10: **return** RS_j

To classify the representative sequential patterns of each user j, RS_j, based on one's feature vector, x_j, we next examine whether RS_j could be caused by x_j by the following proposition.

Proposition 1 *Suppose the training data instance of each user* j, $D_j = (j, x_j, S_j, RS_j, L_j)$, *chosen from a sample dataset of non-cold start users;* x_j *is the feature vector;* L_j *is the sequential-label, which denotes the representative sequential pattern,* RS_j, *mined by the VMSP4MSDT algorithm from a set of single-itemed sequences,* S_j. *If* $\exists$ x'_j,*sub-dimensional or equal to* x_j, *and* S_j *is caused by* x'_j, *then* $x'_j \rightarrow RS_j$.

Proof As each D_j is chosen from the data instances of non-cold-start users in the sample dataset, according to Definition 4, such that all the values of x_j, x'_j, S_j and RS_j are not null. In addition, as S_j is caused by x'_j, we can say that $x'_j \rightarrow S_j$. Furthermore, in Algorithm 1, VMSP4MSDT discovers RS_j to represent all the item-sequences in S_j of each user j, so that S_j determines RS_j. Moreover, both of Steps 2-3 and Steps 5-9 can assure that the value of RS_j is only one representative sequential pattern. In other words, there is only one non-null RS_j value associated with each non-null S_j. It is clear that $S_j \rightarrow RS_j$. Therefore, we can conclude by the following inference: $((x'_j \rightarrow S_j) \wedge (S_j \rightarrow RS_j)) \rightarrow (x'_j \rightarrow RS_j)$. □

3.3 The training phase

Definition 6 A *multi-valued and sequential-labeled decision tree*, $T(V, B)$ is a rooted decision tree with multiple degrees, where V is a set of nodes and B is a set of branches. Each internal node of T contains a continuous or a categorical attribute. And, each leaf node of T contains a sequential-label. Each branch of the continuous attribute corresponds to an interval and each branch of the categorical attribute corresponds to a value. As a categorical attribute can be multi-valued, an internal node with a multi-valued attribute has the same data belonging to multiple branches.

Definition 7 A *upper bound of tree branch*, ub, is defined to restrict the size of the tree as a user-specified and degree-restricted parameter, such that $2 \leq Degree(v_i) \leq ub$, where the internal node, $v_i \in V$, contains a continuous attribute, and $Degree(v_i)$ is the degree of v_i.

Example 2 An example of a decision tree for Definition 6 and Definition 7 as shown in Fig. 2 is learned by MSDT from the training set of 15 customers in Table 1. We set the upper bound of the tree branch, ub, to be 10. The tree has 3 internal nodes and 12 leaf nodes. Each branch of the continuous attribute, *income*, corresponds to an interval and each branch of the two categorical attributes, *hobby* and *gender*, corresponds to a value. The attribute, *hobby*, is multi-valued, and the attribute, *gender*, is single-valued.

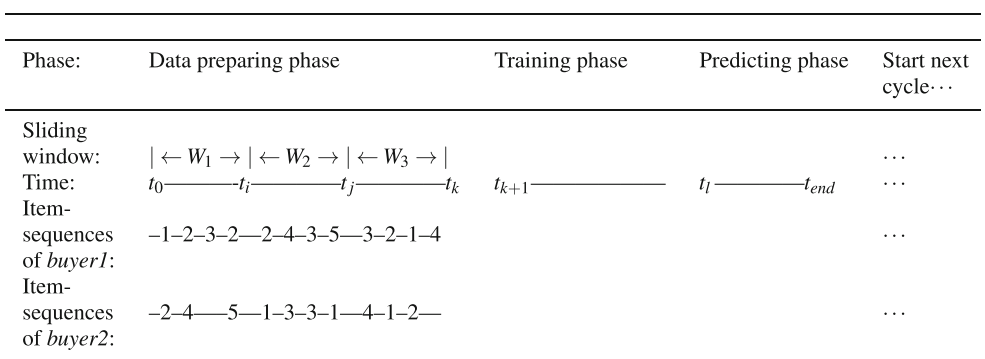


Fig. 1 Two examples for the successive sliding windows over two stream of item-sequences with $\alpha = 4$ during the three phases in the classification lifecycle

While growing the tree, each training instance with the "hobby" attribute being multi-valued is split at the "hobby" internal node into multiple branches.

4 The algorithms

The whole procedure to learn a decision tree classifier to predict a sequential-label is outlined in Algorithm 2 according to the three phases in a classification lifecycle. Initially, at Step 1, a user gives the user-specified parameters to start the lifecycle. Step 2 calls the data-prepare function, which has been explained beforehand in Section 3.2. We further clarify some of the other main steps. At Step 3, the MSDT algorithm with its two auxiliary algorithms as shown in Algorithm 3 through Algorithm 5 are presented in Section 4.1. At Step 6, the predict-data method as shown in Algorithm 6 is described in Section 4.2. Finally, the time complexity analysis of the MSDT algorithm is discussed in Section 4.3.

Algorithm 2 Classify(R, U).

Input: the data source, $R = \{R_j | R_j = (j, x_j, stream_j, S_j)$, where j: the user id, x_j: the feature vector, $stream_j$: data stream of items, and S_j: the item-sequences$\}$. And, the predicted dataset, $U = \{U_i | U_i = (i, x_i, L_i)$, where i: the user id; $i = 1..\theta$; θ: the maximum quantity of predicted instances in the prediction duration; x_i: the feature vector; and L_i: the sequential-label to be predicted$\}$.

Output: a sequential-label set, L, of U.

1: initialize the user-specified parameters, α, ω and θ, where α: the user-specified upper bound of each sliding window size, and ω: the maximal number of sliding windows per classification lifecycle
2: $(D, test\text{-}set) \leftarrow$ data-prepare(R, α, ω), where D: the training set, and *test-set*: the test set
3: *decision-tree* $\leftarrow$ MSDT(D, A), where A: the attribute set of D
4: *ruleset* $\leftarrow$ transform-tree(*decision-tree*)
5: **for** *each data instance to be predicted,* $U_i = (i, x_i, L_i)$*, of each user,* i*, where* $i = 1..\theta$ **do**
6: $\quad L_i$ of $U_i \leftarrow$ predict-data(the root of *decision-tree*, U_i)
7: **return** L *of* U

4.1 The MSDT algorithm

The MSDT algorithm is a process of growing nodes of a decision tree on depth-first recursively. It follows the standard framework adopted by the classical classification methods such as ID3, C4.5, IC, MMC and MMDT. Algorithm 3 explains how MSDT goes.

Algorithm 3 MSDT(D_{CN}, A).

Input: D_{CN}: the training set in the current growing node, and A: the attribute set of D_{CN}.

Output: T: a decision tree.

1: initialize the tree, T, and put the training set, D_{CN}, into the root node of T
2: **if** D_{CN} *is empty* **then**
3: $\quad$ **return** *NULL*
4: $(largeset(D_{CN}), smallset(D_{CN})) \leftarrow$ SPAM4MSDT(all the sequential-labels of D_{CN})
5: **if** CN *satisfies one of the STOP conditions* **then**
6: $\quad$ assigns the leaf node a sequential-label (i.e. a representative sequential pattern) or multiple sequential-labels (i.e. all the maximal sequential patterns) with the maximum support, and return T
7: call the *split-attribute*(A, D_{CN}) to select the best splitting attribute A_i with k branches from A for D_{CN}
8: create a decision node for CN, which contains the attribute A_i
9: **if** A_i *is a categorical attribute* **then**
10: $\quad I \leftarrow$ *partition-discrete-categories*(A_i, k), which partitions the values of the categorical attribute A_i into k categories
else
11: $\quad I \leftarrow$ *partition-continuous-intervals*(A_i, k), which partitions the intervals of the continuous attribute A_i into k intervals
12: grow the tree T for one more level according to the best attribute A_i with k branches of CN
13: **return** T

Initially, MSDT inputs D_{CN} with its attribute set, A, whose values are presorted, for coming analysis. In the above framework, Steps 4-11 are critical points, in which Steps 4-6 determine a leaf node; Steps 7-11 determine the internal node with branches for a tree. We will further explain the framework in the following three subsections. While determining a leaf node, MSDT needs to discover both of frequent and infrequent sequential patterns among sequential-labels in D_{CN} in order to get distinguishable among various sequential-labels. Thus, Steps 4-6, explained in Section 4.1.1, in which calls the SPAM4MSDT algorithm as shown in Algorithm 4 to determine whether the growing node is a leaf node; if not, Steps 7-11, explained in Section 4.1.2, determine the internal node with branches for the tree, in which calls the *split-attribute* function as shown in Algorithm 5.

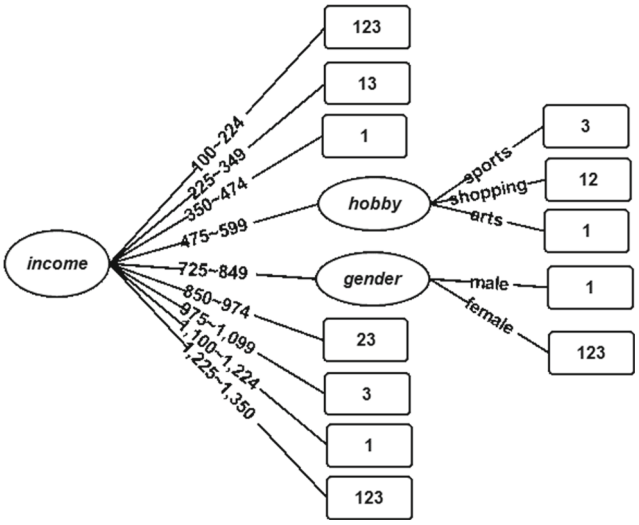


Fig. 2 A multi-valued and sequential-labeled tree built from the training set of Table 1

4.1.1 Determination of leaf node

We first explain as follows how SPAM4MSDT goes in Algorithm 4 and then how MSDT determines a leaf node, which depends on the stop conditions.

Algorithm 4 SPAM4MSDT(L) used in Algorithm 3.

Input: all the sequential-labels of D_{CN}, L.
Output: the set of large sequential-labels with their supports, *largeset*(D_{CN}); and: the set of small sequential-labels with their supports, *smallset*(D_{CN}).

```
1: S ← transform L into a set of item-sequences
2: (SP, ISP) ← SPAM'(S), where SP: the set of
   frequent sequential patterns with their supports,
   and ISP: the set of infrequent sequential patterns
   with their supports
3: if SP is not null then
4:     largeset(D_CN) ← transform SP into a set of
       sequential-labels
5: if ISP is not null then
6:     smallset(D_CN) ← transform ISP into a set of
       sequential-labels
7: return largeset(D_CN), smallset(D_CN)
```

Let $D_{CN} = \{d_1, \cdots, d_i, \cdots, d_r\}$ be the training set, D, in the current growing node, CN, and $\{L_1, \cdots, L_i, \cdots, L_r\}$ be their respective sequential labels. Then, the term, support of L_i, can be defined as: $support(L_i) = L_i.count \,/\, |D_{CN}|$, where $L_i.count$ is the support count of L_i; and $|D_{CN}|$ is the number of records. If the support of L_i is greater than or equal to the user-specified minimum support of the frequent sequential pattern (i.e., *minsup*), L_i is termed large sequential-label. Otherwise, it is termed small sequential-label. Therefore, all of the sequential-labels in D_{CN} can be classified into two sets; *largeset*(D_{CN}) and *smallset*(D_{CN}), where *largeset*(D_{CN}) contains all of the large sequential-labels, also frequent sequential patterns in D_{CN}, and *smallset*(D_{CN}) contains all of the small sequential-labels, not frequent sequential patterns in D_{CN}. However, the algorithms, SPAM and VMSP, do not discover the infrequent sequential patterns in this phase. Especially, the VMSP algorithm removes infrequent items in advance from the sequences during the early stage of the growing phase because they will not appear in any further frequent sequential patterns discovering. This also get rid of the source of the infrequent sequential patterns and led to discover the infrequent sequential patterns impossible. Fortunately, SPAM does not remove infrequent items during the tree-growing phase. Additionally, SPAM can only calculate the support count rather than support of each L_i, we revised it to enable calculation of the support first. Thus, we have revised SPAM into the revision, SPAM', as shown in Step 2 to get both of the frequent and the infrequent sequential patterns. Finally, SPAM4MSDT transforms both the sets of patterns into *largeset*(D_{CN}) and *smallset*(D_{CN}).

Back to the stop conditions in Algorithm 3 to determine a leaf node, we define *difference*(D_{CN}) =

$$\min\{support(L_i)|L_i \in largeset(D_{CN})\} - \max\{support(L_j)|L_j \in smallset(D_{CN})\} \quad (1)$$

If difference of D_{CN} is greater than or equal to a user-specified minimum difference (termed *mindiff*), then D_{CN} is termed clear node. Otherwise, it is termed unclear node. D_{CN} continues to grow, until one of the following stop conditions is fulfilled:

(1) If node D_{CN} is clear, then the MSDT algorithm assigns the sequential-label whose maximal sequential pattern with the maximum support in *largeset*(D_{CN}) as its result class label. Furthermore, if there are more than one maximal sequential pattern with the same maximum support, then MSDT assigns all the sequential-labels with the same maximum support to the node D_{CN}.
(2) Otherwise, if all the attributes have been used up in the path from root down to D_{CN}, and the size of D_{CN} is larger than or equal to a user-specified minimum quantity (termed *minqty*), then (2.1) If *largeset*(D_{CN}) is not empty, then the MSDT algorithm

assigns the sequential-label whose maximal sequential pattern with the maximum support in $largeset(D_{CN})$ as its result class label. Furthermore, if there are more than one maximal sequential pattern with the same maximum support, then MSDT assigns all the sequential-labels with the same maximum support to the node D_{CN}. (2.2) If $largeset(D_{CN})$ is empty, then the MSDT algorithm assigns the sequential-label whose maximal sequential pattern with the maximum support in $smallset(D_{CN})$ as its result class label. Furthermore, if there are more than one maximal sequential pattern with the same maximum support, then MSDT assigns all the sequential-labels with the same maximum support to the node D_{CN}.

(3) Otherwise, if all the attributes have been used up, and the number of data instances is less than *minqty*, then drop off D_{CN}.

Example 3 Let $minsup = 45\%$, and $mindiff = 15\%$. D_{CN} has six data instances and their sequential-labels are "243", "321", "315", "31", "243" and "542" respectively. The SPAM4MSDT algorithm calculates the supports of all the frequent sub-sequence candidates on iterations from 1-sequence until having gotten frequent sequential patterns. At the final iteration, the supports of all the candidates are $< 21 >: 1/6, < 23 >: 2/6, < 24 >: 2/6, < 31 >: 3/6, < 32 >: 1/6, < 42 >: 1/6$ and $< 43 >: 2/6$. As $minsup = 45\%$, we get $largeset(D_{CN}) =$ {"31"} and $smallset(D_{CN}) =$ {"21", "23", "24", "32", "42", "43"}. According to (1), we get $difference(D_{CN}) = 50\% - 33.33\% = 16.67\%$. Since $difference(D_{CN}) > mindiff$, D_{CN} is clear. Finally, as "31" is the maximal sequential pattern with the maximum support in $largeset(D_{CN})$, its result class label is "31".

4.1.2 Determining internal node with branches

To grow internal nodes, MSDT tests the goodness of spitting for each attribute with a measure. Thus, we have tried to design two measures, *sequentialGainRatio* and *sequential-weighted-similarity*. Both of the measures can handle discrete, continuous and multi-valued attributes. We modified two well-known measures. *SequentialGainRatio* has modified the *gain ratio* measure [24], and *sequential-weighted-similarity* has modified the *weighted-similarity* measure [5]. However, the accuracies of the MSDT algorithm based on *sequentialGainRatio* is averagely better than the accuracies of MSDT based on *sequential-weighted-similarity* (will be discussed further in Section 6.1.2 and Section 6.1.4). Therefore, we adopt *sequentialGainRatio* as our measuring strategy. We describe *sequentialGainRatio* first and *sequential-weighted-similarity* next.

To determine the internal node with branches, Step 7 in Algorithm 3 focuses on the split-attribute function. Algorithm 5 explains how split-attribute selects the best splitting attribute with branches. Step 4 and Step 6 calculate the *sequentialGainRatio* measure separately. It depends on the type of the splitting attribute. *SequentialGainRatio* is defined by the following equation.

Algorithm 5 Function split-attribute(A, D_{CN}) used in Algorithm 3.

Input: Data: A: the attribute set of D_{CN}, and D_{CN}: the training set in the current growing node.
Output: A_i: the best-split attribute, and brn: the number of branches.

```
1: for each attribute A_i ∈ A do
2:     if A_i is a categorical attribute then
3:         Partition D_CN into k categories according to A_i
4:         Compute sequentialGainRatio(D_CN, A_i, k)
       else
5:         Partition D_CN into 2 to ub intervals according to A_i
6:         Compute max_{2≤k≤ub} sequentialGainRatio(D_CN, A_i, k)
7: return the best-split attribute, A_i, and the number of branches, brn, that can get the largest sequentialGainRatio
```

$$sequentialGainRatio(D_{CN}, A_i, k) = sInfoGain(D_{CN}, A_i, k)/I(D_{CN}, A_i, k), \quad (2)$$

where $sInfoGain(D_{CN}, A_i, k)$ and $I(D_{CN}, A_i, k)$ is the information gain and the entropy of the splitting of attribute A_i into k intervals on node CN. The information gain in (2) is defined as:

$$sInfoGain(D_{CN}, A_i, k) = I(D_{CN}) - E(D_{CN}, A_i, k), \quad (3)$$

where $I(D_{CN}) = \sum_{j=1}^{m} -p_j \log p_j$ is the entropy of D_{CN}, and $p_j = |p_j|/|D_{CN}|$ is the percentage of a sequential-label L_j in D_{CN}. $E(D_{CN}, A_i, k)$ in (3) is defined as:

$$E(D_{CN}, A_i, k) = \sum_{j=1}^{k} -p_{ij} I(D_{CN}^{A_i,k}(j)), \quad (4)$$

where $P_{ij} = n_j / \sum_{j=1}^{k} n_j = n_j/n'$ is the percentage of a sequential-label L_j in D_{CN} after splitting the

attribute A_i; $D_{CN}^{A_i,k}(j)$ denotes the j-th sub-set based on A_i after partitioning D_{CN} into k sub-sets; and the entropy, $I(D_{CN}^{A_i,k}(j))$ is $\sum_{t=1}^{m} -p_t \log p_t$, where the percentage of a sequential-label L_t in $I(D_{CN}^{A_i,k}(j))$ is $p_t = |p_t|/n_j$. Finally, we define $I(D_{CN}, A_i, k)$ as:

$$I(D_{CN}, A_i, k) = \sum_{j=1}^{k} -p_{ij} \log p_{ij}, \tag{5}$$

where the percentage, p_{ij} is the same as p_{ij} of (4).

Example 4 Let us demonstrate the choice of the best splitting attribute. If Table 2 represents the data stored in node, CN, which has 10 training instances and two classifying attributes; gender and hobby, then the attribute gender is first considered and the *sequentialGainRatio* of the attribute, gender, is computed as: $sInfoGain(D_{CN}, A_i, k) = 2.0253 - 1.6296 = 0.3957$, $I(D_{CN}, A_i, k) = 0.67301$, and thus $sequentialGainRatio(D_{CN}, A_i, k) = 0.5880$. The *sequentialGainRatio* of the attribute, hobby, is computed as 0.2608. Since the *sequentialGainRatio* of the attribute, gender, is larger than that of the attribute, hobby, the attribute, gender, is selected as the next splitting attribute.

Next, we define *sequential-weighted-similarity* as a measure of the splitting of attribute A_i with k branches on node CN as:

$$\begin{aligned} &sequential\text{-}weighted\text{-}similarity(D_{CN}, A_i, k) \\ &= \frac{\sum_{p=1}^{k} nodeSimilarity(L^p) \times n_p}{n'}, \end{aligned} \tag{6}$$

where the similarity of a child node L^p, *nodeSimilarity* (L^p), is the similarity of a node calculated based on the similarity measure between two sequential-labels based on the Jaro-Winkler metric [17, 39] is defined as:

$$\begin{aligned} &nodeSimilarity(L^p) \\ &= \frac{\sum_{i=1}^{r} C_2^{coun_i} + \sum_{i<j} count_i \times count_j \times JaroWinkler(SL_i, SL_j)}{m(m-1)/2}, \end{aligned} \tag{7}$$

where $|L^p| = r, |L| = r, m \neq 1, C_1^{count_i} = count_i$ and $C_1^{count_j} = count_j$.

Example 5 Suppose $L = \{L_1, L_2, \cdots, L_7\}$, where $L_1 = L_2 = SL_1 =$ "12", $L_3 = L4 = L5 = SL_2 =$ "13" and $L_6 = L_7 = SL_3 =$ "123". Therefore, $m = 7$ and $r = 3, count_1 = 2, count_2 = 3$ and $count_3 = 2$. We get that $JaroWinkler(SL_1, SL_2) = 0.7$, $JaroWinkler(SL_1, SL_3) = 0.9111$ and $JaroWinkler(SL_2, SL_3) = 0.9$. Using these values in (7) yields $nodeSimilarity(L) = 0.964$.

If the attribute is continuous, both of the functions, *split-attribute* and *partition-continuous-intervals*, require that the dataset D_{CN} be sorted beforehand in each internal node. This requires much sorting time. For each internal node, CN, sorting is executed $O(r)$ times, where r is the number of continuous attributes. To address this problem, we presort and index continuous attributes only once at the initial state of the MSDT algorithm. At the same time, we keep the sorted and indexed data columns in data cache stored in main memory during the tree growth phase. The indexing method uses B-tree, which allows searches in logarithmic time. Thus, this avoids lengthy sorting and re-sorting at each node.

Table 2 An example with 10 training instances and 2 classifying attributes

User id	Gender	Hobby	Sequential-label
1	2	02	321
2	2	03,04	315
3	1	02	31
4	2	02,03	542
5	2	02,03,04	3
6	1	02,03,04	321
7	2	02,03,04	243
8	1	02,03	23
9	2	02	5
10	1	02	243

4.2 The predict-data algorithm

Predict-data is designed to predict the sequential-label of a data instance as shown in Algorithm 6. It predicts for each instance by traversing the decision tree: starting from the root node, finding a path to the leaf node and using the sequential-label of the leaf node as the prediction result. When an instance has a multi-valued attribute, the prediction may reach several leaf nodes. MSDT takes the union of all of these sequential-labels as the prediction result. In other words, multiple sequential-labels can be the prediction result for each instance. It can predict the result to be multiple sequential-labels (namely *result1*) as well as a single sequential-label (namely *result2*). To get *result2*, Steps 7-8 first call the VMSP4MSDT algorithm to return a representative sequential pattern by transforming *result1* into a set of item-sequences as the input of VMSP4MSDT. And then, Steps 9-10 transform the representative sequential pattern into a single sequential-label as the prediction result;

or readers may choose both of result1 and result2 as the prediction results to compare the accuracies each other, which will be discussed in the experimental section.

Algorithm 6 predict-data(u, *test-data*) used in Algorithm 2.

Input: u is a current node of the decision tree, and *test-data* is the data instance to be predicted.
Output: $result$ is the prediction result.

```
1: if u is a leaf node then
2:     return the sequential-label of u
3: result1 ← ∅; result2 ← ∅
4: for each child v of node u do
5:     if the condition, arc(u, v), is satisfied by
       test-data then
6:         result1 ← result1 ∪ result1 of
           predict-data(v, test-data)
7: sequences ← transforming result1 into a set of
   item-sequences
8: RS ← VMSP4MSDT(sequences)
9: result2 ← transforming RS into a single
   sequential-label
10: result ← result2 or both of result1 and result2
    depends on the requirement of readers
11: return result
```

4.3 Time complexity of MSDT

We first examine the time complexity of MSDT by the following lemma and then discuss the time complexity.

Lemma 1 *Let there be m attributes, n training instances, v events, α: the maximal length of all the sequences, and ub: the upper bound of tree branch, the MSDT algorithm grows a multi-valued and sequential-labeled decision tree in* $O(\alpha mn^2 + ubkv^\alpha m^2 n)$ *time.*

Proof Initially, MSDT is given the training set, D, with the values of continuous attributes sorted for analysis. The time complexity of the tree induction is $O(mn)$;the sorting is $O(n \log n)$ and executed only once. The time complexities of the following functions: *SPAM4MSDT*(sequential-labels of D) is $O(\alpha n)$; split-attribute(A, D_{CN}) is $O(m(k + ub + ub((k+1)v^\alpha + k)))$ using sequentialGainRatio(D_{CN}, A_i, k), $O((k + 1)v^\alpha + k))$, to choose the best attribute, A_i, where $\exists$ A_i with the most branches, k;no matter whether A_i is categorical or continuous, both of the functions, partition-discrete-categories and partition-continuous-intervals, are $O(k)$, which partition the intervals of the continuous attribute A_i into k intervals. As for assigning label to represent a leaf node, MSDT assigns a sequential-label (i.e. a representative sequential pattern) or multiple sequential-labels with the same maximum support (i.e. multiple maximal sequential patterns) to represent the leaf node, $O(maximum(l, s))$, where l is the count of the *largeset*, and s is the count of the *smallset*. Finally, the time complexity of MSDT is $O(n \lg n + mn(\alpha n + maximum(l, s) + m(k + ub + ub((k + 1)v^\alpha + k)) + k)) = O(\alpha mn^2 + ubkv^\alpha m^2 n)$. □

Although v^α in $O(\alpha mn^2 + ubkv^\alpha m^2 n)$ seems to be an influencing factor to the time complexity, it can still be reduced by setting both of the upper bound values of v and α under two respective application settings of MSDT. As for v part, the application setting is to sample a training-set from non-cold-start users who prefer similar v items, and let $v = p$. This results in a precondition that v has the upper bound value, p, to keep it from going too big. As v is also seen as the number of classes, we can further reduce the value of v using concept hierarchy climbing by merging more sub-classes into less super classes. As for α part, the application setting is to use the technology of sliding window to constrain the upper bound value of α, which also means constraining the maximum length of all the sequences. The setting is applicable to analyzing spatial-temporal or state-temporal item-sequences. Fortunately, the lengths of such item-sequences always have an upper bound requirement in such an application using a user-specified size-constrained sliding window. For example, suppose $v = 10$ and $\alpha = 6$, such that a training-set with features and consumed sequences is sampled from non-cold-start users who prefer 10 airports similarly; and each item-sequence is a flight itinerary with six airports at most during the most recent 90 days. As the values of v and α will not be too big under our application settings, v^α is not the influencing factor.

5 Experimental setup

In this section, we first present the experimental questions. Next, we describe the datasets and design the experiments. Finally, we discuss the evaluation measures used in the experiments.

5.1 Experimental questions and strategy

Before comparing MSDT with some baseline algorithms, it raises the other two experimental questions. One is how to choose benchmarking training-sets from some candidate datasets with feasible size for comparisons. The other is how to acquire the optimal hyperparameters of both of MSDT and the baseline algorithms on the benchmarking training-sets that achieve their own best accuracies. To solve these questions, we plan three experiments: Experiment I and

Experiment II at the pretraining stage as well as Experiment III at the training stage. Further, Experiment I has two sub-experiments: Exp. I.1 and Exp. I.2; and, Experiment III has two sub-experiments: Exp. III.1 and Exp. III.2.

The overall strategy of the three experiments are described as follows. At the pretraining stage, to control the influence of some hyperparameters, in Experiment I, we initially test the performances of MSDT on two small datasets based on all the reasonable configurations of the hyperparameters, from which we will get some control variable candidates for Experiment II. Inevitably, training-sets from the small datasets cause a model under-fitting problem. The problem occurs because the training-set is not large enough; so that the model is too simple to learn the true structure of the data [35]. Therefore, in Experiment II, we start by looking for a benchmarking training-set with a feasible size from a large dataset, validated by checking whether it is large enough to reduce the model under-fitting problem. Meanwhile, we also validate whether MSDT on the large training-set based on those control variable candidates really reduce the problem. The way of the validation is to operate those candidates to examine whether the trend of the accuracies based on the large training-set vary with them. During the validation, as we can get the control variables from those candidates, we can operate the variables to acquire the optimal hyperparameter configuration of MSDT that achieves the best accuracy. At the train stage, in Experiment III, using the same large training-set and test-set, we compare the performances of MSDT based on the optimal hyperparameter configuration with the performances of some baseline algorithms based on their own optimal ones.

5.2 Datasets

For the three main experiments, we have selected a total of 4 datasets. A summary of the datasets and their properties is shown in Table 3. All the datasets, further grouped into five database archive files with a metadata description used in the corresponding experiments can be downloaded from our data repository over the *Harvard Dataverse* repository at https://dataverse.harvard.edu/privateurl.xhtml?token=a8b969ae-96da-483a-b0ef-21c6b5f29cfd. A training set and a test set were generated from the dataset of each experiment. The "sequences" attribute of all the datasets contain a set of item-sequences, represented as itineraries of a tourist. And, each item of an itinerary is a scenic spot or district, numbered from 1 to 5. Thus, the value of the sequential-label attribute could be a sequence of 1 to 5.

The *Tourist* dataset is one of the two small real-life datasets. It contains 100 offline buyers who registered their profiles in any online websites first and then consumed tour itinerary services in brick-and-mortar stores offline. The item-sequences were surveyed and sampled from their purchased itinerary services across five districts of Taiwan during the year of 2014. The number of various sequential-labels is 40. To reduce the bias caused by selecting the attributes un-related with sequential-labels, we measure the correlation between the predictor attributes (single-valued and multi-valued) and sequential-label using *sequential-weighted-similarity*, mentioned in Section 4.1.2. *Sequential-weighted-similarity* is between 0 and 1. If it is more than 0.5, the correlation is correlative. Since all the attributes are greater than 0.5, we choose all of them as the candidate attributes for the MSDT algorithm.

The *CDNow-RFM* dataset is the other of the two small real-life datasets. Its features and the sequences are extracted and summarized from the *CDNow_sample* dataset [9] containing 2,357 online buyers about their purchasing records from Jan. 1997 to June 1998, which can be accessed at http://www.brucehardie.com/datasets/. The number of various sequential-labels is 12 when the windows size = 2; and it is 33 when the windows size = 3. After calculating *sequential-weighted-similarity* of each attribute, three non-correlative attributes are removed.

The *msdt2-multi-valued* dataset is a group of large multi-valued and sequential-labeled datasets of tourists. To learn classifiers from training instances with item-sequences collected within sliding windows, we set the sliding window sizes, α = 3..5 for three classification lifecycles. Therefore, the value of the "sequences" attribute is a set of item-sequences with lengths from 3 to 5, which correspond three most recent sliding windows respectively with the incrementally-added sizes. We thus have three large datasets of 3-sequence, 4-sequence and 5-sequence. The item-sequences of each record are generated by the combination of the five functions, which can be accessed at our data repository over the *Harvard Dataverse* repository mentioned above. Besides, the number of various sequential-labels is 10 while $\alpha = 3$; 13 while $\alpha = 4$; and 10 while $\alpha = 5$.

Likewise, the *msdt2-single-valued dataset* has three large single-valued and sequential-labeled datasets with α = 3..5. Since these baseline algorithms cannot handle data with multi-valued features, we remove all the multi-valued features of the large dataset in Experiment III.1. Besides, we should further revise the five functions mentioned in the *msdt2-multi-valued* dataset because the multi-valued features constitute some of the conditions of those functions. The revised functions can be accessed at our data repository over the *Harvard Dataverse* repository mentioned above. Moreover, the number of various sequential-labels is 47 while $\alpha = 3$; 243 while $\alpha = 4$; and 878 while $\alpha = 5$.

5.3 Design of experiments

In this section, we will describe the experimental methodology and the hyperparameters of each experiment respectively. All the experiments were conducted on an Intel Core i7-4700HQ cpu-2.40 GHz computer with 8 gigabytes of main memory. All the algorithms, written in *Java*, were processed according to the steps as shown in Algorithm 2 in each experiment. Each of the results are compared in each experiment by fixing four of the following five parameters: size of training set = 90 for Experiment I.1, size of training set = 235 or 236 on the size of sliding window for Experiment I.2, and size of training set = 10,000 for Experiment II and Experiment III.2, $minsup = 45\%$, $mindiff = 15\%$, $minqty = 10$ and $ub = 6$, in order to analyze their influences. Both of Experiment II and III use 5,000 records as the test set. The values of the parameters for each experiment are specified as follows: Size of training set is increased from 6,000 to 14,000 in increments of 2,000 only for Experiment II; *minsup* is increased from 35% to 55% in increments of 5%; *mindiff* is increased from 5% to 25% in increments of 5%; *minqty* is increased from 2 to 18 in increments of 4 and *ub* is increased from 2 to 10 in increments of 2. Next, we describe the three experiments in detail respectively as follows.

5.3.1 Experiment I

This experiment is designed to initially test the performances of MSDT on the two small datasets based on all the reasonable configurations of the hyperparameters; next, we select a baseline dataset from the two datasets; finally, through examining the performances of MSDT on the baseline dataset by controlling one specific parameter and fixing the other parameters, we could get some control variable candidates for Experiment II. As both the datasets are small, this may result in bias in the estimate. Thus, we adopt 10-fold cross-validation, recommended by [19], to evaluate MSDT on both the datasets in two sub-experiments respectively. As shown in Table 3, Experiment I.1 evaluates on the *Tourist* dataset, and Experiment I.2 evaluates on the *CDNow-RFM* dataset.

By the way, to initialize the user-specified parameters of the classification lifecycle, we set the sliding windows size, $\alpha = 5$ for Experiment I.1, and $\alpha = 2..3$ for Experiment I.2; the maximal number of sliding windows, $\omega = 3$, for both of the experiments; and the maximum quantity of predicted instances in a prediction duration, $\theta = 10$ for Experiment I.1, and $\theta = 235$ if $\alpha = 2$, $\theta = 236$ if $\alpha = 3$ for Experiment I.2.

5.3.2 Experiment II

This experiment is designed to acquire the optimal hyperparameters of MSDT. Besides using the same hyperparameters from Experiment I, Step 1 evaluates MSDT on the *msdt2-multi-valued* dataset with all the other reasonable configuration of hyperparameters. Step 2 chooses the selecting measure for splitting attributes by comparing the performances between MSDT based on the two measures. Step 3 looks for a feasible size of training-sets with different α to reduce the model underfitting problem occurred in Experiment I. Step 4 validates whether MSDT on the large enough training-sets based on the control variable candidates from Experiment I really reduce the problem. Otherwise, go back to Step 3. Step 5 finally gets the control variables from those candidates and operates those variables to acquire the optimal hyperparameters of MSDT.

By the way, to initialize the user-specified parameters of the classification lifecycle, we set the initial sliding window size, $\alpha = 3$; the maximal number of sliding windows, $\omega = 3$; and the maximum quantity of predicted instances in a prediction duration, $\theta = 5,000$.

5.3.3 Experiment III

To check the superiority of MSDT, we found some well-known multi-labeled classification methods as baseline methods to compare with MSDT. As there are two types of

Table 3 A summary of the datasets and their properties

Dataset	N	Input attr.	The sequences attr.	The sequential-label attr.	α	Experiment
Tourist	100	11	1	1	5	Exp. I.1
CDNow-RFM	2,357	7	1	1	2..3	Exp. I.2
msdt2-multi-valued	20,000	9	1	1	3..5	Exp. II, Exp. III.1
msdt2-single-valued	20,000	6	1	1	3..5	Exp. III.2

multi-labeled methods, mentioned in Section 2.2, we will conduct this experiment in the following sub-experiments respectively.

Experiment III.1 is to compare MSDT with the multi-valued and multi-labeled methods, MMC, MMDT and MMD+MMDT, on handling the same datasets as Experiment II did. As MSDT and those baseline methods have the same hyperparameter variables, we can acquire their respective optimal accuracies by the same way as Experiment II did. Besides, to imitate MMD+MMDT, we plan to check whether combining a discretization method with MSDT could improve the accuracy of MSDT. After knowing the combination of MMD with MSDT is not feasible, we combine MSDT with the other well-known heuristic multi-intervals discretization method (namely MID here) by [10] instead. Therefore, we will compare the performances of MSDT with MID+MSDT.

Experiment III.2 is to compare MSDT with the following single-valued and multi-labeled methods including: (a) Algorithm adaptation approaches: CART-ML, ExtraTree-ML, RandomForest-ML, ExtraTrees-ML, ML-KNN [30, 43], MLTSVM [6] and iSOUP-Tree [20]. (b) The methods of deep neural network (DNN) with multi-layer perceptron (MLP): The Scikit-learn Python package has implemented the MLPClassifier method (namely DNN-MLP-ML here) [30], which extends DNN with MLP [26]. (c) Problem transformation approaches: RAkEL [37], CDT [27] and MajorityVotingClassifier (namely MVoting here) [33, 34]. In Experiment III.2, MSDT is conducted by the same way as Experiment III.1 did except different datasets handled by both of the experiments.

5.4 The evaluation measures

[16] stated that classification and prediction methods can be compared and evaluated according to six criteria: predictive accuracy, speed, robustness, scalability, interpretability and goodness of rules. This paper focuses on predictive accuracy (represented as accuracy), speed of growing a tree (represented as the execution time) and the number of rules. To evaluate the accuracy of the prediction for a test-set, the accuracy of that for each test instance must be determined first. Suppose each instance has a pair of L_i and L_j, where L_i is the predicted sequential-label, and L_j is the actual sequential-label. It is not appropriate to assign an accuracy of 1 or 0 if L_i and L_j are similar but not totally the same or different. Therefore, we use the Jaro-Winkler metric [17, 39] to measure the similarity between L_i and L_j to determine the accuracy. The predict-data algorithm as mentioned above can predict the result of each instance to be multiple sequential-labels as well as a single sequential-label. The accuracy of the former, namely AccuracyF, is calculated by averaging all the accuracies of the predicted sequential-labels, each of which is measured with the actual sequential-label of the instance by the Jaro-Winkler metric. The accuracy of the latter, namely AccuracyL, is measured between the pair of L_i and L_j of the instance by the Jaro-Winkler metric. Finally, it calculates the average accuracy for all of the test instances to determine the accuracy of the test set.

6 Experimental results and discussion

We will describe the results of the three experiments according to the pretraining stage in Section 6.1, the formal training stage in Section 6.2 and the discussion in Section 6.3 respectively.

6.1 Results in the pretraining stage

6.1.1 Examination on different sizes of training set

We examine whether the performances of MSDT vary with the sizes of training set. Table 4 shows that the time will increase and the number of rules has an ascending tendency as the size of the set increases. And, both of AccuracyF and AccuracyL have ascending tendencies, which increase first and then remain steady as the size increases. However, we should not go too far in this direction. Otherwise, when the size is increased 2,000 or 4,000 from 10,000, we can see the number of rules is 1.7 times of the latter; but all the accuracies increase first decline then. Besides, we should consider the error rate of accuracy. It is a non-coverage rate of rules in the test-set. It means the percentage of test data in the test-set are not covered and predicted by the rules. Notice that when the size of the training set is 90, the accuracies and the number of rules are much lower than the other training sets with larger size, and the error rates of accuracy are higher than those with larger size. This situation is known as model under-fitting. As the number of rules increases, the tree will have higher accuracies, therefore, this reduces the model under-fitting. As the accuracies remain high enough and steady during the size from 6,000 to 14,000, the training-set with size 10,000 has the lowest error rate of accuracy. Therefore, we choose the benchmarking training set with size = 10,000 for further comparisons.

6.1.2 Examination on data behavior between the datasets

We will describe the examination in the following three steps.

First. Select the baseline small dataset We review the results of Experiment I as shown in Table 5. It shows that the MSDT algorithm with selecting measures of

Table 4 The performances of MSDT vary with the sizes of training sets using the parameters, minsup = 45%, mindiff = 15%, minqty = 10, ub = 6 and $\alpha = 3..5$

Training set size	AccuracyF	AccuracyL	Time(ms.)	Number of rules	Error rate of accuracy
90*[1]	52.09%	50.25%	224	29.3	27.00%
90*[2]	64.38%	58.81%	120	23.7	10.19%*[3]
1,000	67.36%	66.88%	1,235	112.3	8.72%
4,000	67.66%	67.02%	12,916	445.0	5.69%
6,000	68.33%	67.94%	27,576	558.0	5.57%
8,000	68.16%	67.61%	43,544	677.3	4.46%
10,000	68.15%	67.59%	60,999	667.3	2.71%
12,000	68.21%	67.82%	149,876	1,126.0	4.62%
14,000	67.96%	67.63%	182,654	1,129.3	3.91%

(1) The training set size, 90, is sampling from the small dataset, *Tourist*. (2) The training set size, 90, is sampling from the large dataset. (3) Error rate of accuracy = average number of null-labeled records / size of test set = 509.3 / 5,000 = 10.19%

attribute, datasets, whether the dataset include non-correlative attributes and the four experimenting parameters produce different accuracies, time, number of rules and error rate of accuracies. In Table 5, you can see all these results except MSDT based on *sequentialGainRatio* handling the *CDNow-RFM* dataset with correlative attribute because it has failed to build a classifier.

Table 5 shows that all the average accuracies of MSDT handling *CDNow-RFM* are better than those of MSDT handling *Tourist*; however, the error rate of accuracies of the former are all above 70% and worse than the latter. During predicting, this results in many records of all the test-sets of *CDNow-RFM* being null values. Therefore, we take the *Tourist* dataset as the only baseline small dataset (hereinafter referred to as the small dataset), with which for the following Experiment II and III to compare. Furthermore, based on the *Tourist* dataset, Table 5 shows that the average accuracy of MSDT based on *sequentialGainRatio* is better than that of MSDT based on *sequential-weighted-similarity*. Therefore, we only use the *sequentialGainRatio* measure rather than the *sequential-weighted-similarity* measure in all the following

Table 5 Testing the performances of MSDT on two small datasets in Experiment I to get a baseline dataset for comparing with the large dataset in Experiment II

Measure for selecting the attribute	Dataset	Include non-correlative attributes?	Average accuracy	Average baseline accuracy	Time(ms.)	Number of rules	Error rate of accuracy
sequential GainRatio	Tourist	No	51.42%	11.00%	62.5	29.0	27.89%
sequential-weighted-similarity	Tourist	No	49.90%	11.00%	52.8	58.8	20.90%
sequential-weighted-similarity	CDNow-RFM	No	72.38%	41.00%	220.6	68.4	72.64%
sequential GainRatio	CDNow-RFM	Yes	52.91%	41.00%	570.8	81.5	88.66%
sequential-weighted-similarity	CDNow-RFM	Yes	64.85%	41.00%	1,705.4	227.6	73.96%

Each performance is calculated averagely on all the results of MSDT by fixing three of the following four parameters: *minsup* = 45%, *mindiff* = 15%, *minqty* = 10 and *ub* = 6, under their ranges: *minsup* = {35%..55% in increments of 5%}, *mindiff* = {5%..25% in increments of 5%}, *minqty* ={2..18 in increments of 4}, and *ub* = {2..10 in increments of 2}

analyses of the experimental results. This has explained the mention in Section 4.1.2. Besides, Table 5 shows that the average accuracy of all the experimental results are better than all their average baseline accuracies. Here, we define that the baseline accuracy means how often we would be correct if we always predict the majority class, the so-called accuracy paradox [45].

Second. Select the control variable candidates Table 6 shows the performances vary with the four parameters on both of the small dataset and the large dataset. We have found two control variable candidates, i.e. hyperparameters, *mindiff* and *minsup*, of MSDT, while examining their data behavior. To get a more detailed understanding, we explain them with Fig. 3 as follows.

As for the candidate, *mindiff*, Fig. 3a shows that both of the time of the two datasets will increase as *mindiff* increases; Fig. 3b shows that both of the number of rules of the two datasets will also increase as *mindiff* increases. However, Fig. 3c shows that both AccuracyF and AccuracyL of the small dataset decrease first as *mindiff* increases (named accuracy-reduction effect); and then, they increase as *mindiff* increases and exceeds the turning points, 15% for AccuracyF and 10% for AccuracyL (named accuracy-raise effect). Different from the small dataset, Fig. 3c shows that both of AccuracyF and AccuracyL of the large dataset have ascending tendencies as *mindiff* increases.

As for the candidate, *minsup*, Fig. 3d shows that both of time of the two datasets have an ascending tendency as *minsup* increases; Fig. 3e shows that both of the number of rules of the two datasets have an ascending tendency too. However, Fig. 3f shows that both AccuracyF and AccuracyL decrease first as *minsup* increases (the accuracy-reduction effect); and then, they increase after the first turning point, 40%, as *minsup* increases (the accuracy-raise effect). Finally, they decrease as *minsup* increases until reaching the second turning point, 50% (the accuracy-reduction effect). Different from the small dataset, Fig. 3f shows that both of AccuracyF and AccuracyL of the large dataset have ascending tendencies as *minsup* increases.

Those accuracy-reduction effects of both the candidates reflect a model under-fitting problem because all the training-sets are small.

Third. Reducing the under-fitting problem We further explain why MSDT handling the large enough training-sets based on those control variable candidates can reduce the under-fitting problem as follows. As shown in Fig. 3c and f, while the size of training set is large enough, the tree generates much more leaf nodes (rules) with more data quantity and useful generalized information or higher support than that of the small dataset during the increasing of *mindiff* or *minsup*. As for *mindiff*, the large dataset one increases 307.92 rules on average; but the small dataset one increases only 1.03 rules on average. As for *minsup*, the former increases 193.9 rules on average; but the latter

Table 6 The comparisons of the performances between MSDT on the small dataset and MSDT on the large dataset among different parameters

(a) minsup=45%, minqty=10 and ub=6

mindiff	accuracy of small dataset	accuracy of large dataset	time of small dataset	time of large dataset	number of rules of small dataset	number of rules of large dataset
5%	52.77%	62.89%	215.5	22,393.0	28.0	136.7
10%	51.11%	67.74%	221.9	36,438.0	28.9	338.7
15%	51.17%	67.87%	223.7	62,466.0	29.3	667.3
20%	53.34%	68.33%	232.9	145,875.0	32.0	1,144.7
25%	53.34%	68.58%	240.6	149,021.0	32.1	1,368.7
Average	52.35%	67.08%	226.9	83,238.6	30.1	731.2

(b) mindiff=15%, minqty=10 and ub=6

minsup	accuracy of small dataset	accuracy of large dataset	time of small dataset	time of large dataset	number of rules of small dataset	number of rules of large dataset
35%	51.35%	66.17%	192.2	46,109.0	23.3	436.7
40%	49.60%	66.81%	210.9	42,716.0	26.6	429.3
45%	51.17%	67.87%	223.7	62,466.0	29.3	667.3
50%	51.76%	68.15%	232.8	81,805.0	29.8	878.3
55%	46.04%	68.38%	267.3	192,893.0	22.9	1,210.7
Average	49.99%	67.48%	225.4	85,197.8	26.4	724.5

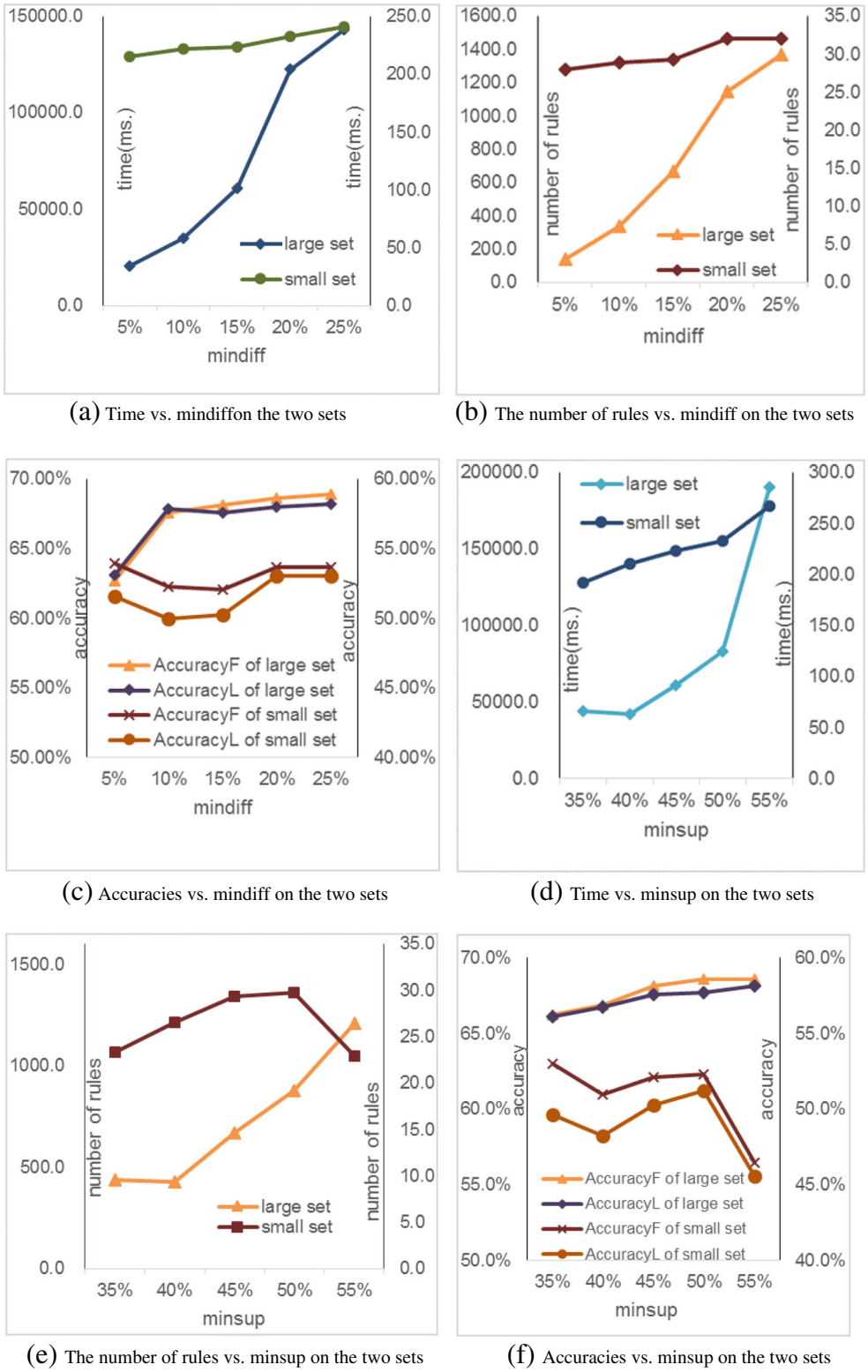
(c) minsup=45%, mindiff=15% and minqty=10

ub	accuracy of small dataset	accuracy of large dataset	time of small dataset	time of large dataset	number of rules of small dataset	number of rules of large dataset
2	55.12%	68.01%	233.0	43,570.0	32.5	703.7
4	52.21%	67.41%	235.9	76,734.0	30.9	748.3
6	51.17%	67.87%	223.7	62,466.0	29.3	667.3
8	51.53%	67.62%	217.2	74,201.0	29.6	1,037.0
10	50.78%	67.51%	216.3	58,570.0	29.2	1,091.7
Average	52.16%	67.68%	225.2	63,108.2	30.3	849.6
(e) Total Avg.	51.42%	67.53%	226.1	74273.9	29.0	743.2

(d) minsup=45%, mindiff=15% and ub=6

minqty	accuracy of small dataset	accuracy of large dataset	time of small dataset	time of large dataset	number of rules of small dataset	number of rules of large dataset
2	51.17%	67.87%	225.2	61,710.0	29.3	667.3
6	51.17%	67.86%	228.5	60,481.0	29.3	667.3
10	51.17%	67.87%	223.7	62,466.0	29.3	667.3
14	51.17%	67.86%	229.5	73,521.0	29.3	667.3
18	51.17%	67.86%	228.1	69,577.0	29.3	667.3
Average	51.17%	67.86%	227.0	65,551.0	29.3	667.3

Average error rate of accuracy of the small dataset = 27.80%, and average error rate of accuracy of the large dataset = 3.48%



(a) Time vs. mindiffon the two sets

(b) The number of rules vs. mindiff on the two sets

(c) Accuracies vs. mindiff on the two sets

(d) Time vs. minsup on the two sets

(e) The number of rules vs. minsup on the two sets

(f) Accuracies vs. minsup on the two sets

Fig. 3 Comparisons of data behavior between the two sets on the two control variables

increases -0.1 rules on average. In other words, the former increases much more rules than the latter to improve the accuracy. Each leaf node (rule) can be applied to more test data than that with lower *mindiff* or *minsup*. This causes the accuracy-raise possibility of the former to be higher than that of the latter; contrarily, this causes the accuracy-reduction possibility of the former to be lower than that of the latter. Thus, the higher accuracy-raise

effect minuses the lower accuracy-reduction effect that increases more accuracy than that of the small dataset at each *mindiff* or *minsup*. As the effect of the reduction of the accuracy is overridden, we can see that both of AccuracyF and AccuracyL of the large dataset have the ascending tendencies as *mindiff* or *minsup* increases.

6.1.3 Summary of the data behavior between the two datasets

From the above experimental results, we can summarize that the model under-fitting problem has been reduced if the size of a training set is large enough. Additionally, the time and the number of rules of the two datasets are consistent approximately and positively correlated against *mindiff* and *minsup*. We can say both the variables are control variables while examining in the large datasets. Therefore, we can use the performance of the large dataset to represent the final performance of the MSDT algorithm. Finally, as shown in Table 6, we can operate the control variable, *mindiff*, to acquire best accuracy of MSDT handling the large dataset based on the optimal hyperparameters, *mindiff* = 0.25, *minsup* = 45%, *minqty* = 10 and *ub* = 6.

6.1.4 Comparisons of performance among the datasets based on sequences in different sizes of the sliding window

In Table 7, we show the breakdowns of the accuracies, the time and the numbers of rules of MSDT based on each of the both selecting measures vary with window size, α. Table 7(1) and (2) show that the overall average accuracy, time and numbers of rules of MSDT based on *sequentialGainRatio* are averagely better than those of MSDT based on *sequential-weighted-similarity*. This has explained the mention in Section 4.1.2. They also show that all the accuracies are all better than the baseline accuracies (defined in Section 6.1.2) of their corresponding α from 3 to 5.

To examine whether α is an influence factor to the accuracy, we use Pearson correlation coefficient (represented as ρ) to evaluate the correlation between all the values of α and the accuracies in both of Table 7(1) and (2). As we calculate that $\rho = -0.031668$, we can conclude that the sliding window size has no correlation with the accuracy during the three classification lifecycles, of which sliding windows are the most recent with incrementally-added sizes. Furthermore, both of Table 7(1) and (2) show that α is not an influence factor to the execution time and the numbers of rules either.

6.2 Results in the formal training stage

The results of Exp. III are presented in Table 8. We compare the performances between MSDT and the baseline algorithms. Table 8(1) and (2) show the performances of all the algorithms based on their respective optimal hyperparameter configurations in Experiment III.1 and Experiment III.2 respectively. Some results are worth being highly noticed. First, both the MSDT experiments perform better accuracies than all the baseline algorithms. Second, in Table 8(2), we note that MSDT outperforms all the three deep learning multi-label algorithms in accuracy. Third, in Table 8(1), MID+MSDT has better accuracy than all the baseline algorithms. However, MSDT still has better accuracies than MID+MSDT. The latter declines the accuracies because the discretization metric of the discretization method, MID, can only measure the strength of dependence between intervals of each attribute and single-labels rather than sequential-labels; so that it considers each sequential-label as a single-label identity and neglects the similarities among various sequential-labels.

Some of the other performance comparisons need further discussions in order as follows.

First, the comparisons of the time: In Table 8(1), we can see the average time of all the baseline algorithms are shorter than both of MSDT and MID+MSDT. Meanwhile, in Table 8(2), except the time of MLTSVM and RAkEL are larger than MSDT, the other 11 baseline methods are shorter than MSDT. However, both are not positive for those methods because they save the training time without spending any time to discover their sequential patterns from sequential-labels at the cost of lower accuracies. Therefore, we can say that the training time of MSDT is comparatively moderate without sacrificing the accuracies.

Second, the comparisons of the number of rules: In Table 8(1), we can see all the baseline algorithms generate too few rules to predict at the cost of the lower accuracies. They stop growing decision trees too early because they consider lots of various sequential-labels as the same multi-labels. For example, the multi-labeled methods would treat "123", "213" and "321" as the same. Meanwhile, in Table 8(2), six of all the algorithms are decision tree methods, including MSDT, CART-ML, ExtraTree-ML, RandomForest-ML, ExtraTrees-ML and iSoup-tree. It shows that CART-ML and ExtraTree-ML generate more rules than MSDT, except iSoup-tree. Contrast to the tree methods except MSDT, iSoup-tree spends more time to build a tree, however, it grows a smaller tree, which then generates too few rules to predict at the cost of the lower accuracies. The reason is that it stops growing the decision tree too early because they consider lots of various sequential-labels as the same multi-labels. Furthermore, we estimate the number of rules of both the ensemble algorithms, ExtraTrees-ML and RandomForest-ML as follows. ExtraTree-ML generated 1,268.7 rules through a decision tree. And, ExtraTrees-ML grew a forest with 100 extra-trees, which thus generated roughly 126,870 rules.

Table 7 Comparisons of the performances between MSDT based on the two measures among the three large datasets with sequences in different sizes of the sliding window in Experiment II

The most recent sliding window size	AccuracyF	AccuracyL	Average accuracy	Baseline accuracy	Time (ms.)	Number of rules	Error rate of accuracy
(1)							
$\alpha = 3$	64.86%	65.57%	65.22%	20.38%	97,513.9	646.7	6.16%
$\alpha = 4$	71.83%	71.48%	71.66%	19.14%	57,832.8	675.8	3.45%
$\alpha = 5$	66.03%	63.93%	64.98%	19.14%	60,312.8	794.1	3.46%
Average	67.57%	66.99%	67.29%	19.55%	71,886.5	705.5	4.36%
(2)							
$\alpha = 3$	64.76%	65.36%	65.06%	20.38%	120,925.8	1282.1	5.59%
$\alpha = 4$	71.35%	70.34%	70.85%	19.14%	76,981.6	1047.1	2.63%
$\alpha = 5$	65.03%	62.95%	63.99%	19.14%	65,902.8	1328.0	3.18%
Average	67.05%	66.22%	66.63%	19.55%	87,936.7	1219.1	3.80%

(1) The MSDT algorithm based on the *sequentialGainRatio* meaure. (2) The MSDT algorithm based on *sequential-weighted-similarity* measure

Table 8 Comparisons of the performances between MSDT and the baseline algorithms on the training set with size 10,000

Method	Average accuracy	Average time to build model(ms)	Average number of rules	Average error rate of accuracy
(1)				
MSDT	68.58%	149,021.0	1,368.7	5.35%
MID+MSDT	64.21%	279,283.7	4,692.7	6.05%
MMDT	57.89%	117.3	1.0	0%
MMC	58.35%	282.3	4.0	0%
MMD+MMDT	57.89%	748.0	4.0	0%
(2)				
MSDT	67.29%	143,169.0	639.7	14.09%
CART-ML	56.60%	231.7	1,268.7	3.81%
ExtraTree-ML	56.60%	99.0	1,268.7	3.81%
RandomForest-ML	60.01%	6,861.7	n/a	6.53%
ExtraTrees-ML	59.88%	1,738.3	n/a	6.44%
ML-KNN	49.33%	14.7	n/a	2.32%
MLTSVM	49.86%	334,305.3	n/a	0.00%
iSOUP-Tree	59.01%	1,996.3	33.0	5.37%
DNN-MLP-ML-adam	61.60%	7,374.3	n/a	4.15%
DNN-MLP-ML-sgd	59.25%	19,167.0	n/a	1.08%
DNN-MLP-ML-lbfgs	61.22%	71,878.3	n/a	10.62%
RAkEL	58.53%	218,745.0	n/a	0.00%
CDT	57.92%	6,890.7	n/a	0.00%
MVoting	58.85%	18,441.3	n/a	0.06%

(1) All of MSDT, MID+MSDT and MMDT are conducted with $\alpha = 3..5$ based on the optimal hyperparameters, $minsup = 45\%$, $mindiff = 25\%$, $minqty = 10$ and $ub = 6$; MMC is conducted with that based on the optimal hyperparameters, $minsup = 55\%$, $mindiff = 15\%$, $minqty = 10$ and $ub = 6$; and MMD+MMDT is conducted with that based on the optimal hyperparameters, $minsup = 50\%$, $mindiff = 20\%$, $minqty = 6$ and $ub = 6$. (2) All the above methods handle data preprocessed by the VMSP4MSDT algorithm

Besides, both of ExtraTrees-ML and RandomForest-ML grew several decision trees on various sub-samples of the sample data-set. The sub-sample size in RandomForest-ML is always the same as the data-set size while the samples are drawn with replacement if the bootstrap samples were used. As RandomForest-ML grew 100 decision trees, the same reason as ExtraTrees-ML is also suitable for RandomForest-ML. We found four of the baseline methods, CART-ML, ExtraTree-ML, RandomForest-ML and ExtraTrees-ML, prefer growing larger binary trees with the *gini* index than MSDT does. Therefore, they generate more rules. From the discussions mentioned above, we can say that the number of rules of MSDT is comparatively moderate without sacrificing the accuracies.

6.3 Discussion

All the results can be summarized as follows. First, we have excluded the size of sliding window as an influence factor to the accuracy, the time and the number of rules. Second, we have acquired the two control variables, *mindiff* and *minsup*, positively correlated with all the performances if the size of a training set is large enough. And, we suggest readers use the optimal parameters, *mindiff* = 25% and *minsup* = 45%, as the default values while using MSDT handling a large enough dataset. Third, no matter whether the features of the datasets are single-valued or multi-valued, MSDT outperforms all the baseline multi-label classification algorithms in accuracy even if three of them are deep learning multi-label algorithms. And, the time and the number of rules of MSDT are comparatively moderate without sacrificing the accuracies than all the baseline methods. Fourth, MSDT has achieved the average accuracy and the best accuracy: 67.29% and 73.36% respectively. All these things make it clear that MSDT not only can classify both of the large datasets, the multi-valued and the single-valued, but also performs well in terms of the accuracy.

7 Conclusions

This study starts with proposing a representative sequential pattern, i.e. a personalized sequential pattern, as the preference pattern of each non-cold-start user. This makes sequential pattern mining possible to be user-centric. Given some features of each user can cause one's item-sequences, we have shown that those features can also cause one's representative sequential pattern (denoted as a sequence-label). This paper therefore has presented the MSDT algorithm. It is to our knowledge the first algorithm that can classify data whose class labels are sequential, and some predictor attributes are multi-valued. And, the learned classifier is then used to predict each cold-start user an initial sequential-label (representative sequential pattern) only based on one's features. The rules generated can further help businesses/scientists to interpret what factors cause such behavior patterns of subjects. Finally, the predicted representative sequential pattern of each cold-start user is used to recommend one some initial matched item-sequences.

While applying MSDT to the application like a study of behavior of endangered or migration species, we suggest to rename the types of cold-start users (i.e. the previously-unseen, the rarely-doing or the rarely-buying) as the cold-start species of the previously-untagged, the new-migrating or rarely-migrating. As for future researches, combining the discretization methods with MSDT still leaves a potential improvement opportunity. MSDT is applicable to analyzing the item-sequences which have an upper bound requirement to constrain their maximal length using a user-specified size-constrained sliding window during a time period. The time complexity of MSDT has a potential improvement opportunity when an application needs a larger sliding window size. Besides, adapting and implementing our algorithms in the big data platforms such as *Spark* or *Hadoop* could address the problem about the real-time processing of data streams.

Acknowledgements I am very grateful to Professor Ray-I Chang and Huichen Huang for refining the writing of this paper; and those authors for sharing their API or open source codes: Jonathan Liang and Oliver Mannion, the authors of the API, CasperDataSets; Philippe Fournier-Viger, the author of the SPAM algorithm; Philippe Fournier-Viger and Antonio Gomariz, the authors of the VMSP algorithm; and all the authors of APIs in Python or Java, used to code all the baseline algorithms.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

1. Agrawal R, Ghosh S, Imielinski T, Iyer B, Swami A (1992) An interval classifier for database mining applications. In: Proceedings of VLDB'92, Vancouver, pp 560–573
2. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: Proceedings of ACM SIGKDD KDD'02, Edmonton, pp 429–435
3. Biswas S, Lakshmanan LVS, Ray SB (2017) Combating the Cold Start User Problem in Model Based Collaborative Filtering. arXiv:1703.00397 [cs.IR]
4. Breiman L (2001) Random Forests. Mach Learn 45(1):5–32
5. Chen YL, Hsu CL, Chou SC (2003) Constructing a multi-valued and multi-labeled decision tree. Expert Syst Appl 25(2):199–209
6. Chen WJ, Shao YH, Li CN, Deng NY (2016) MLTSVM: A novel twin support vector machine to multi-label learning. Pattern Recognit 52:61–74

7. Chou SC, Hsu CL (2005) MMDT: A multi-valued and multi-labeled decision tree classifier for data mining. Expert Syst Appl 28(4):799–812
8. Dhaliwal J, Puglisi SJ, Turpin A (2012) Practical efficient string mining. IEEE Trans Knowl Data Eng 24(4):735–744
9. Fader PS, Hardie BGS (2001) Forecasting repeat sales at CDNOW: a case study. Interfaces 31(3):S94–S107
10. Fayyad UM, Irani KB (1993) Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of IJCAI'93, Chambery, vol 2, pp 1022–1027
11. Fournier-Viger P, Wu CW, Tseng VS (2013) Mining maximal sequential patterns without candidate maintenance. In: Proceedings of ADMA'13, Hangzhou, pp 169–180
12. Fournier-Viger P, Wu CW, Gomariz A, Tseng VS (2014A) VMSP: Efficient vertical mining of maximal sequential patterns. In: Proceedings of Canadian AI'14, Montréal, pp 83–94
13. Fournier-Viger P, Gomariz A, Gueniche T, Soltani A, Wu CW, Tseng VS (2014b) SPMF: A java open-source pattern mining library. J Mach Learn Res 15:3569–3573
14. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
15. Graham B, Dodd D (2008) Security analysis, 6th edn. McGraw-Hill, New York
16. Han J, Kamber M (2006) Data mining: Concepts and techniques, 2nd edn. Morgan Kaufmann, San Francisco
17. Jaro MA (1989) Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. J Am Stat Assoc 84(406):414–420
18. Kadous MW, Sammut C (2005) Classification of multivariate time series and structured data using constructive induction. Mach Learn 58:179–216
19. Kohavi R (1995) A study of cross validation and bootstrap for accuracy estimation and model selection. In: Proceedings of IJCAI'95, Montreal, pp 1137–1143
20. Osojnik A, Panov P (2018) Tree-based methods for online multi-target regression. Intell Inf Syst 50(2):315–339
21. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine Learning in Python. Mach Learn Res 12:2825–2830
22. Plantevit M, Choong YW, Laurent A, Laurent D, Teisseire M (2005) M2SP: Mining sequential patterns among several dimensions. In: Proceedings of PKDD'05, Porto, pp 205–216
23. Quinlan JR (1979) Discovering rules from large collections of examples: a case study. In: Michie D (ed) Expert systems in the microelectronic age. 6th edn. Edinburgh University Press, Edinburgh, pp 169–201
24. Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106
25. Quinlan JR (1993) C4.5: Programs for machine learn. Morgan Kaufmann, San Mateo
26. Raiko T, Valpola H, Lecun Y (2012) Deep learning made easier by linear transformations in perceptrons. In: Proceedings of PMLR, La Palma, vol 22, pp 924–932
27. Read J, Martino L, Luengo D, Olmos P (2015) Scalable multi-output label prediction: From classifier chains to classifier trellises. Pattern Recogn 48(6):2096–2109
28. Sahoo N, Singh PV, Mukhopadhyay T (2012) A hidden Markov model for collaborative filtering. MIS Q 36(4):1329–1356
29. Schlüter T, Conrad S (2012) Hidden markov model-based time series prediction using motifs for detecting inter-time-serial correlations. In: Proceedings of ACM SAC'12, Riva del Garda, pp 158–164
30. Scikit-learn developers (2018) Scikit-learn user guide Release Release 0.19.2. https://scikit-learn.org/0.19/_downloads/scikit-learn-docs.pdf
31. Schafer JB, Frankowski D, Herlocker J, Sen S (2007) Collaborative filtering recommender systems. In: Brusilovsky P, Kobsa A, Nejdl W (eds) The adaptive web. Lecture notes in computer science. Springer, Berlin, p 4321
32. Steinberg D, Colla P (2009) Cart: Classification and regression trees. In: Wu X, Kumar V (eds) The top ten algorithms in data mining, vol 9. CRC press, pp 179–203
33. Szymański P, Kajdanowicz T (2019) Scikit-multilearn: a scikit-based Python environment for performing multi-label classification. J Mach Learn Res 20(1):209–230
34. Szymański P, Kajdanowicz T, Kersting K (2016) How is a data-driven approach better than random choice in label space division for multi-label classification? Entropy 18(282):1–30
35. Tan PN, Steinbach M, Kumar V (2006) Introduction to data mining. Addison Wesley, Boston
36. Tsai CJ (2014) A study of improving the performance of mining multi-valued and multi-labeled data. Inf-Lithuan 25:95–111
37. Tsoumakas G, Katakis I, Vlahavas I (2011) Random k-Labelsets for Multilabel Classification. IEEE Knowl Data En 23(7):1079–1089
38. Wang M, Iyer B, Vitter JS (1998) Scalable mining for classification rules in relational databases. In: Proceedings of IDEAS'98, Cardiff, pp 58–67
39. Winkler WE (2006) Overview of record linkage and current research directions, research report series, statistics #2006-2. U.S Census Bureau, Washington
40. Xiao S, Dong M (2015) Hidden semi-Markov model-based reputation management system for online to offline (O2O) e-commerce markets. Decis Support Syst 77:87–99
41. Xing Z, Pei J, Keogh E (2010) A brief survey on sequence classification. ACM SIGKDD Explor Newsl 12(1):40–48
42. Zaki MJ (2001) Spade: an efficient algorithm for mining frequent sequences. Mach Learn 42(1-2):31–60
43. Zhang ML, Zhou ZH (2007) ML-KNN: A lazy learning approach to multi-label learning. Pattern Recognit 40(7):2038–2048
44. Zheng Y (2015) Trajectory data mining: an overview. ACM Trans on Intell Syst and Technol 6(3:29):1–41
45. Zhu X, Davidson I (2007) Knowledge discovery and data mining: Challenges and realities. IGI Global, Hershey

Chang-Ling Hsu received the B.B.A. degree in Computer Science from Soochow University in 1989, the M.E. degree in Information and Electronic Engineering from National Central University in 1991 and the Ph.D. degree in Information Management from National Central University in 2004. He had served in software enterprises and IT departments during 1991-2000. From 2000 to 2006, he joined the faculty of Department of Information Management in Hungkuang University, Taiwan. He is currently one of the faculty at Department of Information Management in Ming Chuan University, Taiwan. His research interests include machine learning, big data analytics and information retrieval.