A Data Generator for Multi-Stream Data

Zaigham Faraz Siddiqui[†], Myra Spiliopoulou[†],
Panagiotis Symeonidis[*], and Eleftherios Tiakas[*]

University of Magdeburg[†]; University of Thessaloniki[*].
[siddiqui,myra]@iti.cs.uni-magdeburg.de; [symeon,tiakas]@csd.auth.gr

Abstract. We present a stream data generator. The generator is mainly intended for multiple interrelated streams, in particular for objects with temporal properties, which are fed by dependent streams. Such data are e.g. customers their transactions: learning a model of the customers requires considering the stream of their transactions. However, the generator can also be used for conventional stream data, e.g. for learning the concepts of the transaction stream only.
The generator is appropriate for testing classification and clustering algorithms on concept discovery and adaptation to concept drift. The number of concepts in the data can be specified as parameter to the generator; the same holds for the membership of an instance to a class. Hence,it is also appropriate for synthetic datasets on overlapping classes or clusters.

1 Introduction

Most of the data stored in databases, archives and resource repositories are not static collections: they accumulate over time, and sometimes they cannot (or should not) even be stored permanently - they are observed and then forgotten. Many incremental learners and stream mining algorithms have been proposed in the last years, accompanied by methods for evaluating them [3, 1]. However, modern applications ask for more sophisticated stream learners than can currently be evaluated on synthetically generated data. In this work, we propose a generator for complex stream data that adhere to multiple concepts and exhibit drift. The generator can be used for the evaluation of (multi-class) stream classifiers, stream clustering algorithms over high-dimensional data and relational learners on streams.

Our generator is inspired by recommendation engines, where data are essentially a combination of static objects and adjoint streams: people rank items - the rankings constitute a *fast* (or conventional) stream; new items show up, while old items are removed from the provider's portfolio - the items constitute a *slow* stream; new users show up, while old users re-appear and rank items again, possibly exhibiting different preferences as before - users also constitute a *slow* stream. In [4] we devised the term *perennial objects* for a stream of objects that appear more than once and may change properties. In conventional relational learning, these objects would be static and have one fixed label, while in relational stream learning they may have different labels at different times. In [4] we proposed a decision tree stream classifier for a stream of perennial objects.

Our generator extends the generator of static item recommendations presented in [6] in two ways. First, our generator builds a *stream* of ratings, hence it can be used to test recommenders designed for dynamic data. Second, and most importantly, our generator builds the ratings' stream upon a synthetic set of evolving profiles, thus allowing the evaluation of learners designed for complex dynamic environments. In particular, consider an application that categorizes customers into classes A, B, C (A is best), given their response to recommendations and considering some demographic attributes (e.g. having children, having a car etc). What synthetic data are needed to evaluate a learner for this application? A synthetic set (or stream) of ratings is not sufficient, because it does not contain customers (and their labels). Next to the stream of ratings, we need a set that describes the customers and associates them to their ratings in a *non-random* way. Further, since customer preferences may change, concept drift must be incorporated into the relationship between customers and their ratings. Our generator is designed for this kind of multi-relational (stream) learning.

The core idea of our generator is as follows. The preference of a user towards some item(s) defines its behaviour. Multiple users' exhibiting similar behaviour can be grouped/categorised together as a single user profile. Conversely to learning task where these profiles are learned from among a group of users, the generator first creates these user profiles which serve as a prototypes. These profiles are then used to generate individual user data according the item preferences stored in them. Noise can be imputed to the data by forcing a user to rank in discordance to her profile with some probability. Drift is imputed to the data by allowing a profile to exist only for some timepoints and then forcing it to mutate to one or more profiles with some probability.

The paper is organized as follows. In Section 2 we give a formal description of the problem along with the related work. We explain the multi-relational generator in Section 3 with some results in Section 4. We summarise and discuss future improvements in Section 5.

2 Problem Specification and Background Literature

In this section we explain the learning task over the multiple interrelated streams (i.e., slow and fast streams) in more detail, which our generator is going to simulte. In subsection 2.2 we also discuss the studies that address similar problems.

2.1 Problem Specification

In our introductory example, we considered a slow stream of users $\mathcal{T}$ and fast ranking/transaction stream $\mathcal{S}^1$ that feeds the user stream (i.e., the users') with rantings/purchases from a slow stream of items $\mathcal{S}^2$. Stream of user is referred as *target* streams as the learning task concerns solely $\mathcal{T}$, e.g. finding groups of users that show similar behaviour when rating/purchasing different items, predicting whether a user will like a certain item (Y) or not (N), or labelling users on their "lifetime value" for the company (typically in four classes A, B, C, D where A

is best and D is worst). The contents of the inter-connected streams (i.e., the ranking stream and the item stream via ranking stream) should be taken into account when building the classifier.

Learning on a Stream of Perennial Objects. The stream of perennial objects (i.e., slow stream) $\mathcal{T}$ exhibits three properties that are atypical for streams. Differently from conventional stream objects that are seen, processed and forgotten, objects of $\mathcal{T}$ may not be deleted: an examinations office may file away the results of a successful exam, but does not file away the students who passed the exam; a product purchase may be shifted to a backup medium after completion, but the customer who did the purchase remains in the database. One can even argue that $\mathcal{T}$ is not a stream at all. However, it is obvious that new objects arrive (new customers, new students), while old objects are filed away after some time (e.g. students who completed their degree and customers who have quitted the relationship with the company). We use the term *perennial objects* or *stream of perennial objects* for the slow stream $\mathcal{T}$.

Second, the objects in $\mathcal{T}$ may appear several times, e.g. whenever the properties of a user (e.g. her address) change and whenever this user is referenced by a fast stream (i.e. when the user assigns a new rating or purchases a new item). Third, the label of a $\mathcal{T}$ object may change over time: a user who earlier responded positively towards and item (Y) may stop doing so (label becomes N); a B-user may become an A-user or a C-user. Hence, the label of a perennial objects x is not a constant; at every timepoint t, at which x is observed, its label is $label(x, t)$. The learning task is to predict this label at t, given the labelled data seen thus far and given the streams that feed $\mathcal{T}$.

2.2 Related Work

Our generator is inspired from the properties of the perennial stream and builds on a generator for recommender systems by Symeonidis et al.[6]. This generator is intended for learning in a static context. We outline it here briefly.

The generator of [6] produces a unipartite user-user (friendship) network and a bipartite user-item rating network. In contrast to purely random (i.e., Erdos-Renyi) graphs, where the connections among nodes are completely independent random events, the synthetic model ensures dependency among the connections of nodes, by characterizing each node with a m-dimensional vector with each element a randomly selected real number in the interval [-1,1]. This vector represents the initial user profile used for the construction of the friendships and ratings profiles, which are generated as follows:

- For the construction of the friendship network, two nodes are considered to be similar and thus of high probability to connect to each other if they share many close attributes in their initial user profile. Given a network size N and a mean degree k of all nodes, the generator starts with an empty network with N nodes. At each time step, a node with the smallest degree is randomly

selected (there is more than one node having the smallest degree). Among all other nodes whose degrees are smaller than k, this selected node will connect to the most similar node with probability $1 - p$, while a randomly chosen one with probability p. The parameter $p \in [0, 1]$ represents the strength of randomness in generating links, which can be understood as noise or irrationality that exists in almost every real system.
- For the construction of the user-item rating network, the generator follows a similar procedure. It uses the following additional parameters as well: (i) the ratings range, (ii) the mean number of rated items by all users. Notice that each user can rate different items from others and has in his profile a different number of rated items, following the power law distribution.

xSocial is a multi-modal graph generator that mimics real social networking sites to produce simultaneously a network of friends and a network of their co-participation [2]. In particular, xSocial consists of a network with N nodes, each of which has a preference value $calf_i$. At each time, every node performs three independent actions (write a message, add a friend and comment on a message). A node chooses his friends either by their popularity of by the number of messages on which they have commented together, which is determined by his preference $calf_i$. A node can also follow the updated status of his friends by putting comments on the corresponding newly written messages.

The generator of Symeonidis et al., [6] uses both structural (friendship network) and content-based information (item-rating network) for making recommendations to the users. However, it is only suitable for static learning. xSocial, on the other hand, simulates the temporal/stream-based problem but it only uses the structural information (i.e., networks of friends and their interactions) for making recommendations to the users. Different from [6], our generator aims to alleviate the problem of static recommendations by making use of the dynamic ratings profile (i.e., a user's rating preferences may change overtime), and unlike xSocial [2] its primary focus is on content-based recommendations.

3 Generating Profiles & Transactions with Concept Drift

Our generator is inspired by the idea of predicting user ratings in a recommendation engine, and builds upon the generator of [6] (c.f. Section 2.2). In particular, our generator creates data according to the following scenario:

> Each user adheres to a *user profile*, while each item adheres to an *item profile*[1]; the user profiles correspond to classes. The rating of a user u for an item i depends upon affinity/preference the of the user profile of u towards the item profile of i; ratings are generated at each timepoint t. At certain timepoints, user profiles mutate, implying that the ratings of the users for the items change. An example of learner object for some

[1] just as user profiles serve as prototypes for the generation of concrete user data (c.f. Section 1), item profiles serve the same purpose

Table 1. Parameters of the generator.

Param	Description
P^i	set of item profiles with $N^i = \|P^i\|$ number of profiles
P^u	set of user profiles with $N^u = \|P^u\|$ number of profiles
n^i	number of items per item profile
n^u	number of users per user profile
v^i	number of synthetic variables that describe an item profile
v^u	number of synthetic variables that describe a user profile
τ_d	number of drift levels across the time axis
A_d^u	number of active user profiles at drift level d
$\mathcal{L}$	max lifetime of a drift level as number of timepoints
R	max number of items rated by a user at any timepoint
$\phi_{\mathcal{U}\to\mathcal{I}}$	the probability of a user profile $\mathcal{U}$ selecting an item from item profile $\mathcal{I}$ for rating.

Item Profiles	Var 1	Var 2	Var 3	Var 4
IP1	23 ±4	52 ±9	97 ±2	8 ±9
IP2	2 ±9	1 ±7	46 ±3	91 ±6
IP3	72 ±7	71 ±3	26 ±2	52 ±1
IP4	3 ±4	2 ±4	27 ±5	25 ±3

Fig. 1. Sample item profiles with $v^i = 4$ synthetic variables. The mean and variance associated with each variable are used to generate items according to the normal distribution.

learner is to predict the profile of a user at a given timepoint, when provided with the users' ratings data.

In more detail, our generator takes as input the parameters depicted in Table 1, and described in sequel. It generates: item profiles and from them items; user profiles and from them users; and ratings of users for items at each timepoint. A user profile may live at most $\mathcal{L}$ timepoints before it mutates.

Generation of item profiles and items.

Item profiles are described by v^i synthetic variables. The generator creates a set P^i with N^i item profiles and stores for each one the mean and variance of each of the v^i variables. Next, each of these item profiles is used as prototype for the generation of n_i items, producing $n^i \times N^i$ items in total. Items also adhere to the v^i variables; the value of each variable in an item adhering to profile $\mathcal{I}$ is determined by the mean and variance of this variable in the profile $\mathcal{I}$. The description of item profiles and is depicted in Figure 1.

The number of items considered (rated) at some timepoint may vary from one timepoint to the next, but there is no bias towards items of some specific profile(s). Hence, item profiles are not exhibiting concept drift.

User Profiles	Var 1	Var 2	Var 3	Item Profile Probabilities			
UP1	13 ±0	22 ±5	51 ±1	IP1	IP2	IP3	IP4
				0.1	0.6	0.25	0.05
				20 ±5	50 ±8	80 ±9	29 ±1
UP2	34 ±4	55 ±0	68 ±9	IP1	IP2	IP3	IP4
				0.7	0.1	0.1	0.1
				90 ±2	25 ±5	93 ±4	9 ±1
UP3	21 ±5	98 ±4	1 ±5	IP1	IP2	IP3	IP4
				0.2	0.5	0.1	0.2
				42 ±2	95 ±2	10 ±0	12 ±5

Fig. 2. Sample user profiles with mean and variance for synthetic variables with probabilities of selecting an item from a certain item profile (row 1) and mean and variance of the rating that item (row 2), where $v^u = 3$.

Generation and transition of user profiles.

User profiles are described by a set of parameters v^u. These are synthetic variables. The generator creates a set P^u with N^u user profiles and stores for each one the mean and variance of each variable in v^u. User profiles serve as templates for the generation of users, in much the same way as item profiles are used to generate items. However, there are two main differences. First, user profiles are subject to transition, and not all of them are active at each drift level $d = 1, \ldots, \tau_d$. Second, a user profile exhibits affinity towards some item profiles, expressed through the probabilities between the item profile and the user profile. The description of user profiles is depicted in Figure 2.

The affinity of user profiles towards item profiles manifests itself in the user ratings: a generated user adheres to some user profile and rates items belonging to the item profile(s) preferred by her user profile. The affinity $\phi_{\mathcal{U}\to\mathcal{I}}$ is defined as the probability of a user profile $\mathcal{U}$ selecting an item from item profile $\mathcal{I}$ for rating. The probability $\phi_{\mathcal{U}\to\mathcal{I}}$ is controlled by a user-defined global parameter $UP2IP \in [0, 1]$. If $UP2IP$ is close to zero, user profiles show strong affinity towards a certain item profile while if the value is closer to 1, the probabilities are initialised randomly.

At each drift level $d = 1, \ldots, \tau_d$, only a subset of user profiles $A^u_d \subseteq P^u$ are active[2]. The active profiles at each drift level is determined at the beginning. For drift level $d > 1$, the generator maps the profiles A^u_{d-1} of level $d-1$, to the new profiles A^u_d of level d on similarity, i.e. the transition probability from an old to a new profile is a function of the similarity between the two profiles. The result is a *profile transition graph*, an example of which is depicted in Figure 3. This graph is generated and then the thread of each profile is recorded for inspection.

The coupling of profile transition to the profile similarity function ensures that profile mutation corresponds to a gradual drift rather than an abrupt shift. The extent of profile mutation is further controlled by a user-defined global variable $\in [0, 1]$ that determines the *true preference* of an old user profile for a

[2] the number of active profiles at each drift level d is an integer and calculated using $\frac{N^u}{d}$ and is similar for each drift level

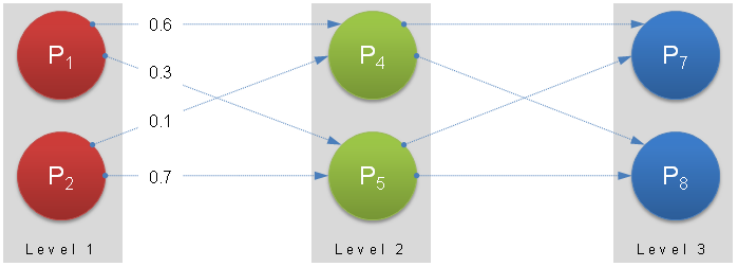

Fig. 3. A profile transition graph; each column corresponds to a timepoint, indicating that the number of profiles/classes may change from one timepoint to the next (transition probabilities between level 2 and level 3 have been omitted)

new user profile. A value close to zero means that the most similar new profile will always be preferred. Larger values allow for a weaker preferential attachment, while a value close to 1 means that the new profile is chosen randomly, and the transition is essentially a concept shift rather than a drift (a "drift" is a change of gradual nature, e.g. when a profile mutates into a similar one; while a "shift" is a more drastic and abrupt change, e.g. a profile being *replaced* by another one). The similarity between two user profiles $\mathcal{U}$ and $\mathcal{U}'$ is defined in Equation 1.

$$sim(\mathcal{U},\mathcal{U}') = \sqrt{\sum_{\mathcal{I}\in P^i} (\phi_{\mathcal{U}\to\mathcal{I}} - \phi_{\mathcal{U}'\to\mathcal{I}})} \tag{1}$$

where $\phi_{\mathcal{U}\to\mathcal{I}}$ is the prob. of rating an item from profile $\mathcal{I}$ for $\mathcal{U}$, $\mathcal{U} \in A^u_d$ and $\mathcal{U}' \in A^u_{d+1}$.

The affinity of user profiles towards item profiles manifests itself in the user ratings: a generated user adheres to some user profile and rates items belonging to the item profile(s) preferred by her user profile. Affinity is also affected by profile transitions. Once a user profile $\mathcal{U}$ mutates to $\mathcal{U}'$, all its users adhere to the $\mathcal{U}'$ profile: they prefer the item profiles to which $\mathcal{U}'$ shows affinity, and rate items adhering to these item profiles.

Generation of users and ratings.

For each user profile $\mathcal{U} \in P^u$, the generator creates n^u users. As for items, users adhere to the set of parameters v^u as user profiles; the value of each parameter in a user adhering to profile $\mathcal{U}$ is determined by the mean and variance of this variable in the profile $\mathcal{U}$.

The profiles of each drift level d exists for at most $\mathcal{L}$ timepoints, before profile transition occurs; the lifetime of a profile is chosen randomly. At each of these timepoints, the generator creates ratings for all users in each active profile. For each user profile $\mathcal{U}$ and user u adhering to $\mathcal{U}$, and for each item profile $\mathcal{I}$ is selected based on the probability $\phi_{\mathcal{U}\to\mathcal{I}}$. An item i is randomly chosen from the

$\mathcal{I}$ and a rating value is generated based on mean and variance in $\mathcal{U}$ for rating (c.f. Figure 2). A user can rank at most R items per timepoint.

4 Working with the Data Generator

We have used our generator to evaluate the performance of the classification rule miner (CRMPES) [5] which is incorporated into a tree induction algorithm (TrIP) [4]. CRMPES is used to generate new richer attributes that exploit dependency between attributes by using classification rules which otherwise are ignored by the TrIP. To test CRMPES, we developed the generator with user and item profiles. The purpose was to study how CRMPES adapts to complex patterns (spanning multiple attributes) in the presence of concept drift. The details on the experiments can be found in the paper [5].

5 Conclusions

We presented a multi-stream generator that has been inspired from the domain of recommendation system. It generates ratings data for users according to user profiles. With time the profiles mutates into newer ones. The mutation can be adjusted to simulate drastic shifts as well more gradual drifts. The generator can be used for evaluating supervised and unsupervised learning task for discovering and adaptation to concept drift.

Acknowledgements: Work of the first author was partially funded by the German Research Foundation project SP 572/11-1: IMPRINT: Incremental Mining for Perennial Objects. The cooperation between the two research groups is supported by the DAAD project travel grant 50983480.

References

1. A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. KDD '09, pages 139–148. ACM, 2009.
2. N. Du, H. Wang, and C. Faloutsos. Analysis of large multi-modal social networks: Patterns and a generator. ECML/PKDD '10, pages 393–408. Springer, 2010.
3. J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. KDD '09, pages 329–338. ACM, 2009.
4. Z. F. Siddiqui and M. Spiliopoulou. Tree induction over perennial objects. SSDBM'10, pages 640–657. Springer-Verlag, 2010.
5. Z. F. Siddiqui and M. Spiliopoulou. Classification rule mining for a stream of perennial objects. RuleML@IJCAI '11. Springer-Verlag, 2011.
6. P. Symeonidis, E. Tiakas, and Y. Manolopoulos. Transitive node similarity for link prediction in social networks with positive and negative links. RecSys '10, pages 183–190. ACM, 2010.