FPSMining: A Fast Algorithm for Mining User Preferences in Data Streams

Jaqueline A. J. Papini, Sandra de Amo, Allan Kardec S. Soares

Federal University of Uberlândia, Brazil
jaque@comp.ufu.br, deamo@ufu.br, allankardec@gmail.com

Abstract. The traditional preference mining setting, referred to here as the *batch setting*, has been widely studied in the literature in recent years. However, the dynamic nature of mining preferences increasingly requires solutions that quickly adapt to changes. The main reason for this is that user's preferences are not static and can evolve over time. In this article, we address the problem of mining *contextual* preferences in a data stream setting. *Contextual Preferences* have been recently treated in the literature and some methods for mining this special kind of preferences have been proposed in the batch setting. The main contributions of this article are the formalization of the contextual preference mining problem in the stream setting and the introduction of two very efficient algorithms for solving this problem. We implemented both algorithms and showed their efficiency and scalability through a set of experiments over synthetic and real datasets.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*data mining*

Keywords: context-awareness, data mining, data streams, incremental learning, preference mining

1. INTRODUCTION

The huge increase in the volume of digital data seen in recent years was partly caused by a new class of emerging applications - applications in which the data are generated at very high rates, in the form of data streams. In general, a data stream may be seen as a sequence of relational tuples that arrive continuously in variable time. Some typical fields of application for data streams are: the financial market, credit card transaction flow, web applications and sensor data. Traditional approaches for data mining cannot successfully process the data streams mainly due to the potentially infinite volume of data and its evolution over time. Consequently, several data stream mining techniques have emerged to deal properly with this new data format [Domingos and Hulten 2000; Gama 2010; Bifet et al. 2011].

Most of the research on preference mining has focused on the batch setting, where the mining algorithm has a set of static information on user preferences at its disposal [Jiang et al. 2008; de Amo et al. 2013]. However, in most situations, user preferences are dynamic. For instance, consider an *online news site* that wants to discover the preferences of its users regarding news and make recommendations based on that. Notice that due to the dynamic nature of news, it is plausible that user's preferences would evolve rapidly with time. In times of elections, a user can be more interested in *politics* than in *sports*. In times of Olympic Games, it would probably be the opposite.

This work focuses on a particular kind of preferences, the *contextual preferences*. Preference Models can be specified under either a *quantitative* [Crammer and Singer 2001] or a *qualitative* [de Amo et al. 2013] framework. In the quantitative formulation, preferences about movies (for instance) can be elicited by asking the user to rate each movie. In the qualitative formulation, the preference model consists in a set of rules specified in a mathematical formalism, able to express user preferences. In this article, we consider the *contextual preference rules* (cp-rules) introduced by Wilson [2004]. A cp-rule

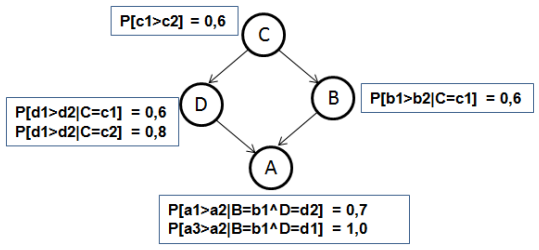
We thank the Brazilian Research Agencies CNPq and FAPEMIG for supporting this work.

Id	A	B	C	D
t_1	a_1	b_1	c_1	d_1
t_2	a_1	b_1	c_1	d_2
t_3	a_2	b_1	c_1	d_2
t_4	a_1	b_2	c_1	d_2
t_5	a_2	b_1	c_2	d_1
t_6	a_3	b_1	c_1	d_1

(a)

(t_1,t_2)
(t_1,t_3)
(t_4,t_5)
(t_4,t_2)
(t_5,t_6)
(t_3,t_5)
(t_4,t_1)

(b)



(c)

Fig. 1. (a) An instance I. (b) A Preference Database $\mathcal{P}$. (c) Preference Network $\mathbf{PNet}_1$.

allows to specify that some values of a particular attribute are preferable to others *in a given context*. For instance, a user may prefer comedies to dramas *if the director is Woody Allen*.

In this article we propose two qualitative methods for mining contextual preferences from a *preference stream* sample, namely the FPSMining and IncFPSMining algorithms. As pointed out by Kontaki et al. [2010], algorithms designed for processing data streams should satisfy the following properties: (1) fast response time; (2) *incremental evaluation* in order to be able to efficiently detect and incorporate changes in data over time; (3) low rate of data access and (4) in memory storage in order to avoid expensive disk access. The FPSMining algorithm has been introduced by Papini et al. [2013], a preliminary short version of the present article. Although executing very fast in the datasets considered in our previous work [Papini et al. 2013], it does not satisfy properties (2) and (4) above. In this article we extend the preliminary version [Papini et al. 2013] with a new algorithm, named IncFPSMining, which follows an incremental approach, differently from FPSMining which extracts an entirely brand new preference model from the preference stream data at each *refreshing* point. In order to property (4) be satisfied in our approach, a technique for memory management has been implemented in order to limit the growth of the *sufficient statistics* when dealing with a large amount of data. For lack of space, only the main ideas and the results involving this technique is presented in this article. Besides these two important improvements, the current version also includes: (a) Experiments with synthetic data in order to enable tests with a huge amount of data; (b) The computation of the statistical significance of the results concerning the two algorithms over synthetic data; (c) The introduction of another quality measure – the Comparability Rate (CR) and also (d) The complexity analysis of the two proposed algorithms.

2. BACKGROUND ON CONTEXTUAL PREFERENCE MINING IN THE BATCH SETTING

In this section we briefly introduce the problem of mining contextual preferences in a batch setting. Please see [de Amo et al. 2013] for more details on this problem.

A *preference relation* on a finite set of objects $A = \{a_1, a_2, ..., a_n\}$ is a strict partial order over A, that is a binary relation $R \subseteq A \times A$ satisfying the irreflexivity and transitivity properties. We denote by $a_1 > a_2$ the fact that a_1 is preferred to a_2. A *Preference Database* over a relation R is a finite set $\mathcal{P} \subseteq \text{Tup}(R) \times \text{Tup}(R)$ which is *consistent*, that is, if $(u, v) \in \mathcal{P}$ then $(v, u) \notin \mathcal{P}$. The pair (u, v), usually called bituple, represents the fact that the user prefers *the tuple u to the tuple v*. Fig. 1 (b) illustrates a preference database over R, representing a sample provided by the user about his/her preferences over tuples of I (Fig. 1 (a)).

The problem of mining contextual preferences in the batch setting consists in extracting a *preference model* from a preference database provided by the user. The preference model is specified by a *Bayesian Preference Network* (BPN), specified by (1) a directed acyclic graph G whose nodes are attributes and the edges stand for attribute dependency and (2) a mapping θ that associates to each node of G a finite set of conditional probabilities. Fig. 1(c) illustrates a BPN $\mathbf{PNet}_1$ over the relational schema $R(A, B, C, D)$. Notice that the preference on values for attribute B depends on the context C: if $C = c1$, the probability that value b_1 is preferred to value b_2 for the attribute B is 60%. A BPN allows

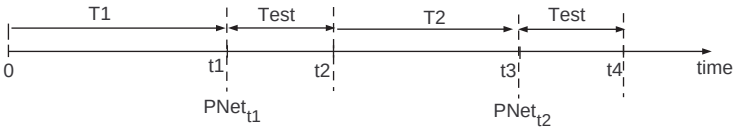


Fig. 2. The dynamics of the mining and testing processes through time

the inference of a *preference ordering* on tuples. The following example illustrates how this ordering is obtained. For lack of space we do not present a formal definition here. For more details on the theoretical background behind this ordering, please see [Wilson 2004; de Amo et al. 2013].

Example 2.1 *Preference Order.* Let us consider the BPN $\mathbf{PNet}_1$ depicted in Fig. 1(c). In order to compare the tuples $u_1 = (a_1, b_1, c_1, d_1)$ and $u_2 = (a_2, b_2, c_1, d_2)$, we proceed as follows: (1) Let $\Delta(u_1, u_2)$ be the set of attributes for which u_1 and u_2 differ. In this example, $\Delta(u_1, u_2) = \{A, B, D\}$; (2) Let $\min(\Delta(u_1, u_2)) \subseteq \Delta(u_1, u_2)$ such that the attributes in $\min(\Delta)$ have no ancestors in Δ (according to graph G underlying the BPN $\mathbf{PNet}_1$). In this example $\min(\Delta(u_1, u_2)) = \{D, B\}$. The necessary and sufficient conditions for u_1 to be preferred to u_2 are: $u_1[D] > u_2[D]$ and $u_1[B] > u_2[B]$; (3) Compute the following probabilities: p_1 = probability that $u_1 > u_2 = P[d_1 > d_2 | C = c_1] * P[b_1 > b_2 | C = c_1] = 0.6 * 0.6 = 0.36$; p_2 = probability that $u_2 > u_1 = P[d_2 > d_1 | C = c_1] * P[b_2 > b_1 | C = c_1] = 0.4 * 0.4 = 0.16$. In order to compare u_1 and u_2 we select the higher between p_1 and p_2. In this example, $p_1 > p_2$ and so, we infer that u_1 is preferred to u_2. If $p_1 = p_2$ we conclude that u_1 and u_2 are incomparable.

A BPN is evaluated by considering its *accuracy (acc)* and *comparability rate (CR)* with respect to a *test* preference database $\mathcal{P}$. The *accuracy* is defined by $\text{acc}(\mathbf{PNet}, \mathcal{P}) = \frac{N}{M}$, where M is the number of bituples in $\mathcal{P}$ and N is the amount of bituples $(t_1, t_2) \in \mathcal{P}$ compatible with the preference ordering inferred by **PNet** on the tuples t_1 and t_2. The *comparability rate* is defined by $\text{CR}(\mathbf{PNet}, \mathcal{P}) = \frac{F}{M}$ where F is the number of elements of $\mathcal{P}$ which are comparable by **PNet**. The comparability rate allows a better understanding of the overall behavior of the model quality - with this measure is possible to identify, for example, if a drop in accuracy was caused because the model was not able to compare much of bituples submitted to it, or because the model compared them erroneously. The *precision (prec)* is defined by $\text{prec}(\mathbf{PNet}, \mathcal{P}) = \frac{acc}{CR} = \frac{N}{F}$. Notice that each one of the three measures can be derived from the two others.

3. PROBLEM FORMALIZATION IN THE STREAM SETTING

The main differences between the batch and the stream settings concerning the contextual preference mining problem we address in this article may be summarized as follows:

—*Input data*: to each sample bituple (u, v) collected from the stream of clicks from a user on a site, is associated a timestamp t standing for the time the user made this implicit choice. Let T be the infinite set of all timestamps. So, the input data from which a preference model will be extracted is a *preference stream* defined as a (possibly) infinite set $P \subseteq Tup(R) \times Tup(R) \times T$ which is *temporally consistent*, that is, if $(u, v, t) \in P$ then $(v, u, t) \notin P$. The triple (u, v, t) that we will call *temporal bituple*, represents the fact that the user prefers tuple u over tuple v at the time instant t.

—*Output*: the preference model to be extracted from the preference stream is a *temporal BPN*, that is, a $PNet_t$ representing the model state at instant t. At each instant t the algorithm is ready to return a preference model $PNet_t$ updated with the stream elements until instant t.

—*The preference order induced by a BPN at each instant t*: At each instant t we are able to compare tuples u and v by employing the Preference Model $PNet_t$ updated with the elements of the preference stream until the instant t. The preference order between u and v is denoted by $>_t$ and is obtained as illustrated in example 2.1.

—*The accuracy and comparability rate at instant t*: The quality of the preference model $PNet_t$ returned by the algorithm at instant t is measured by considering a finite set $Test$ of preference

		$c_1 > c_2$	$c_2 > c_1$	$c_4 > c_5$
A	a_1	3	1	-
	a_2	-	-	2
B	b_3	1	-	1
	b_5	1	-	1

$c_1 > c_2$	4
$c_2 > c_1$	2
$c_4 > c_5$	3

(a)

		$c_1 > c_2$	$c_2 > c_1$	$c_4 > c_5$
A	a_1	3	2	-
	a_2	-	-	2
B	b_3	1	-	1
	b_5	1	-	1
	b_6	-	1	-

$c_1 > c_2$	4
$c_2 > c_1$	3
$c_4 > c_5$	3

(b)

	u_1			u_2		
T	A	B	C	A	B	C
t_1	a_1	b_3	c_1	a_1	b_3	c_2
t_2	a_1	b_3	c_2	a_1	b_5	c_1
t_3	a_2	b_5	c_2	a_1	b_3	c_1
t_4	a_2	b_3	c_4	a_2	b_6	c_5
t_5	a_1	b_5	c_1	a_1	b_5	c_2
t_6	a_2	b_3	c_4	a_2	b_3	c_5
t_7	a_1	b_3	c_1	a_1	b_5	c_2
t_8	a_2	b_5	c_1	a_1	b_6	c_2
t_9	a_1	b_5	c_4	a_2	b_5	c_5
t_{10}	a_1	b_6	c_2	a_1	b_6	c_1

(c)

Fig. 3. (a) Sufficient statistics for attribute C at the time instant t_9. (b) Sufficient statistics for attribute C at the time instant t_{10}. (c) Preference stream S until time instant t_{10}.

samples arriving at the system after instant t, that is, by considering a finite set $Test$ whose elements are of the form (u, v, t') with $t' \geq t$. Let $\mathcal{P}$ be the (non temporal) preference database obtained from $Test$ by removing the timestamp t' from its elements. The *acc* and *CR* measures of the preference model $PNet_t$ obtained at instant t are evaluated according to the formulae given in the previous section applied to the (static) BPN $PNet_t$ and the non temporal preference database $\mathcal{P}$. The quality of the algorithm is measured periodically over time. Fig. 2 illustrates this dynamic process of mining and testing of the preference models from the preference stream.

Now we are ready to state the problem of Mining Contextual Preferences from a Preference Stream:

Input: a relational schema $R(A_1, A_2, ..., A_n)$, and a preference stream S over R.
Output: whenever requested, return a BPN_t over R having good accuracy and comparability rate, where t is the time instant of the request.

4. ALGORITHMS FOR MINING PREFERENCES IN STREAMS

In this article we propose an algorithm for mining user contextual preferences in the stream setting: the *FPSMining Algorithm*. In addition, we also propose an incremental version of this algorithm, named *IncFPSMining Algorithm*. Our purpose is to compare the performance of these two algorithms over synthetic and real data, and evaluate whether we will achieve better results with the incremental version of the target algorithm of this article.

In order to save processing time and memory, in both algorithms we do not store the elements of the preference stream processed so far, we just collect *sufficient statistics* from it. In both algorithms, the sufficient statistics are *incrementally* updated in an online way for every new element that comes in the preference stream and the training process is carried out by extracting a preference model (a BPN) from these sufficient statistics. Example 4.1 illustrates the sufficient statistics collected from a preference stream.

Example 4.1. *Sufficient Statistics.* Let $R(A, B, C)$ be a relational schema with $a_1, a_2 \in \text{dom}(A)$, b_3, $b_5, b_6 \in \text{dom}(B)$ and $c_1, c_2, c_4, c_5 \in \text{dom}(C)$. Let S be a preference stream over R as shown in Fig. 3(c), where the T column stands for the time when the temporal bituple was generated, and $u_1 >_{t_i} u_2$ (u_1 is preferred to u_2 at t_i) for every temporal bituple (u_1, u_2, t_i) in the preference stream, for $1 \leq i \leq 10$. Consider the sufficient statistics for attribute C shown in Fig. 3(a) collected from the preference stream S until the time instant t_9. The table on top of Fig. 3(a) shows the *context counters* regarding the preferences on the values of the attribute C, and the table on the bottom shows the *general counters* over C. Context counters account for the possible causes for a particular preference over values of an attribute, and general counters stores the number of times that a particular preference over an attribute appeared in the stream. With the arrival of the temporal bituple $l = (u_1, u_2, t_{10})$, where $u_1 = (a_1, b_6, c_2)$ and $u_2 = (a_1, b_6, c_1)$, the sufficient statistics are updated as follows (see Fig. 3(b)):

(1) Compute $\Delta(u_1, u_2)$, which is the set of attributes where u_1 and u_2 differ in l. In this example, $\Delta(u_1, u_2) = \{C\}$, and so only the attribute C will have its statistics updated with the arrival of l; (2) Increment context counters a_1 and b_6 regarding the preference $c_2 > c_1$ (table on top of Fig. 3(b)). Notice that in the temporal bituple l the values a_1 and b_6 are possible contexts (causes) for the preference $c_2 > c_1$, just because they are equal in both tuples (u_1 and u_2). Since we had no context b_6 so far, it is inserted in the statistics; (3) Increment the general counter of the preference $c_2 > c_1$ (table on the bottom of Fig. 3(b)).

In order to limit the growth of the sufficient statistics, both algorithms perform a simple memory management procedure. Since the structure of the sufficient statistics has been specifically designed to operate in the problem addressed in this article, so it was necessary to develop a specific mechanism for the memory management of the proposed structure. In general, this mechanism can be described as follows. We can abstract our statistics as a tree, where the leaves are represented by general counters and context counters. For each leaf, we store the time of its last update. Periodically, in order to reduce runtime overhead, the algorithms perform the memory management of their statistics, which in short consists in eliminating leaves that have not been visited since a long time (i.e., the time of its last update differs from the current time by an amount greater than a threshold). When a leaf is removed, we verify if the nodes traversed to reach this leaf recursively need to be removed as well, in the case of these nodes do not have other children. According to tests carried out on synthetic data this mechanism proved to be effective.

4.1 The FPSMining Algorithm

The main idea of the FPSMining (**F**ast **P**reference **S**tream **Mining**) is to create a preference relation from the most promising dependencies between attributes of a preference stream. The *degree of dependence* of a pair of attributes (X, Y) is a real number (between 0 and 1) that estimates how preferences on values for the attribute Y are influenced by values for the attribute X. We adapted the concept of *degree of dependence* introduced by de Amo et al. [2013] to deal with sufficient statistics instead of a complete set of preferences. Its computation is carried out as described in Alg. 1. In order to facilitate the description of Alg. 1 we introduce some notations as follows: (1) We denote by $T^{time}_{yy'}$ the finite subset of temporal bituples $(u_1, u_2, t) \in S$, such that $t \leq time$, $(u_1[Y] = y \wedge u_2[Y] = y')$ or $(u_1[Y] = y' \wedge u_2[Y] = y)$; (2) We denote by $S^{time}_{x|(y,y')}$ the subset of $T^{time}_{yy'}$ containing the temporal bituples (u_1, u_2, t) such that $u_1[X] = u_2[X] = x$. Example 4.2 illustrates the computation of the degree of dependence on the statistics.

Algorithm 1: The degree of dependence of a pair of attributes

Input: Q: a snapshot of the sufficient statistics from the preference stream S at the time instant *time*; (X, Y): a pair of attributes; two thresholds $\alpha_1 > 0$ and $\alpha_2 > 0$.

Output: the degree of dependence of (X, Y) with respect to Q at the time instant *time*.

1 **for** *each pair* $(y, y') \in$ ***general counters*** *over* Y *from* Q, $y \neq y'$ *and* (y, y') *comparable* **do**

2 **for** *each* $x \in$ ***dom***(X) *where* x *is a cause for* (y, y') *being comparable* **do**

3 Let $f_1(S^{time}_{x|(y,y')}) = \max\{N, 1 - N\}$, where

$$N = \frac{|\{(u_1, u_2, t) \in S^{time}_{x|(y,y')} : u_1 >_t u_2 \wedge (u_1[Y] = y \wedge u_2[Y] = y')\}|}{|S^{time}_{x|(y,y')}|}$$

4 Let $f_2(T^{time}_{yy'}) = \max\{f_1(S^{time}_{x|(y,y')}) : x \in \mathbf{dom}(X)\}$

5 Let $f_3((X, Y), Q) = \max\{f_2(T^{time}_{yy'}) : (y, y') \in$ **general counters** over Y from Q, $y \neq y'$, (y, y') comparable$\}$

6 **return** $f_3((X, Y), Q)$

Example 4.2. *Degree of Dependence on the Statistics.* Let us consider the preference stream in Fig. 3(c) until instant t_{10} and the snapshot Q of its sufficient statistics for attribute C shown in Fig. 3(b). In order to compute the degree of dependence of the pair (A, C) with respect to the snapshot Q, we first identify the context counters related to A in Fig. 3(b). The thresholds we consider are $\alpha_1 = 0.1$

and $\alpha_2 = 0.2$. The support of (c_1, c_2) and (c_4, c_5) are $(4+3)/10 = 0.70$ and $3/10 = 0.30$, respectively. Therefore, we do not discard any of them. Entering the inner loop (line 2 of Alg. 1) for (c_1, c_2) we have only one set named $S_{a_1|(c_1,c_2)}$. The support of $S_{a_1|(c_1,c_2)}$ is $5/5 = 1.0$ and $N = 3/5$. Hence, $f_1(S_{a_1|(c_1,c_2)}) = 3/5$ and $f_2(T_{c_1c_2}) = 3/5$. In the same way, for (c_4, c_5) we have $S_{a_2|(c_4,c_5)}$ with support $2/2 = 1.0$ and $N = 2/2 = 1.0$. Therefore, $f_1(S_{a_2|(c_4,c_5)}) = 1.0$ and $f_2(T_{c_4c_5}) = 1.0$. Thus, the degree of dependence of (A, C) is $f_3((A, C), Q) = \max\{3/5, 1.0\} = 1.0$.

Given this, our algorithm builds a BPN_t from the preference stream using the Alg. 2.

Algorithm 2: The FPSMining Algorithm

Input: $R(A_1, A_2, ..., A_n)$: a relational schema; S: a preference stream over R.
Output: whenever requested, return a BPN_t over R, where t is the time instant of the request.

1 Take a snapshot Q of the sufficient statistics from S at the time instant t.
2 **for** *each pair of attributes* (A_i, A_j)*, with* $1 \leq i, j \leq n, i \neq j$ **do**
3 Use Alg. 1 for calculate the degree of dependence dd between the pair (A_i, A_j) according to Q
4 Let Ω be the resulting set of these calculations, with elements of the form (A_i, A_j, dd)
5 Eliminate from Ω all elements whose $dd < 0.5$ (indicates a weak dependence between a pair of attributes)
6 Order the elements (A_i, A_j, dd) in Ω in decreasing order according to their dd
7 Start the graph G_t of the BPN_t with a node for each attribute of R
8 **for** *each element* $(A_i, A_j, dd) \in$ *ordered set* Ω **do**
9 Insert the edge (A_i, A_j) in the graph G_t only if the insertion does not form cycles in G_t
10 Once the graph G_t of the BPN_t was created, estimate the conditional probabilities tables θ_t of the BPN_t, using the Maximum Likelihood Principle (see [de Amo et al. 2013] for details) over Q.
11 **return** BPN_t

4.2 The IncFPSMining Algorithm

The main idea of the IncFPSMining is the following: for each *chunk* of b temporal bituples arrived (parameter of the algorithm called "*grace period*") the current preference model M built so far is *updated*. This model M consists of a graph with some edges v_1,v_2,...,v_n, each one with degree of dependence dd measured at the time that M has been constructed. The *gap* of an edge v_i measures how close dd is from the minimum limit 0.5 that is $gap = dd - 0.5$. Only edges having *gap sufficiently high* are admitted at each update. The threshold is given by the *Hoeffding Bound* [Hoeffding 1963] ϵ associated to the random variable *gap*. It is computed as $\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}}$, where: 1) R is the size of the range of values of the random variable X associated to the problem considered (in our case, $X = gap$). Therefore, the higher value R of *gap* is 0.5 (so, $R = 0.5$); 2) δ is the probability that $X_{future} - X_{current} > \epsilon$; 3) n is the number of temporal bituples seen so far.

The Hoeffding Bound ensures (with an error probability δ) that, if the degree of dependence dd_t of an edge v at instant t satisfies $dd_t - 0.5 \geq \epsilon$, when the number of temporal bituples seen so far was n, then in any future instant t_{fut} its degree of dependence dd_{fut} must satisfy $(dd_{fut} - 0.5) - (dd_t - 0.5) \leq \epsilon$. That is $dd_{fut} - dd_t \leq \epsilon$, and so, dd_{fut} is not very far from the acceptable degree of dependence at the current instant t. So, the edges that were introduced in an earlier phase will not have their dd get closer to the limit 0.5 than before. The example 4.3 illustrates this process.

Example 4.3. *Hoeffding Bound Guarantee.* Let us consider the Hoeffding Bound $\epsilon = 0.02$ and let us suppose a selected edge v at instant t having $dd_t = 0.55$. Then the *gap* $(0.55 - 0.5)$ at t is $0.05 > 0.02$, considered reasonable. So in any future instant t_{fut}, the gap dd_{fut} will not change much, i.e., $FutGap = (dd(t_{fut}) - 0.5)$ may not be very far from $CurrGap = (0.55 - 0.5)$. The Hoeffding Bound guarantees that this gap will satisfy: $CurrGap - FutGap \leq \epsilon$. Therefore $FutGap \geq CurrGap - 0.02 = 0.05 - 0.02 = 0.03$, which is quite acceptable.

This algorithm only considers the statistics related to edges that have not been inserted in the graph so far. Thus, we first select edges not belonging to the current graph, whose dd verifies the Hoeffding

Bound condition ($gap = dd - 0.5 \geq \epsilon$). For each one of these edges we test if its inclusion produces cycles in the graph. If so, we evaluate the dd of all edges in the cycle, and eliminate the edge with the worst dd. The IncFPSMining algorithm is described in Alg. 3.

Algorithm 3: The IncFPSMining Algorithm

Input: $R(A_1, A_2, ..., A_n)$: a relational schema; S: a preference stream over R.
Output: whenever requested, return a BPN_t over R, where t is the time instant of the request.

```
1  Let G_t be a graph with a node for each attribute of R
2  for each temporal bituple l of S do
3      Increment n, the number of elements seen until t
4      if n mod grace period = 0 then
5          Compute Hoeffding bound ε
6          Take a snapshot Q of the statistics from S at t
7          for each possible edge e_i outside G_t do
8              Use Alg. 1 for calculate the degree of dependence dd of e_i according to Q
9          Let Ω be the resulting set of these calculations, with elements of the form (e_i, dd)
10         Order the elements (e_i, dd) in Ω in decreasing order according to their dd
11         for each pair (e_i, dd) ∈ ordered set Ω do
12             if dd − 0.5 ≥ ε then
13                 Insert the edge e_i in G_t
14                 if e_i has created cycle in G_t then
15                     Remove from G_t the edge with lower dd in the cycle
16         Once G_t of the BPN_t was created, estimate the tables θ_t of the BPN_t over Q.
```

4.3 Complexity Analysis of Both Algorithms

We divided the complexity analysis of both algorithms in three parts, as explained below.
–*The complexity of processing each element of the stream and update the sufficient statistics*: (i) Scan a temporal bituple to save the Δ (set of attributes that have different values in the two tuples) and the B (set of attributes that have equal values in the two tuples) are $O(l)$, where l is the number of attributes; (ii) Update the statistics of an attribute is $O(|B|)$. Thus, update the statistics for all attribute in Δ is $O(|\Delta|.|B|)$. Therefore, the complexity for processing each element in both algorithms in the worst case is $O(|\Delta|.|B|)$, where $|\Delta| + |B| = l$, and in the best case is $O(l)$.
–*The complexity of creating the preference model (BPN)* in the FPSMining algorithm is calculated as follows: (i) the computation of the degree of dependence between the pairs of attributes is $O(l^2)$; (ii) the computation of the topology is $O(l^2.e)$, where e is the number of edges of the BPN. The sub-steps used in this calculation were: (a) the elimination of the pairs of attributes whose $dd < 0.5$ is $O(l^2)$, (b) the ordination of the pairs of attributes is $O(l^2.\log l)$ in the average case, and (c) for each pair of attributes the acyclicity test of the insertion of its edge in the graph is $O(e)$, so the total complexity of this sub-step is $O(l^2.e)$; (iii) the computation of the conditional probability tables is $O(e.m)$, where m is the number of training instances processed. Therefore, the total complexity for building the model in the FPSMining algorithm is $O(e.(l^2 + m))$.The complexity of updating the preference model in the IncFPSMining algorithm is made analogously, and in the worst case is also $O(e.(l^2 + m))$. However, over time, the IncFPSMining will consider only a few edges in its processing, differently from the FPSMining which always considers all possible edges. Thus, in the practice, as the more attributes the stream has, the greater the advantage of the IncFPSMining over the FPSMining;
–*The complexity of using the model (BPN)* for ordering a temporal bituple in both algorithms is $O(l + e)$, which can be reduced to $O(l)$ when the BPN is a sparse graph.

5. EXPERIMENTAL RESULTS

In this section we describe the results concerning the performance of the FPSMining and IncFPSMining over synthetic and real datasets. Our algorithms were implemented in Java and all the experiments

Table I. The configuration parameters in the tests with synthetic data

# attributes	|dom| per attribute	|stream|
6, 8, 10, 12, 14, 16	10 elements	10m, 25m, 50m, 75m, 100m

performed on a Windows 7 machine with 3.40 GHz clocked processor and 12 GB RAM.

The Experiments Protocol. In order to evaluate our algorithms, we adapted the sampling technique proposed by Bifet et al. [2011] (based on holdout for data stream) to the preference mining scenario. This sampling technique takes three parameters as input : n_{train}, n_{test} and n_{eval}. The n_{train} and n_{test} variables represent, respectively, the number of elements in the stream to be used to train and test the model at each evaluation. The variable n_{eval} represents the number of evaluations desired along the stream. For example, let us consider the values[1] n_{train} = 10k, n_{test} = 1k and n_{eval} = 9090. Consider $S = \{e_1, e_2, e_3, ...\}$ the preference stream used in the tests. The dynamic of the evaluation for this example is as follows: (1) elements e_1 to e_{10k} from S are used to train the model; (2) elements e_{10001} to e_{11k} are used to test the quality of the model. The *acc*, *CR* and *prec* of the model are calculated according to this test period; (3) elements e_{11001} to e_{21k} are used to train the model, and so on for 9090 cycles. The preference models are produced by both algorithms at each chunk of n_{train} temporal bituples arrived. Let us call this moment of *refresh point*. At each refresh point the FPSMining produces its model from scratch whereas IncFPSMining updates the model that has been incrementally generated (at each grace period) so far.

Synthetic Data. In order to evaluate the ability of our algorithms to deal with large volumes of data, we performed a set of experimental tests on synthetic data. The synthetic data[2] were generated by an algorithm based on Probabilistic Logic Sampling [Jensen and Nielsen 2007], which samples bituples for a preference stream S given a BPN with structure G and parameters θ. We have considered different structures G (varying the number of nodes) and parameters θ using the configuration shown in the Table I. We performed tests with streams up to 100 million[3] elements. The default values for the holdout parameters were $n_{train} = 10k$ and $n_{test} = 1k$. For the tests with 10m, 25m, 50m, 75m and 100m of elements, we used $n_{eval_{10m}} = 909$, $n_{eval_{25m}} = 2272$, $n_{eval_{50m}} = 4545$, $n_{eval_{75m}} = 6818$ and $n_{eval_{100m}} = 9090$. For calculate the *degree of dependence*, we used $\alpha_1 = 0.2$ and $\alpha_2 = 0.1$. Besides, the default values for IncFPSMining were grace period = 1k and $\delta = 10^{-7}$.

In our experiments, we consider the statistical test proposed by Tan et al. [2005] to detect statistical significance in the differences of the algorithms performance. For this approach, we need to use a t-distribution to compute the confidence interval for the true difference between the algorithms: $d_t^{cv} = \overline{d} \pm \gamma$, where $\overline{d}$ is the average difference of the experiment, $\gamma = t_{(1-\alpha),n_{eval}-1} \times \hat{\sigma}_{d^{cv}}$ and $\hat{\sigma}_{d^{cv}}$ is the standard deviation. The coefficient $t_{(1-\alpha),n_{eval}-1}$ is obtained from a probability table with two input parameters, its confidence level $(1-\alpha)$ and the number of degrees of freedom $(n_{eval}-1)$.

1. Performance Analysis. Fig. 4(a) and (b) show the average values of *acc*, *CR* and *prec* obtained with FPSMining and IncFPSMining for streams with different number of attributes and size, respectively. In the Fig. 4(a) we used 50m elements and in the Fig. 4(b) we used 10 attributes. The quality of both algorithms remained stable over the different streams. Furthermore, our two algorithms could compare practically all the bituples of tests they were submitted ($\overline{CR}$ columns). In these tests we also calculated the statistical significance regarding the slight improvement presented by IncFPSMining compared to FPSMining. Our question is: At $\alpha = 95\%$, can we conclude that the IncFPSMining outperforms FPSMining? The null and alternate hypotheses for *acc* and *CR* are $H0 : IncFPS \leq FPS$ and $HA : IncFPS > FPS$. The results show that $H0$ is rejected, and thus, HA is substantiated. Thus, although the difference between the two algorithms is very small, it is statistically significant. Fig. 4(c) illustrates how *acc* of the FPSMining evolve over time. We choose the FPSMining for this test

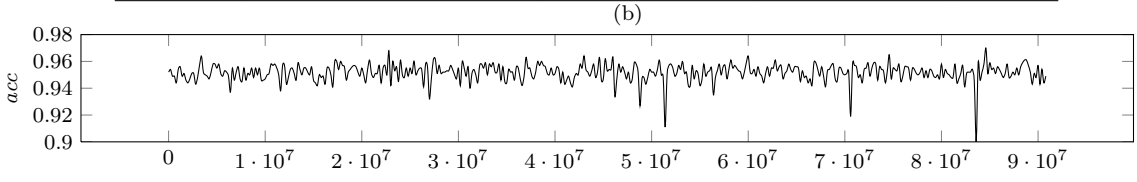
[1] 1k = 1000.
[2] Available at `http://lsi.facom.ufu.br/JIDM2014-SyntheticData/`
[3] We often abbreviate *million* by *m* in the text.

# attributes	IncFPS. $\overline{acc}$	IncFPS. $\overline{CR}$	IncFPS. $\overline{prec}$	FPS. $\overline{acc}$	FPS. γ	FPS. $\overline{CR}$	FPS. γ	FPS. $\overline{prec}$
6	0.95168	1.000	0.95168	0.94915	2×10^{-4}	0.997	2×10^{-4}	0.95160
8	0.94936	1.000	0.94936	0.94322	4×10^{-4}	0.993	4×10^{-4}	0.94942
10	0.95239	1.000	0.95239	0.95162	1×10^{-4}	0.999	1×10^{-4}	0.95239
12	0.95059	1.000	0.95059	0.94831	3×10^{-4}	0.997	3×10^{-4}	0.95057
14	0.95053	1.000	0.95053	0.94858	2×10^{-4}	0.997	3×10^{-4}	0.95053
16	0.95115	1.000	0.95115	0.94932	2×10^{-4}	0.998	2×10^{-4}	0.95115

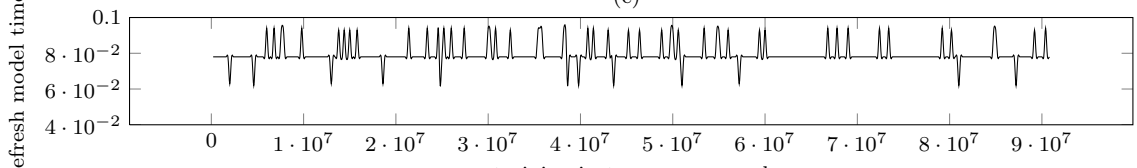
(a)

|stream|	IncFPS. $\overline{acc}$	IncFPS. $\overline{CR}$	IncFPS. $\overline{prec}$	FPS. $\overline{acc}$	FPS. γ	FPS. $\overline{CR}$	FPS. γ	FPS. $\overline{prec}$
10m	0.95236	1.000	0.95236	0.95151	3×10^{-4}	0.999	3×10^{-4}	0.95238
25m	0.95243	1.000	0.95243	0.95165	1×10^{-4}	0.999	1×10^{-4}	0.95244
50m	0.95239	1.000	0.95239	0.95162	1×10^{-4}	0.999	1×10^{-4}	0.95239
75m	0.95235	1.000	0.95235	0.95155	1×10^{-4}	0.999	1×10^{-4}	0.95235
100m	0.95240	1.000	0.95240	0.95160	9×10^{-5}	0.999	9×10^{-5}	0.95240

(b)



(c)



(d)

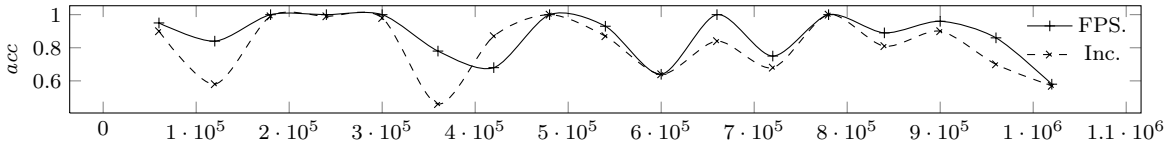
Fig. 4. Experimental Results on Synthetic Data

because it has the lower limit quality of our algorithms. In this experiment, we consider the stream with 100m of elements and 10 attributes. This curve shows that the *acc* values of the FPSMining were reasonably stable over time.

2. Execution Time. Fig. 4(d) shows the time measured in seconds taken by FPSMining to generate the model at each refresh point. The same data used in the experiment (c) have been considered here. This stream produces about 20 GB of data. Notice that the time to refresh the model is very small, on the order of milliseconds. Though the FPSMining builds the entire BPN every holdout cycle, the sufficient statistics are updated incrementally, i.e., they are always ready to use, which makes this process fast. For this test we used the memory management explained in the previous section. We carried out an analysis regarding the time required to generate the model in two scenarios: (1) without the memory management of the sufficient statistics; (2) with the memory management. In (1) the time taken to generate the model has increased according to the increase of the number of training instances processed, whereas in (2) the time required to generate the model has remained constant over time, as can be seen in the Fig. 4(d). This scenario highlights the necessity of using the memory management of the sufficient statistics.

Real Data. In order to evaluate our algorithms over real-world datasets, we considered data containing preferences related to movies collected by GroupLens Research[4] from the MovieLens web site[5] concerning six different users (named U_i, for $i = 1, 2, 3, 4, 5, 6$). We simulated preference streams from

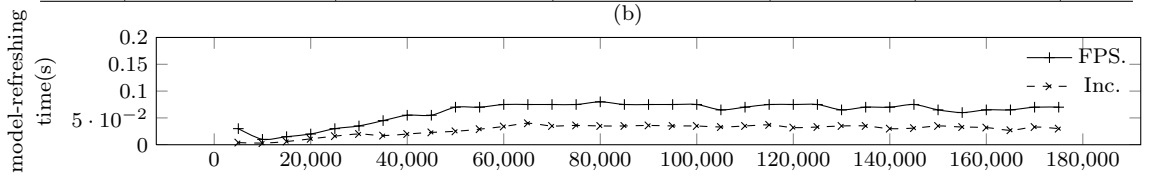
[4]Available at `http://www.grouplens.org/taxonomy/term/14`
[5]Available at `http://movielens.umn.edu`



(a)

			IncFPS.			**FPS.**			**Hoeffding Tree**				
User	Tuples	Bituples	$\overline{acc}$	$\overline{CR}$	$\overline{prec}$	$\overline{acc}$	$\overline{CR}$	$\overline{prec}$	$\overline{acc}$	γ	$\overline{CR}$	γ	$\overline{prec}$
U_2	7046	178,500	0.58	0.82	0.70	0.58	0.82	0.70	0.41	0.03	0.66	0.05	0.63
U_3	4449	112,200	0.60	0.90	0.66	0.60	0.90	0.66	0.33	0.03	0.63	0.03	0.52
U_4	4165	102,000	0.59	0.94	0.62	0.59	0.94	0.62	0.31	0.05	0.55	0.08	0.48
U_5	3414	86,700	0.61	0.90	0.69	0.61	0.90	0.69	0.36	0.06	0.60	0.08	0.52
U_6	3164	76,500	0.61	0.91	0.66	0.61	0.91	0.66	0.39	0.07	0.62	0.07	0.58

(b)



(c)

Fig. 5. Experimental Results on Real Data

these data, as follows: we stipulated a time interval λ, and each tuple in the dataset D_i of movies evaluated by the user U_i was compared to all others movies of D_i in a radius λ relative to its timestamp, thus generating temporal bituples for each user U_i. We calculated the λ (in days) for each user U_i according to the sparsity related to the time of the tuples in the dataset D_i, and the values obtained were: $U_1 = 77$, $U_2 = 37$, $U_3 = 140$, $U_4 = 50$, $U_5 = 69$ and $U_6 = 135$. The resulting preference stream S_i has five attributes (director, genre, language, star and year), and its elements correspond to preferences on movies concerning user U_i. Dataset D_1 is constituted by movies evaluated by the user U_1 from 5th Aug 2001 to 3rd Jan 2009. The evaluation time periods for the other users are: from 10th Oct 2006 to 2nd Jan 2009 for user U_2, from 3rd Aug 2000 to 1st Jan 2009 for user U_3, from 13th Nov 2006 to 5th Jan 2009 for user U_4, from 5th Jul 2002 to 5th Jan 2009 for user U_5 and from 17th Jul 2002 to 18th Dec 2008 for user U_6. The default values for the holdout parameters were $n_{train} = 5k$ and $n_{test} = 100$. The n_{eval} values for each user U_i were calculated as follows: $n_{eval_{Ui}} = |D_i| \ / \ (n_{train} + n_{test})$. The default values for α_1 and α_2 (*degree of dependence*) were: 0.2 and 0.1 respectively. For IncFPSMining the default grace period was $1k$ and $\delta = 10^{-7}$.

1. Performance Analysis. Fig. 5(a) illustrates how the *acc* of the FPSMining and IncFPSMining algorithms evolve through time. In this experiment, we consider the dataset D_1 related to user U_1. The total number of training instances processed by both algorithms was close to 1.1×10^6, corresponding the entire stream of evaluations made by the user U_1. Although most of the time FPSMining shows better predicting capability than IncFPSMining, the difference between the performances of both algorithms is not very expressive.

2. Baseline. As will be discussed in section 6, we did not find any published algorithm that addresses the exact same problem we address. Nevertheless, the classification task is the closest to the mining preference task among the numerous existing data mining tasks. So, we decided to design a baseline by adapting a classifier to compare the performance of our algorithms. In this approach, the classes are the ratings given by the users to the movies and can take the values: 1, 2, 3, 4 or 5. We designed this baseline so that in each cycle of the holdout, the sets of training and testing samples of our algorithms contain the same movies used by the classifier. This ensures a fair evaluation process. Fig. 5(b) compares the performance of our algorithms with a baseline widely used in the literature: the Hoeffding Tree algorithm [Domingos and Hulten 2000]. For these experiments, we used the MOA

[Bifet et al. 2010] implementation of Hoeffding Tree (HT) in the default configuration parameters[6]. In these tests we used the users U_2, U_3, U_4, U_5 and U_6. Regarding our algorithms, we performing tests with both hypotheses: 1) assuming that FPSMining is strictly better than IncFPSMining and 2) vice versa. Both hypotheses were rejected, which leads us to conclude that our two algorithms have tied in these data. Thus, for these experiments the main question is: At $\alpha = 95\%$, can we conclude that FPSMining and IncFPSMining outperform the baseline? The null and alternate hypotheses for *acc* and *CR* are $H0 : FPS, IncFPS \leq HT$ and $HA : FPS, IncFPS > HT$. The results show that $H0$ is rejected, and thus, HA is substantiated. This shows that our two algorithms outperform the baseline. Thus, we can conclude that our algorithms, which were specifically designed for preference mining, perform better than a classical classifier.

3. Execution Time. Fig. 5(c) shows the time measured in seconds taken by the FPSMining and IncFPSMining algorithms to generate the model at each refresh point. In this experiment, we consider the dataset D_2 related to user U_2. Notice that the time to refresh the model is very small, on the order of milliseconds. Besides, the time taken by the IncFPSMining algorithm to generate the model is almost half of the time taken by FPSMining, mainly due to incremental building of it. Notice also that the time of both algorithms remains almost constant with increasing number of training instances processed, mainly due to the efficiency of the memory management used.

6. RELATED WORK

There is a vast literature available on preference learning in the batch setting [Holland et al. 2003; Jiang et al. 2008; de Amo et al. 2013]. Most articles propose a preference learning algorithm, a method to elicit user preferences or even on a formalism about preferences. de Amo et al. [2013] proposes CPrefMiner, which is a technique for learning contextual user preferences in the batch setting.

Meanwhile, proposals for suitable algorithms to solve the problem of learning user preferences in streams have been little explored in the literature. To the best of our knowledge there is no work in the literature addressing this specific topic. Learning techniques have been used to efficiently predict user preferences in Context-based Recommendation Systems [Lops et al. 2011] or in Hybrid Recommendation Systems [Burke 2002]. These techniques are used to mitigate the well-know item-cold start challenge faced in recommendation systems research. However, they are usually classical classification algorithms used to predict *user evaluations on individual items* (as in [Melville et al. 2002], designed for a batch setting) or specific algorithms to predict *item rankings* from a set of item-ranking samples provided by a user (as in [Shivaswamy and Joachims 2011] also designed for a batch setting). We have found no work focused on *pairwise* preference mining algorithms in a data stream setting, that is, algorithms designed to extract a preference model from a set of time-stamped pairs of items. For that reason, we decided to adapt a well-known classification technique designed to predict user evaluations on items in a data stream setting, the Hoeffding Tree algorithm [Domingos and Hulten 2000] in order to predict the preference order between pairs of objects. This algorithm is used in Section 5 as baseline for FPSMining and IncFPSMining.

Other work related to ours is [Jembere et al. 2007], that presents an approach to mine user preferences in an environment with multiple context-aware services, but uses incremental learning only for the context, and not for the user's preferences.

7. CONCLUSION AND FURTHER WORK

In this article we proposed two algorithms to solve the problem of mining user contextual preferences in a stream setting. The first algorithm, FPSMining, has been introduced in a previous short version of this article in [Papini et al. 2013]. It does not verify one important precept pointed out by Kontaki

[6]Hoeffding Tree: gracePeriod $g = 200$, splitConfidence $c = 10^{-7}$, tieThreshold $t = 0.05$, numericEstimator $n = GAUSS10$.

et al. [2010] for efficient data streams applications, the incremental evaluation. So, the second algorithm IncFPSMining introduced in this extended version aimed at proposing an incremental technique for mining preferences in data stream. A baseline method adapted from the classical classification algorithm Hoeffding Tree [Domingos and Hulten 2000] has also been implemented. We remark that most existing Content-based Recommendation Systems use classification algorithms in order to predict user preferences. An extensive series of experiments executed on synthetic and real data showed that our algorithms are very efficient and largely outperform the baseline. However, contrarily to our expectation, the incremental algorithm did not show a superior performance when compared to the non-incremental one. As immediate future work, we intend to develop different methods for mining preferences from streams where concept drift occurs in the preference data along time. In fact, this situation is expected in real world applications, where training data are collected during a large period of time. That is, it seems natural to think of users preferences as not static. A very preliminary proposal in this direction has already been developed and the initial tests allow us to conjecture that in a scenario where concept drift is explicitly inserted in the stream, the superiority of the incremental algorithm will be surely evident.

REFERENCES

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. MOA: massive online analysis. *Journal of Machine Learning Research* 11 (1): 1601–1604, Aug., 2010.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. Data stream mining: a practical approach. Tech. rep., The University of Waikato, 2011.

Burke, R. Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction* 12 (4): 331–370, 2002.

Crammer, K. and Singer, Y. Pranking with ranking. In *Neural Information Processing Systems (NIPS)*. MIT Press, Vancouver, British Columbia, Canada, pp. 641–647, 2001.

de Amo, S., Bueno, M. L. P., Alves, G., and da Silva, N. F. F. Mining user contextual preferences. *Journal of Information and Data Management (JIDM)* 4 (1): 37–46, 2013.

Domingos, P. and Hulten, G. Mining high-speed data streams. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, NY, USA, pp. 71–80, 2000.

Gama, J. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, Minneapolis, Minnesota, USA, 2010.

Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58 (301): 13–30, March, 1963.

Holland, S., Ester, M., and Kiessling, W. Preference mining: a novel approach on mining user preferences for personalized applications. Tech. rep., Universitätsbibliothek der Universität Augsburg, Augsburg, 2003.

Jembere, E., Adigun, M. O., and Xulu, S. S. Mining context-based user preferences for m-services applications. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, Washington, DC, USA, pp. 757–763, 2007.

Jensen, F. V. and Nielsen, T. D. *Bayesian Networks and Decision Graphs*. Springer Publishing Company, Incorporated, New York, NY, USA, 2007.

Jiang, B., Pei, J., Lin, X., Cheung, D. W., and Han, J. Mining preferences from superior and inferior examples. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, pp. 390–398, 2008.

Kontaki, M., Papadopoulos, A., and Manolopoulos, Y. Continuous processing of preference queries in data streams. In *Proceedings of the International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 47–60, 2010.

Lops, P., de Gemmis M., and Semeraro, G. Content-based recommender systems: state of the art and trends. In *Recommender Systems Handbook*. Springer, USA, pp. 73–105, 2011.

Melville, P., Mooney, R. J., and Nagarajan, R. Content-boosted collaborative filtering for improved recommendations. In *National Conference on Artificial Intelligence*. AAAI, CA, USA, pp. 187–192, 2002.

Papini, J. A. J., de Amo, S., and Soares, A. K. S. FPSMining: a fast algorithm for mining user preferences in data streams. In *28th Brazilian Symposium on Databases (SBBD 2013- Short Paper Session)*. Brazilian Computer Society, Recife, Brazil, pp. 61–66, 2013.

Shivaswamy, P. K. and Joachims, T. Online learning with preference feedback. *CoRR* vol. 1111.0712, pp. 1–5, 2011.

Tan, P.-N., Steinbach, M., and Kumar, V. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

Wilson, N. Extending cp-nets with stronger conditional preference statements. In *Proceedings of the National Conference on Artifical Intelligence*. AAAI Press, San Jose, California, pp. 735–741, 2004.