Clustering Individual Transactional Data for Masses of Users

Riccardo Guidotti
ISTI-CNR & University of Pisa, Italy
riccardo.guidotti@isti.cnr.it

Anna Monreale
University of Pisa, Italy
anna.monreale@di.unipi.it

Mirco Nanni
ISTI-CNR, Pisa, Italy
mirco.nanni@isti.cnr.it

Fosca Giannotti
ISTI-CNR, Pisa, Italy
fosca.giannotti@isti.cnr.it

Dino Pedreschi
University of Pisa, Italy
dino.pedreschi@di.unipi.it

ABSTRACT

Mining a large number of datasets recording human activities for making sense of individual data is the key enabler of a new wave of personalized knowledge-based services. In this paper we focus on the problem of clustering individual transactional data for a large mass of users. Transactional data is a very pervasive kind of information that is collected by several services, often involving huge pools of users. We propose *txmeans*, a parameter-free clustering algorithm able to efficiently partitioning transactional data in a completely automatic way. *Txmeans* is designed for the case where clustering must be applied on a massive number of different datasets, for instance when a large set of users need to be analyzed individually and each of them has generated a long history of transactions. A deep experimentation on both real and synthetic datasets shows the practical effectiveness of *txmeans* for the mass clustering of different personal datasets, and suggests that *txmeans* outperforms existing methods in terms of quality and efficiency. Finally, we present a *personal cart assistant* application based on *txmeans*.

1 INTRODUCTION

The most disruptive effect of our always-connected society is *data*, the digital breadcrumbs left behind us as a side effect of our everyday usage of digital technologies. Thanks to these data, human activities are becoming observable, measurable, quantifiable and, predictable. At individual level, each person generates more than 5Gb of data per year. An avalanche of information that, for the most part, consists of *transactions* (or baskets), i.e., a special kind of categorical data in the form of sets of event data, such as the items purchased in a shopping cart, the web pages visited in a browsing session, the songs listened in a time period, the clinical events in a patient's history. Such kind of data may be key enablers of a new wave of knowledge-based services, and of new scientific discoveries.

Several application contexts involve the analysis of a large number of datasets, each one characterized by different properties. For instance, this is the case of individual transactional data – retail sales, web sessions, credit card transactions, etc. – where each user produces historical data that need to be analyzed separately from other users. This requires that a parameter-tuning phase is included in any data mining method we want to apply, driven by the necessity to automatically capture the wide diversity of individual behaviors. Due to the potentially large number of datasets (e.g. users in nowadays massive systems scale up to billion), it is generally unfeasible to determine in an ad-hoc manner the optimal parameter configuration for each of them. Therefore, we need *auto-focus* data mining methods that adjust their parameter setting to the characteristics of the dataset under analysis, to the aim of extracting personalized patterns from transactional data of each user [17].

In this paper we focus on the problem of performing transactional clustering for a large number of different datasets. Given a collection of transactions, transactional clustering consists in discovering groups of homogeneous transactions which share many common items [30]. In the state of the art, all the methods for transactional clustering require either a parameter tuning process that is not automatic, or an extremely heavy automated process that does not scale to large user bases [3, 12, 13, 30, 36]. Hence, repeatedly applying the existing procedures on thousands or millions of different datasets – which is the case when dealing with a large population of users – is simply unfeasible. We refer to this problem, i.e., the separate individual clustering of many individual transactional datasets, as *mass clustering*.

The problem to design parameter-free clustering algorithms has been addressed in the context of non transactional data by solutions like *xmeans* [22], which are perfect for solving many instances of the clustering problems. Unfortunately, they are not directly applicable to transactional data. To the best of our knowledge, the only existing parameter-free transactional clustering algorithms are [5, 7]. Nevertheless, they are based on a scanning schema which is generally not efficient and overestimates the real number of clusters. In addition, they do not provide representative transactions, i.e., the items that characterize the transactions contained in each cluster.

In this paper we propose *txmeans*, a new parameter-free clustering method providing a viable solution to the problem of clustering a massive number of different datasets. Our proposal follows a strategy similar to *xmeans* [22], but is designed and improved for finding clusters in the specific context of transactional data.

Txmeans overcomes the deficiencies of existing methods, indeed, it automatically estimates the number of clusters and, besides extracting the clusters, it provides the *representative transaction* of each cluster, which summarizes the pattern captured by that cluster. *Txmeans* employs a top-down divisive approach which starts from a unique cluster, and then iteratively splits the cluster into two sub-clusters. *Txmeans* calculates the representative baskets as the centroids of the sub-clusters by adopting a procedure described in

KDD '17, August 13-17, 2017, Halifax, NS, Canada

DOI: 10.1145/3097983.3098034

[12]. The representative baskets enable *txmeans* to decide through the *Bayesian Information Criteria* [24] (adapted to transactional data) whether the cluster under analysis needs further partitioning or not. The capability to adapt to specific individual datasets combined with a high efficiency put *txmeans* in the position of outperforming all competitors when transactional clustering must be applied on a very large number of different datasets, both in terms of clusters quality and in terms of running time.

We extensively validate our method using a large array of synthetic and real datasets. *Txmeans*' performance shows stability across diversified situations, including noise and varying clustering structures, and it scales to large datasets. The results suggest that *txmeans* is the best approach for the clustering of a massive number of datasets, but also for the clustering of a single individual dataset *txmeans*'s performance are significantly better than those of the competitors. The proposed approach enables sophisticated processes, based on individual clustering, to offer personalized services such as prediction-based and recommendation-based services.

As final result, we show how to exploit the individual clusters and representative transactions to build a *personal cart assistant* that suggests to the customer the items to put in her shopping list.

The paper is organized as follows. Section 2 discusses the related work. Section 3 defines the problem setting and Section 4 describes our clustering method. Section 5 shows a deep experimentation, while Section 6 illustrates *txmeans* in a recommender system application. Finally, Section 7 concludes the paper.

2 RELATED WORK

In the literature there is a great variety of papers proposing approaches for clustering transactional data. Most existing algorithms require the setting of different parameters which often may be difficult to tune. The first algorithm proposed for transactional clustering is *large item* [30]. It requires a support threshold indicating the minimum number of occurrences for an item to be *large* for a cluster. Through a scan of the transactions, it evaluates a global cost function and decides the cluster for that transaction, or creates a new one. Algorithms like *rock* and *clope* were proposed with the same scanning strategy but different cost functions [13, 35, 36].

The most common parameter is the number of clusters k [12, 15, 32, 37]. *Tkmeans* [12] is an example of this category, and also represents the first attempt to use a clustering strategy – basically the standard *k-means* [27] – different from direct optimization of a cost function. Also our *txmeans* does not minimize a cost function, and it inherits from *tkmeans* the method to extract cluster centroids. Another approach able to extract cluster centroids, but only for categorical dataset, is *k-modes* [6] which employs the mode instead of the mean. A recent approach with a different purpose is proposed in [9]: *purtreeclust* is a method for clustering customers through their purchase trees which are built on the customers transactions. The *txmeans* method can recall *purtreeclust* in the construction of a tree to extract the clusters, however the measure used to perform the split is different, and the predictive clustering trees works only on classical categorical data and not on transactional data.

A further notion widely used in cost functions to measure the similarity of data objects is entropy [2, 3, 8, 19]. In [3] is proposed *coolcat* that iteratively chooses the suitable cluster for each transaction by minimizing the entropy at each step. A similar procedure is followed by the algorithm *limbo* [2]. A dual approach is proposed in [19] where starting from a single cluster a Monte Carlo process is used to select a transaction and to assign it to another cluster to decrease the entropy. The process is repeated until convergence.

Besides parameter tuning, when dealing with transactional data another problem is high dimensionality: it makes algorithms inefficient in terms of execution time and clustering quality. To not suffer from high dimensionality, *subspace* clustering algorithms like [11, 38] have been proposed, with the goal of finding clusters embedded in subspaces of the original data with their own dimensions. Algorithms like [4, 10] use bipartite graph theory to cluster datasets. They generate co-clustering results where columns and rows are simultaneously partitioned. In transactional data this means an unnatural split of the clusters that overlap over a few frequent items. Moreover, they are often memory and time consuming, and inappropriate for clustering large datasets.

Some proposals were made to overcome manual parameter tuning. [8] introduces an entropy-based clustering evaluating the similarity with incremental entropy, and finally, generating a clustering tree containing clusterings with different numbers of clusters. The authors of [34] propose to run their algorithm with different numbers of clusters and choose the result that optimizes a novel index of quality. In terms of execution time this method is clearly inefficient.

Atdc [7] is a parameter-free transactional clustering algorithm which, similarly to our method, adopts a top-down strategy resembling a decision tree algorithm. [5] proposes the *practical* parameter-free method that through scanning automatically identifies clusters even in presence of rare items. Finally, also the *dhcc* algorithm presented in [33] is a parameter-free procedure based on a divisive hierarchical clustering approach. However, *dhcc* is especially designed to work on categorical rather than transactional data.

The main difference between these methods and *txmeans* is the ability to extract centroids, i.e., representative itemsets for each cluster. Moreover, among all the methods in the literature only *txmeans* and *practical* are able to deal with rare and very common items: *rare items* lead algorithms to return singleton clusters, while *very common items* incorrectly causes the merge of different clusters.

3 PROBLEM SETTING

In this section we define the context and the problem we want to solve, i.e., the *mass transactional clustering* problem.

Let $B = \{b_1, \ldots, b_N\}$ be a set of N baskets (or transactions) and $I = \{i_1, \ldots, i_D\}$ a set of D items., a *basket* b_i is defined as a subset of items such that $\emptyset \subset b_i \subseteq I$. Datasets of transactions B are usually denoted as *transactional data*, and represent a special case of high-dimensional categorical data. Indeed, each transactional dataset can be represented as categorical by representing each attribute value as a boolean attribute in I. In the following, we refer to transactional data in the sense of *high-dimensional* categorical datasets which in turns can also be considered as *sparse* categorical datasets.

Definition 3.1 (Transactional Clustering). Given a set of baskets B, we define the *transactional clustering* problem as the partitioning of B into K disjoint sets $\mathbb{C} = \{C_1, \ldots, C_K\}$ such that $\mathbb{C}$ is optimal in terms of homogeneity and simplicity, and for each set C_i a representative transactions r_i is provided.

This means that the baskets in each C_i must exhibit a high degree of overlap in comparison to any transaction in $B \setminus C_i$, while keeping the clustering structure concise. Notice that, in general, the subset $I_i \subseteq I$ of a cluster is not disjoint to the subset $I_j \subseteq I$ of other clusters.

When dealing with applications that involve a large number of logically separated datasets, there is the need of solving many instances of the transactional clustering problem. For instance, each dataset might contain the baskets of a different user and we want to analyze each user separately in order to profile her.

Definition 3.2 (Mass Transactional Clustering). Given a set of users $U = \{u_1, \ldots, u_M\}$ each one with her personal set of baskets $\mathcal{B} = \{B_{u_1}, \ldots, B_{u_M}\}$, we refer to *mass transactional clustering* as the problem which consists in solving the transactional clustering problem individually on each set B_{u_i} of baskets of each user $u_i \in U$.

Since the number of users $u \in U$ can be very large in real applications, the above problem definition implies some technical requirements on the methods aimed to solve it. First, the clustering of each different user dataset B_{u_i} can yield a different number K_i of clusters, which needs to be automatically determined, since a repeated intervention of an expert manually setting the K_i for each dataset is impractical. Second, since at least M runs of the clustering method are needed, i.e., one for each user dataset B_{u_i}, the algorithm needs to be efficient. Also, each dataset B_{u_i} can be large, depending on the application and the temporal period covered by the data, thus the algorithm needs to be scalable and applicable to big data. Third, in the context of transactional data an important semantic feature is the notion of *cluster representative*, that is a transaction that should represent the characteristics of the transactions belonging to the cluster. Producing such kind of cluster summary is a significant plus, since it provides a simple way to explain the content of the clusters, and can also support various useful personal applications, among which the *personal cart assistant* application described in Section 6 represents a typical example.

While each of the above requirements is satisfied by some existing algorithm, there is no method meeting all of them together. This paper introduces an algorithm able to do it, also being competitive against the state-of-art competitors on each single requirement.

4 PROPOSED METHOD

In this section we describe the components of *txmeans*. Inspired by *xmeans* [22], *txmeans* is a hierarchical divisive clustering method based on iterative bisections with the extraction of a representative transaction for each cluster. Its aim is to reach efficiency without sacrificing the clustering quality and, most important, without parameter tuning. We start by introducing the overall algorithm, and then we present details of the basic functionalities such as stopping criteria, cluster representatives and splitting procedure.

4.1 *Txmeans* Algorithm

In analogy with [7, 22], we address the clustering problem through a top-down, divide-and-conquer strategy: we start from an initial set containing a single cluster, then, iteratively we try to split a cluster in two sub-clusters. *Bisecting* strategies proved to be very effective in several different contexts like when clustering mobility data [14].

Algorithm 1: *txmeans*(B)

Input : B - set of baskets
Output: $\mathbb{C}$ - set of clusters, R - set of representative baskets

1 $r \leftarrow getRepr(B)$; // extract representatives
2 $Q.push(\langle B, r\rangle)$; // initialize queue
3 $R \leftarrow \emptyset$; $\mathbb{C} \leftarrow \emptyset$; // initialize result
4 **while** $Q \neq \emptyset$ **do**
5 $\quad \langle C, r\rangle \leftarrow Q.pop()$; // extract from the queue
6 $\quad I \leftarrow \bigcap_{b \in C}$; // calculate common items
7 $\quad C^* \leftarrow \{c \setminus I | c \in C\}$; // remove common items from baskets
8 $\quad r^* \leftarrow r \setminus I$; // remove common items from representative
9 $\quad C', C'', r', r'' \leftarrow bisectBaskets(C^*)$; // split cluster
10 $\quad bic_o \leftarrow bic(\{C^*\}, \{r^*\}, |C^*_N|, |C^*_D|)$; // BIC original
11 $\quad bic_s \leftarrow bic(\{C', C''\}, \{r', r''\}, |C^*_N|, |C^*_D|)$; // BIC splt
12 $\quad$ **if** $bic_s > bic_o$ **then**
13 $\quad\quad C' \leftarrow \{c \cup I \mid c \in C'\}$; $r' \leftarrow r' \cup I$; // restore items
14 $\quad\quad C'' \leftarrow \{c \cup I \mid c \in C''\}$; $r'' \leftarrow r'' \cup I$; // restore items
15 $\quad\quad Q.push(\langle C', r'\rangle)$; $Q.push(\langle C'', r''\rangle)$; // update queue
16 $\quad$ **else**
17 $\quad\quad \mathbb{C} \leftarrow \mathbb{C} \cup \{C\}$; $R \leftarrow R \cup \{r\}$; // update result
18 $\quad$ **end**
19 **end**
20 **return** $\mathbb{C}, R$;

The general schema of *txmeans*, which implements this approach, is specified in Algorithm 1. The algorithm starts extracting a representative basket r (described in Section 4.3) for the whole dataset B, and puts both B and r into a queue Q that keeps track of the set of baskets still to be considered for splitting (lines 1–2).

At each iteration, a cluster C and its representative r are extracted from Q (line 5). In steps 6–8 the algorithm identifies the items I which are common to all the baskets, and removes them from the transactions of the cluster and from its representative. The results are denoted by $*$. The point of this task is that such items *(i)* provide no useful knowledge for the bisecting step, and *(ii)* a large number of common items tends to *flatten* the similarity values, making it more difficult to appreciate the variability in the other parts of the transactions. Preliminary tests showed that keeping common items affects the splitting step, degrading the overall performance.

Then, the partitioning of C into two disjoint sub-clusters C', C'' is calculated using *bisectBaskets* (described in Section 4.3) over the *clean* transactions (line 9). After that, lines 10–11 calculate the *BIC* (Section 4.2) on the original cluster (bic_o) and on the two sub-clusters (bic_s). Here, $|C_N|$ is the number of baskets in C while $|C_D|$ is the number of different items in C. If the split is useful (line 12) the common items are reinserted, and C', C'' and r', r'' are added to Q (lines 13-15). Otherwise, the original cluster C and its representative basket r are added to the final sets $\mathbb{C}$ and R (line 17).

4.2 *Txmeans* Stopping Criterion

Given a cluster $C \subseteq B$ and two disjoint sub-clusters $C', C'' \subseteq C$, we need a criterion to decide whether the splitting is actually useful, i.e., the split significantly improves the homogeneity of C, in which case it is performed and the procedure reiterates on each sub-cluster.

In the literature, various measures and cost functions are adopted [5, 30]. However, they are all *global* measures which consider the whole partitioning during the clustering process.

A *local* measure to drive this decision is the *Bayesian Information Criterion (BIC)* [24], which selects the model with the highest *BIC* value. *BIC* has been successfully employed in [22] to control the splitting process and to determine the number of clusters. Yet, to the best of our knowledge it was never considered for transactional clustering, since it involves a *variance* computation [16] and thus requires central values for each cluster which are unavailable in most transactional clustering methods. The representative baskets computed in our solution provide exactly this kind of information, thus enabling the use of the *BIC* criteria formalized as follows [22]:

$$bic(N, D, C, r) = \mathcal{L}(N, D, C, r) - \frac{|C|(D+1)}{2} \log |C|$$

$$\mathcal{L} = \sum_{C_i} N_i \log N_i - N_i \log N - \frac{N_i}{2} \log 2\pi - \frac{N_i D}{2} \log \sigma^2 - \frac{N_i - |C|}{2}$$

$$\sigma^2 = \frac{1}{N_i - |C|} \sum_{b_i} dist(b_i, r_{(i)})^2$$

where C and r are the set of clusters and their representatives, $\mathcal{L}$ is the log-likelihood, N=$|C_N|$ and D=$|C_D|$ are the number of baskets and items in C, $|C|$ is the number of clusters (one in bic_o, and two in bic_s, see Algorithm 1), N_i=$|C_i|$ is the number of baskets in cluster C_i, σ^2 is the variance and $r_{(i)}$ is the representative of basket b_i.

According to literature [22], the *BIC* criterion can be adopted with success when the size of the data sample is larger than the data dimensionality, i.e., $N \gg D$. This requirement is typically satisfied by the input dataset B. Moreover, *BIC* is evaluated over each single cluster C, and not the whole dataset (the only exception being the first iteration of the process). Therefore, it is expected that the actual dimensionality, now reduced to $|C_D|$ (i.e., the set of items in cluster C), is such that $|C_D| \ll D$, since clusters group similar transactions. *Txmeans* further strengthens this property by removing the common items in each cluster before splitting. This step influences the computation of *BIC* by changing the similarity values involved in most computations. In particular, *(i)* the effect of items removal is an increase in the values of BIC, since the dimensionality of data decreases, meaning (from a more theoretical perspective) a smaller number of free parameters and therefore a better model; and *(ii)* in practice, we verified experimentally that at each candidate split the effects of item removal on the original cluster (bic_o) and on the new pair of clusters (bic_s) are usually similar enough to keep the split decision unaffected.

4.3 *Txmeans* Bisecting Schema

Center-Based Optimization. At each iteration, the cluster splitting process invoked by *txmeans* tries to divide a cluster C into two compact subgroups $\{C', C''\}$. The criteria we adopt is based on a distance function between the cluster elements and a *representative basket*. More formally, we want to find a partitioning such that:

- C',C'' have corresponding representative baskets r',r'';
- the partitioning minimizes the Sum of Squared Errors

$$SSE = \sum_{b \in C'} dist(b, r')^2 + \sum_{b \in C''} dist(b, r'')^2$$

where $dist(a, b)$ is a distance function based on the measure of overlap of items between sets a and b.

Algorithm 2: $getRepr(B)$

Input : B - set of baskets
Output: r - set of representative baskets

1 $I \leftarrow \bigcup_{b \in B} b \setminus \bigcap_{b \in B} b$; // calculate not common items
2 $\forall i \in I. freq(i) \leftarrow |\{b \in B | i \in b\}|$; // calculate frequencies
3 $i \leftarrow 0$; $r_{(i)} \leftarrow \bigcap_{b \in B} b$; $d_{(i)} \leftarrow \infty$; // initialize variables
4 **while** $I \neq \emptyset$ **do**
5 $m \leftarrow \text{argsmax}_{i \in I} freq(i)$; // set of max-freq items
6 $r_{(i+1)} \leftarrow r_{(i)} \cup m$; // update representative
7 $d_{(i+1)} \leftarrow \sum_{b \in B} dist(b, r_{(i+1)})^2$; // compute SSE
8 **if** $d_{(i)} \leq d_{(i+1)}$ **then** $I \leftarrow \emptyset$; // best representative found
9 **else** $i \leftarrow i + 1$; $I \leftarrow I \setminus m$; // update variables
10 **end**
11 **return** $r_{(i)}$;

A consequence of our notion of optimality is that each basket belongs to the cluster minimizing the distance with its "centroid", i.e., maximizing the overlap among the items.

Distance Function. While in theory *txmeans* could adopt any distance function $dist(\cdot, \cdot)$ for comparing transactions, in practice, being designed for transactional data, the number of reasonable choices is reduced. According to the literature [30], distances like Euclidean or Manhattan are not suitable in this context. Indeed, the function $dist(a, b)$ should be based on the measure of overlap of items between baskets a and b, which suggests measures such as set intersection, match similarity or Jaccard coefficient. Since the latter is known to be more robust and adequate for sparse vector data (like transactions), *txmeans* adopts Jaccard distance as default. Finally, we want to stress that a strategy like the one proposed by the *k-modes* algorithm for categorical data does not work for sparse transactional data. Indeed, if we binarize a sparse transactional dataset in the corresponding categorical dataset, the zeros usually predominate, and therefore the corresponding modes will be zero for most of the columns, i.e., the centroids will be empty at every iteration. Also, alternative approaches where lower thresholds are adopted (i.e., lower than the 50% involved by the mode) might avoid empty centroids but would introduce a new parameter to set, since each dataset might require a different threshold value.

Representative Baskets. Using Algorithm 2, *txmeans* extracts the *representative baskets* following a parameter-free heuristics defined in [12] that first selects the items contained in every transaction in B (lines 1–3), and then refines such approximation by adding the most frequent items (lines 5–6). This process is iterated as long as each step improves the solution (lines 7–8), thus stopping when a locally optimal representative r is generated. A representative basket is a virtual transaction that (approximately) minimizes the overall distance from all the baskets in the cluster, therefore capturing the items that best characterize it, i.e., the typical combination of items expected to appear in any of its transactions.

Bisecting Schema. *Txmeans* exploits the representative baskets for partitioning a set of baskets B into two disjoint sets C', C'' using the *bisecting* procedure [27] reported in Algorithm 3. First of all, two representatives are selected among the baskets in B (line 2). These initial centroids are selected with *selectInitialCentroids*, which randomly picks several pairs of baskets, and returns the

Algorithm 3: $bisectBaskets(B)$

Input : B - set of baskets
Output: C', C'' - baskets partitioning; r', r'' - repr

1 $i \leftarrow 0; SSE_{(i)} \leftarrow \infty;$ // init. variables
2 $r'_{(i)}, r''_{(i)} \leftarrow selectInitialCentroids(B);$ // init. variables
3 **while** True **do**
4 $\quad \{C', C''\} \leftarrow assignBaskets(B, \{r'_{(i)}, r''_{(i)}\});$ // assign baskets
5 $\quad r'_{(i+1)} \leftarrow getRepr(C');$ // calculate representative
6 $\quad r''_{(i+1)} \leftarrow getRepr(C'');$ // calculate representative
7 $\quad SSE_{(i+1)} \leftarrow \sum_{b \in C'} dist(b, r'_{(i+1)})^2 + \sum_{b \in C''} dist(b, r''_{(i+1)})^2;$ // calc. SSE
8 $\quad$ **if** $SSE_{(i+1)} \geq SSE_{(i)}$ **then**
9 $\quad\quad$ **return** $C', C'', r'_{(i)}, r''_{(i)};$
10 $\quad$ **end**
11 $\quad i \leftarrow i + 1;$ // update variable
12 **end**

pair with the highest distance value. Then *assignBaskets* (line 4) compares each basket $b \in B$ to the two representative baskets r' and r'' and associates it to the closest one. When all baskets have been assigned to a cluster, r' and r'' are re-computed through *getRepr* (lines 5–6). The process is reiterated as long as the SSE decreases (lines 7–9), i.e., the SSE at step i+1 is higher than that of step i.

4.4 Termination and Complexity

We provide a few theoretical properties about *txmeans*, including proofs of termination and computational complexity.

Theorem 4.1 (Termination). *The* txmeans *algorithm terminates for any input dataset.*

Proof. Both Alg. 2 and Alg. 3 terminate for any input data. The stop condition of the loop in Alg. 2 is that I becomes empty, since it decreases strictly monotonically at each iteration (steps 9 or 11–12). Also, Alg. 3 follows the classical *kmeans* structure, and the loop stops when the SSE does not strictly increase. Since the number of possible clusterings, and thus of possible SSE values, is finite, the strictly monotonic sequence of SSE values produced throughout the iterations must eventually reach a (local) minimum in a finite number of steps. Finally, the loop in Alg. 1 iteratively removes a cluster and replaces it with strictly smaller ones. In the worst case all clusters will be broken down to singletons in a finite number of steps. That avoids any possible unbounded loop, leading to termination. □

Theorem 4.2 (Complexity). *The computational complexity of* txmeans *is $O(It \cdot N \cdot K \cdot D)$, where It is the number of iterations required to reach convergence in a run of* bisectBaskets*, N=$|B|$ is the number of transactions in input, K is the number of clusters detected and D is the number of distinct items in the dataset.*

Proof. As mentioned in the proof of Theorem 4.1, the *txmeans* algorithm iteratively splits the initial dataset B into sub-clusters. The number of iterations corresponds to the number K of clusters returned at the end of the computation. All the operations performed at each iteration involve scanning the transactions in the cluster only once, thus the overall cost is $O(N \cdot D)$, notice that the size of the clusters is always $O(N)$ in case all iterations produce extremely unbalanced splits. The only exception is Alg. 3: it follows a *kmeans* structure with k=2, and all the operations performed at each step are linear in the number of transactions and their length. That includes also the calls of Alg. 2, since they involve $O(D)$ iterations each taking only constant-time operations plus the distance computation in step 7. The latter, though apparently requiring $O(N \cdot D)$ time, actually can be computed incrementally along the different iterations (both numerator and denominator of the Jaccard distance monotonically increase at each iteration), thus taking only $O(N)$, and lifting the overall cost of Alg. 2 to $O(N \cdot D)$. There is no clear bound on the number of iterations required by Alg. 3 to converge, which is then kept as a parameter It and leads to a cost equal to $O(It \cdot N \cdot D)$, which dominates the complexity of each iteration of *txmeans*. The overall complexity, thus, results to be $O(It \cdot N \cdot K \cdot D)$. □

The theoretical complexity of *txmeans* is similar or smaller than most competitors in the literature. Where not explicit, we inferred the complexities from the corresponding papers. *Tkmeans* [12], *clope* [36] and *practical* [5] follow a *k-means* structure, i.e., $O(It \cdot N \cdot K \cdot D)$ that is the same as *txmeans*. *Coolcat* [3] has a similar cost, plus a $O(S^2)$ due to the initialization over a sample of size S, that dominates the complexity if $S > \sqrt{N}$. *Atdc* [7] iteratively performs a partitioning having cost $O(It \cdot N)$ followed by a stabilization step $O(It' \cdot N \cdot K)$. The two steps are repeated till convergence, leading to an overall cost of $O(It \cdot It' \cdot It'' \cdot N^2 \cdot K^2)$, where It, It' and It'' are the number of iterations for each component of the algorithm.

4.5 Dealing with Very Large Datasets

We remark that *txmeans* has been designed to solve the mass transactional clustering problem, thus considering that many different datasets must be efficiently clustered without parameter tuning. While this situation usually results in analyzing small- or medium-size datasets, nowadays we might need to move to a "big data" context where each single dataset actually contains a huge set of transactions, therefore calling for a scalable approach.

In this case, *txmeans* can be adapted to integrate a sampling strategy. Thanks to the representative baskets, the clustering can be computed on a subset, then assigning the remaining transactions according to their closest representative basket [3]. The algorithm exploiting the sampling is working as follows. First, it randomly selects a subset $S \subset B$ of S_N transactions from the input dataset B. To do that, it employs the approach proposed in [18] to estimate S_N, i.e., $S_N = ss/(1 + (ss-1)/N)$, where ss=$Z^2 p(p-1)$. Z and p are fixed and set, respectively, to the z-score of confidence level 99% and to 0.5 (but potentially modifiable for special cases). Then the transactions in S are clustered using Algorithm 1. Finally, the rest of the transactions are associated to the proper cluster using the representatives R through a *k-nearest-neighbour* strategy with k=1.

Empirical results show that, besides efficiency, sampling improves also clustering quality. This is mainly due to noisy and rare items, whose impact appears to be significantly reduced by sampling, since most of them will disappear and thus will not distort the clusters structure. When clear from the context we will refer to *txmeans* using the sampling strategy simply as *txmeans*.

5 EXPERIMENTS

In this section we evaluate the performance of *txmeans* in the context of mass transactional clustering where only parameter-free algorithms or methods using automated techniques for parameters estimation can be employed. Then, we evaluate the scalability of

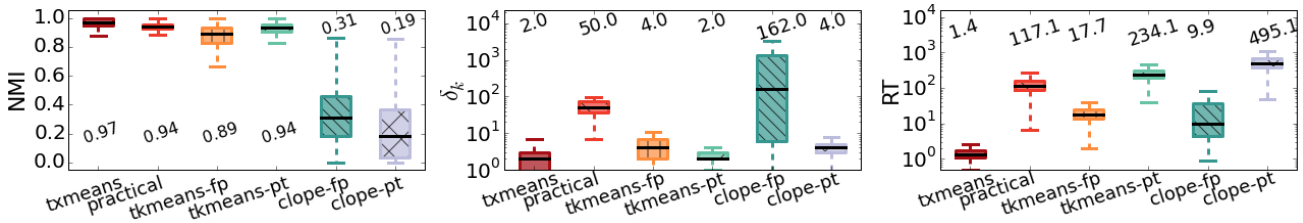

Figure 1: *NMI*, δ_k and *RT* evaluation for comparing algorithms on 10k different dataset with structure *DS*1.

txmeans. We also assess the quality of the clusterings, by studying the *txmeans* performance in various synthetic and real datasets.

5.1 Performance Indicators

To evaluate the clustering quality we compared the clusters returned with the real ones. We quantified the similarity between the two sets of clusters with the *Normalized Mutual Information (NMI)* [29]. *NMI* was preferred over *purity* because *(i)* it is more sensitive to the change in the clustering results, and *(ii)* it takes into account unbalanced distributions and does not necessarily improve when the number of clusters increases (as purity does). Given two sets of clusters $\mathbb{C} = \{c_1 \dots c_k\}$ and $\mathcal{G} = \{g_1 \dots g_{k'}\}$

$$\text{NMI}(\mathbb{C}, \mathcal{G}) = \frac{I(\mathbb{C}, \mathcal{G})}{0.5 * H(\mathbb{C}) + 0.5 * H(\mathcal{G})} \in [0, 1]$$

where $I(\mathbb{C}, \mathcal{G}) = \sum_k \sum_j (|c_k \cap g_j|/N) \log(N|c_k \cap g_j|/|c_k||g_j|)$ is the mutual information [29], and $H(\mathbb{C}) = -\sum_k (|c_k|/N) \log(|c_k|/N)$ is entropy [25]. Good clusterings have a *NMI* ~1, bad clusterings ~0.

Besides *NMI*, we evaluated the *deviation* $\delta_k = |\mathbb{C}| - |\mathcal{G}|$ between the real number of clusters and the number of clusters detected: $\delta_k \sim 0$ when the right number of cluster is detected, $\delta_k > 0$ if more clusters than real ones are detected, $\delta_k < 0$ otherwise.

Finally, we analyzed the running time *RT* (in sec) of the methods. Experiments were run on a Mac OS v10.11.4, 2,6 GHz i5, 8GB DDR3.

5.2 Competitors

We evaluate our method against a set of competitors sharing some features with *txmeans* yet following different algorithmic structures.

Practical [5] and *atdc* [7] are both parameter-free. *Practical* first scans the data and assigns each basket to an existing cluster or to a new one according to a cost function inspired by "tf-idf"; then, it moves the baskets from a cluster to another one. Note how the structure of *practical* and *txmeans* are completely different. *Atdc* adopts a divisive approach similar to *txmeans*, but *atdc* scans the baskets and iterates between a partitioning and a stabilization phase. *Tkmeans* [12] adapts the *k-means* [27] definition of distance to measure transactions dissimilarity, and computes centroids using the same approach of *txmeans*. *Coolcat* [3] works on a random sample of the baskets, as *txmeans*. We also report the performance of *clope* [36] as it represents a reference approach and it was designed for market-basket data, which is analyzed in our case study. *Clope* requires a *repulsion* parameter r that is hard to be interpreted.

We omit the comparison with *rock* [13], *subcad* [11], *limbo* [2], *clicks* [38] and *largeitem* [30], since, according to the literature, are outperformed respectively by *practical*, *atdc* and *clope*. Moreover, we omit the comparison with *k-modes* [6] and *dhcc* [33] because they are designed for categorical data, and with *purtreeclust* [9] because its target is the clustering of users through their transactions.

5.3 Mass Transactional Clustering Evaluation

In this section we evaluate *txmeans* and its competitors with respect to the task of clustering a massive number of different datasets[1]. As already discussed, this is a common situation when the datasets of a large number of users must be clustered in order to accomplish further tasks (as it is discussed and shown in Section 6). As consequence, efficiency in computing every single clustering and freedom from parameters are two mandatory requirements to meet.

Since real datasets containing users' transactions annotated with cluster labels are publicly not available, we used the synthetic data generator employed in [5], that we name *DS*1, to create individual synthetic datasets. *DS*1 was kindly provided by the authors of [7], where the generator is accurately described. *DS*1 allows to specify the following parameters to modify the dataset and the clustering structure: the number of baskets N, the number of items D, the average basket length T, the number of clusters C, the percentage of outliers O (i.e., proportion of items that do not contribute to form any cluster), and the percentage P of overlap among transactions of distinct clusters. Different combinations of O and P allow to simulate various situations, thus enabling an objective experimentation.

By using *DS*1, we generated 100k datasets with characteristics selected randomly in $N \in [1000, 10000]$, $D \in [100, 1000]$, $T \in [10, 30]$, $C \in [4, 16]$, $P \in [0, 50]$, $O \in [0, 30]$. Hence, we test mass transactional clustering using a wide set of different datasets generated with random structures and simulating different users, and we evaluate the performance considering the clusterings of all the datasets.

Moreover, since we are simulating a real application scenario, we are not suggesting the correct setting to the methods requiring parameters. Indeed, for these algorithms we adopt two versions: *fixed parameter (fp)* for which we fix a parameter setting for all the datasets, and *parameter tuning (pt)* for which we simulate the search of the best parameters through an heuristic for each dataset by running each method several times and varying the parameters.

In this experiment we consider the following competitors: *practical* because is the best parameter-free algorithm according to the state of the art, *tkmeans* because uses the same approach as *txmeans* for extracting the centroid, and *clope* because has a completely different approach not requiring to specify as parameter the number of cluster. As shown in the following, *atdc* and *coolcat* have relatively poor performance, even when clustering simple and single datasets. Hence their performance are not reported in this test.

We name the parameter-based competitors with fixed parameters *tkmeans-fp* and *clope-fp*, and those with parameter tuning *tkmeans-pt* and *clope-pt*. For *tkmeans-fp* and *clope-fp* we fix k=6 and r=2 respectively. We used the heuristic technique "knee method" to select the best k and r: we run *tkmeans* for $k \in [4, 16]$ and we store the

[1] The Python code of the algorithms and the data generators is at https://goo.gl/sAJ7WO.

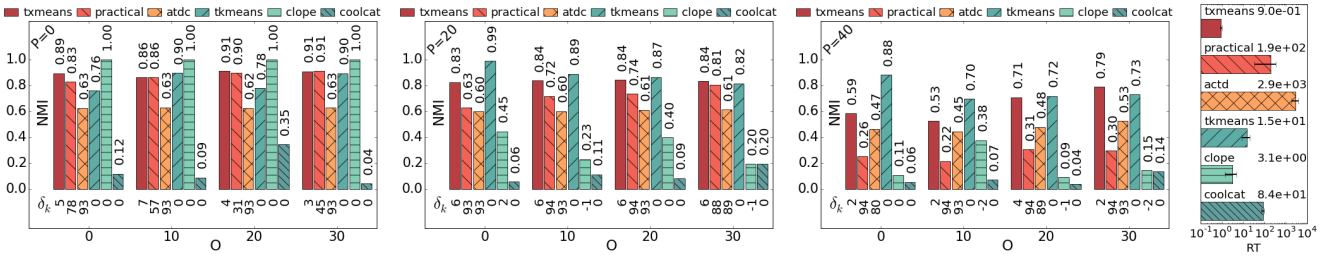

Figure 2: *NMI*, δ_k and *RT* evaluation for comparing algorithms on *DS*1 with N=2000, D=200, T=10 and C=6.

SSE for every run. Then we select as best k the one corresponding in the "knee" point in which the SSE curve changes [27]. We adopted a similar technique for *clope* considering the trend change of the *Profit* function [36] with $r \in [0.1, 3.5]$. Note that these parameters cannot be selected by optimizing an evaluation metric like *NMI* because at this stage we do not possess any information about the clustering structure of the dataset we have to analyze.

Figure 1 depicts the boxplots of NMI, δ_k, RT (from left to right). The numbers represent the median values of each boxplot. *Txmeans* shows remarkably the best performance: it returns for each dataset the purest clusters (a median level of 0.97 *NMI*) in a few seconds without any parameter tuning and it deviates on average of only 2 clusters from the real number. Considering these three aspects at the same time, this level of performance is reached by none of the competitors. *Practical* and *tkmeans-pt* have a *NMI* gap w.r.t. *txmeans* of only 0.03. However, they both have a median *RT* two orders of magnitude greater than *txmeans*. Moreover, while *tkmeans-pt* succeeds in minimizing δ_k, *practical* has an average deviation of ~50 clusters, that is unacceptable because, even if the clusters extracted are pure, they would provided a too dispersed summary of users'patterns for any real application. *Tkmeans-pt* is highly penalized in terms of *RT* by the multiple runs for tuning the number of clusters k, while *tkmeans-fp* has lower running times, but also lower *NMI* and higher δ_k. Note that, *tkmeans-fp* has overall good *NMI* and δ_k because the most frequent number of clusters for the generated datasets is exactly 6. *Clope* is not competitive in none of the versions. In conclusion, *txmeans* is the best algorithm in case of transactional clustering on a massive number of datasets.

5.4 Assessing Clustering Quality

The goal of these experiments is to evaluate the ability of *txmeans* and its competitors to correctly identify clusters under specific circumstances in synthetic datasets and real-world datasets.

Synthetic Datasets. For synthetic datasets, we study the performance when specific datasets among those generated on the previous experiment must be clustered. In particular, we employ *DS*1 to generate different datasets with controlled overlap percentage $P \in \{0, 10, 20, 30, 40, 50\}$ and outliers percentage $O \in \{0, 10, 20, 30\}$ and we fix the other dimensions to N=2000, D=200, T=10, C=6.

In this experiment we offer a significant advantage to parameter-dependent algorithms, by setting their parameters to optimal values: the real number of clusters k for *tkmeans* and *coolcat*, and for *clope* the value of r that minimizes δ_k, with $r \in [1.0, 3.5]$. Finally, the sample S for *coolcat* is selected through the same function adopted by *txmeans*. Figure 2 shows the performance by barplots of *NMI*, with δ_k reported below the bars. When varying the percentage of outliers O for a given level of overlap P, the *NMI* of all the algorithms has small fluctuations. The overall level of *NMI* decreases significantly when P grows. *Txmeans* and *tkmeans* are the two most stable ones, but the latter is given the right number of clusters as input. However, *txmeans* maintains good performance for $P \geq 30$ and O=30 and overcomes also *tkmeans*. *Clope* is the best performer when there is no overlap and no noise, then its performances rapidly decrease for growing P. *Coolcat* is always the worst performer.

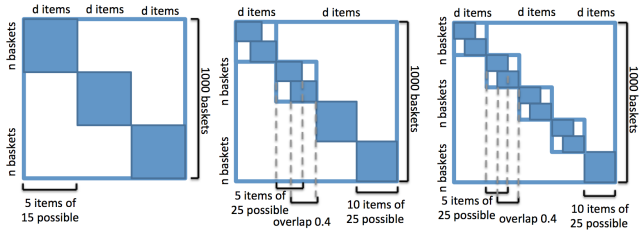

Figure 3: Structure of synthetic data for *DS*2, *DS*3, *DS*4.

Algorithm	DS2 (N=1000,D=75,C=3)			DS3 (N=1000,D=100,C=6)			DS4 (N=1000 D=125,C=9)		
	NMI	δ_k	RT	NMI	δ_k	RT	NMI	δ_k	RT
txmeans	0.66	**11.5**	***0.42***	***0.87***	**6.8**	***0.43***	0.83	**10.4**	***0.62***
practical	0.57	51.8	20.96	0.73	***20.7***	6.06	0.68	***36.6***	12.22
atdc	0.53	***49.1***	130.86	0.65	48.4	189.52	0.72	49.5	261.92
tkmeans	***0.67***	-	2.12	0.75	-	3.41	***0.86***	-	4.79
coolcat	0.01	-	14.4	0.22	-	24.97	0.34	-	31.94
clope	**1.00**	-	**0.16**	**0.99**	-	**0.10**	**0.99**	-	**0.10**

Table 1: Clustering performance on *DS2*, *DS3*, *DS4* datasets. Best performer in bold, second best performer in bold-italic.

Moreover, it is worth to notice that even though the difference of *NMI* between *txmeans* and *practical* can be considered small, *practical* has a significant deviation δ_k, already observed also in Section 5.3, which represents a clear weakness of this approach. Both *practical* and *atdc* largely overestimate the number of clusters, while the overestimation of *txmeans* is very small.

Finally, the average *RT* values are reported on the right of Figure 2. As previously observed, *txmeans* is the most efficient: its *RT* is one order of magnitude smaller than *clope* and two orders of magnitude smaller than *practical*. Furthermore, like *coolcat*, due to the sampling, *txmeans* is also the most constant in *RT*.

We performed additional experiments on other simpler synthetic datasets (*DS2*, *DS3*, *DS4*) generated according to the state of the art [5, 7, 34]. The clustering structure of these dataset is reported in Figure 3. These datasets have a well defined clustering structure, and each basket distinctly belongs to one cluster. *DS*2, has a one-layer clustering with C=3 clusters of the same size and each basket

Algorithm	Mushrooms (N=8124,D=22,C=2)			Zoo (N=101,D=17,C=7)			Congress (N=435 D=16,C=2)		
	NMI	δ_k	RT	NMI	δ_k	RT	NMI	δ_k	RT
txmeans	***0.406***	***3.5***	**1.082**	**0.826**	**-2.2**	0.066	0.358	7.0	0.304
practical	0.001	**-0.5**	***5.775***	0.426	-5.3	***0.029***	***0.470***	**0.3**	***0.124***
atdc	0.216	4.0	33.421	0.736	***-3.0***	0.091	0.297	***1.0***	0.319
tkmeans	0.158	*	302.645	0.773	*	2.161	**0.475**	*	5.448
coolcat	0.005	*	73.184	0.544	*	0.890	0.413	*	13.015
clope	**0.419**	*	7.754	0.796	*	**0.005**	0.382	*	**0.042**

Table 2: Clustering performance on real-world data sets. Best performer in bold, second best performer in bold-italic.

has length T=5 and is characterized by D/C=15 different items. *DS*3 and *DS*4 are built similarly to *DS*1, but they have a two-layer clustering structure: in *DS*3 the top layer has four clusters, two of which have sub-clusters; in *DS*4 the top layer has five clusters, four of which have sub-clusters. In both cases the items overlap in sub-clusters is 0.4 and the average basket length is $T \in \{5, 10\}$.

For each clustering structure *DS*2, *DS*3, *DS*4 we generate ten datasets. Tab. 1 reports the mean value of the evaluation measures. The deviation δ_k is considered only for parameter-free methods. *Clope* is the best performer but we must consider that the best r is provided as input, and that in practice, as previously shown, this requires an extensive tuning. *Txmeans* is the second best performer w.r.t. *NMI* for *DS*3 and the third for *DS*2 and *DS*4 without requiring any tuning. Also *tkmeans* has very good performance. All the parameter-free algorithms overestimate the number of clusters, but the overestimation of *txmeans* is much smaller than that of *practical* and *atdc*. Finally, *txmeans* also has the smallest *RT*. Also these experiments confirm the effectiveness of *txmeans* in extracting in the shortest time the clusters with the highest quality.

Real-World Datasets. Moving to real-world, we analyzed three datasets from the UCI repository[2]: *Mushrooms*, *Congressional Votes* and *Zoo*. Classes information is used as ground truth for validation.

As showed in Table 2, *txmeans* performs well on all the datasets. Like before, the deviation δ_k is considered only for parameter-free methods. For *Mushrooms* dataset *txmeans* is the second best performer with respect to *NMI* and δ_k and the best performer with respect to *RT*. Good performances are obtained only by *txmeans* and *clope*: *practical* underestimates the number of clusters, while *atdc* overestimates it. The parameter-free algorithms underestimate the real number of species in *Zoo*. Despite this fact, *txmeans* is the best performer and returns a partitioning even better than parameter-based algorithms for which the number of clusters was correctly specified. In *Congressional* dataset *txmeans* is not the best one but the results are comparable to those obtained by the competitors. Overall, the experiments on real datasets suggest that *txmeans* provides consistent and stable results with respect to the competitors confirming the suitability observed on synthetic data.

5.5 Evaluating Scalability

We evaluate the scalability of *txmeans* by observing its performance when varying N, D, C on *DS*1. We fixed $T = 20$, $P = 20\%$, $O = 20\%$.

The *first column* of Figure 4 shows the scalability varying C. For *NMI* we observe that when C is small there are better performances

[2]http://archive.ics.uci.edu/ml/

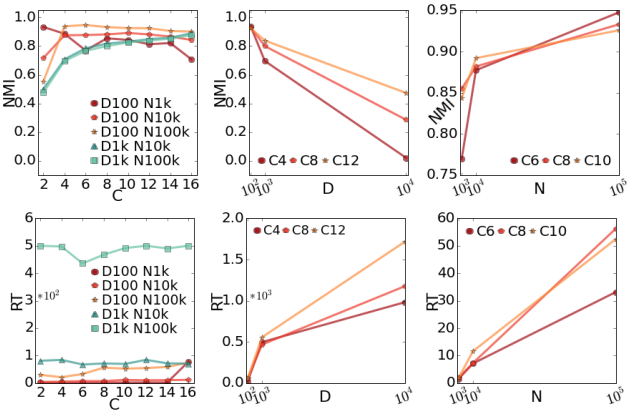

Figure 4: Dataset *DS*1, scalability w.r.t: *(first column)* clusters C, *(second column)* items D, *(third column)* baskets N.

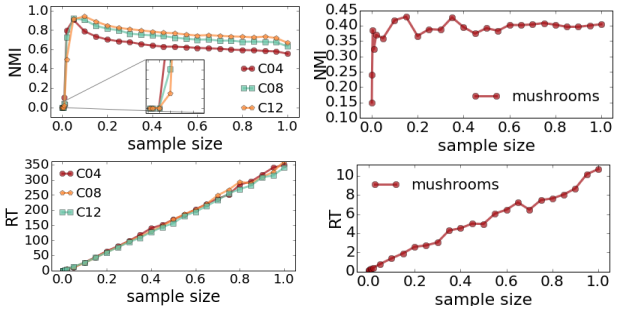

Figure 5: *NMI* and *RT* varying S on *DS*1 ($N = 10,000$, $D = 1,000$, T=20, P=20 and O=10) (left), and on *Mushrooms* (right).

with small datasets, while with high C there are better performances for large datasets. The *RT* does not fluctuate when varying C.

The *second column* reports the variation of D with fixed N=100, 000. The *NMI* is influenced by the dataset density, indeed fixing N we observe that increasing D the *NMI* values decrease drastically. However, the *NMI* decreases more slowly for higher number of clusters. The *RT* grows less than linearly in D and it is higher for high C.

Finally, in the *third column* of Figure 4 we observe the scalability varying N (with D=100). The *NMI* grows with N: the more the baskets, the less the noise, and the better are the representatives. The *RT* grows linearly and it is not influenced by C.

5.6 Evaluating Sample Size

In this last experimental section we evaluate the performance of *txmeans* varying the size of the initial sample S both in case of synthetic and real world datasets. Figure 5 *(left)* illustrates the variations of *NMI* and *RT* for *txmeans* on *DS*1 when changing the *sample size* S for a dataset with T=20, D=1, 000, N=10, 000, P=20, O=10, while on the *(right)* we can observe the same indicators for the *Mushrooms* dataset. A clear trend appears for both datasets: a first peak of high values of *NMI* is positioned in a range of sample size between 0.05 and 0.15, then the trend decreases a little before stabilizing. The range [0.05, 0.15] confirms that the function using the z-score [18] that we adopted to select the size S_N of the initial sample S is a good choice. Indeed, it returns samples S with a size which is generally between the 5% and the 20% of the whole dataset. Finally, the *RT* grows linearly with the sample size.

6 REAL CASE STUDY APPLICATION

The efficiency and the freedom from parameters of *txmeans* makes it possible to adopt clustering-based strategies in applications that need to handle mass transactional clustering. We present an application where clustering individual transactional data for masses of users is necessary to build personalized recommendation systems that requires to analyze thousands of shopping sessions datasets. In this context, transactional data are typically treated with simple strategies, while more complex approaches are avoided exactly for the same reasons that motivated the development of *txmeans*. Enabled by the capabilities of *txmeans*, the proposed solution is a first attempt to go in the opposite direction.

The application consists in a *Personal Cart Assistant* suggesting to retail customers potential items to add to their current basket based on their shopping behavior. The suggestions are based on the representative baskets and clusters obtained from their purchasing history. In the following we describe our solution, we present baselines approaches, and we show comparative empirical results.

Personal Cart Assistant. Given the baskets B_u of a customer u, the *personal cart assistant* (*pca*) method cluster B_u using *txmeans* to obtain $\mathbb{C}_u = \{C_1, \ldots, C_k\}$ and $R_u = \{r_1, \ldots, r_k\}$. $P_u = \langle \mathbb{C}_u, R_u \rangle$ is the customer's shopping profile [1] used as basis for a model-based collaborative filtering approach [26, 28]. Then, given the current incomplete basket $L = \{i_1, \ldots, i_n\}$ of user u, *pca* finds the $r_i \in R_u$ which is closer to L in terms of Jaccard distance, and then uses the baskets in C_i to generate suggestions. A weight is associated to each candidate item, computed as the sum of similarities between current basket L and the baskets in C_i that contain the candidate items. Then, only the highest-weight items are suggested. This process is an instance of the general *collaborative filtering* approach, with a set of users (here corresponding to single baskets), user's preferences (the items in L), users selected as similar w.r.t. the user's preferences (the baskets in C_i) based on item bought in the past (r_i). A difference from classical collaborative filtering is that our user's preferences (i.e., the shopping list L) are binary instead of scores [21].

Baselines. We compared the performances of our method against the following baselines. *Last* suggests the items in the last basket purchased. *Rand* suggests random items among those in B_{u_i}. *Most* recommends the most frequent items in B_{u_i}. *Mbcf* is a memory-based collaborative filtering method on transactional data [21]. Although in the literature there are other methods personalized on user behavior, e.g. [23, 31], none of them try to complete the current shopping list, and thus cannot be directly compared.

Real Dataset. We tested our method over real data describing the purchases of the customers of a large Italian supermarket chain. Customers are provided with a loyalty card which allows to link different shopping sessions, and therefore reconstruct their personal shopping history. We analyzed a dataset of $2,670,343$ shopping sessions occurred in Leghorn province over 2010–2013, corresponding to about $10k$ loyal customers, i.e., customers active in at least ten months every year. For each customer we have $N{\sim}240$ baskets, $D{\sim}100$ different items, and an average basket length T of $\sim$8 items.

Representative Baskets Analytics. The number of clusters per customer ranges from 2 to 22 with a mean of $\sim$5 and a skewed left normal distribution: every customer has her own purchasing behavior made of a different number of patterns. In Table 3 we report the representative baskets of three customers to highlight the differences in terms of number of clusters and baskets composition. Customer *A* has two very different representatives: the first one contains perishable items while the second one is also related to bathroom products. Customer *B* always purchases *salad bags, wine* but, the first two representatives are characterized by *apples, tomatoes*, while the second two by *oranges, onions*. The purchases of customer *C* are characterized by two different patterns: we notice three representatives dominated by *bovine steak, bread, milk*, versus two representatives containing cake ingredients and female personal products. Finally, in general, there is a sort of "matrioska effect" where most of the representatives contain a small kernel of frequent items and larger representatives contain additional frequent item. This delineates two shopping behaviors: *big-purchases* typically performed on a monthly base, containing disparate types of items and not only perishable foods, and *small-purchases* performed more frequently, containing both the favorite and most consumed items together with perishable products.

	Representative Baskets (Support)
A	{bread, breakfast snacks, pasta, yogurt, tomato sauce} (0.55), {basil, body cream, bovine steak, fresh pasta, onions, shampoo, tomatoes, toothpaste} (0.45)
B	{salad bag, wine, apples, tomatoes} (0.17), {salad bag, wine, apples, tomatoes, chocolate, bananas} (0.29), {salad bag, wine, oranges, onions, potatoes} (0.44), {salad bag, wine, oranges, onions, milk, kiwi} (0.12)
C	{bovine steak, bread, milk} (0.11), {bovine steak, bread, milk, fresh cheese, tomato sauce} (0.18), {bovine steak, brad, milk, fresh cheese, tomato sauce, pasta, tomatoes, canned tuna} (0.17), {bread, milk, sugar, absorbent, fresh cheese, eggs} (0.18), {bread, milk, sugar, absorbent, chocolate, flour, fresh cheese, prepared for cakes}(0.36)

Table 3: Examples of representative baskets.

Recommendation Evaluation. The first three years of data is used to extract the representative transactions and the cluster for each customer, while the last year is used for testing. We observe that for each customer $\sim$4 clusters are detected, thus few patterns are needed to represent the shopping behavior of a customer. For each customer we test the methods over each session L, in chronological order, updating the models as follows: *pca* assigns L to a cluster with respect to the representative baskets, *mbcf* considers also L into the model, *most* updates the frequencies of each item in L, *last* becomes L, *rand* also considers the items in L in its choices. Only baskets having length at least $\omega \in [2, 16]$ were considered, and the current basket L in the test set is split in two parts w.r.t. a percentage $\theta \in [0.2, 0.8]$. As quality measure we report the $F_{0.5}$ [20], which puts more emphasis on precision than recall as we focus on providing useful suggestions. The evaluations are aggregated by averaging the scores of all the customers. Figure 6 shows $F_{0.5}$ for the recommenders varying ω and θ. *Pca* performs better than any other approach with a minimum improvement of at least 0.02 (obtained over *mbcf*, the second best method), corresponding to 1 to 3 more items correctly suggested. Figure 6 *(left)*, with varying ω and θ=0.5, shows that the larger is the minimum basket length, the worse are performances. The fact that for short baskets the recommendation is easier reveals that the probability that frequent

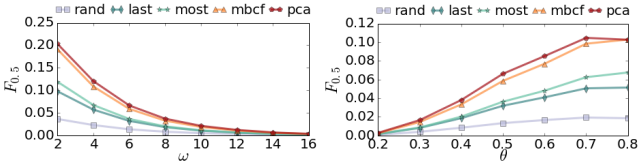

Figure 6: Recommendation performances as $F_{0.5}$ on real dataset varying minimum length ω *(left)*, split θ *(right)*.

items are regularly purchased is high. Figure 6 *(right)* shows that, for $\omega \geq 6$, the larger is the portion θ of L considered, the better are the performances, as expected, and *pca* is the best method.

7 CONCLUSIONS

In this paper we have introduced the mass transactional clustering problem and *txmeans*, a parameter-free transactional clustering algorithm able to partition efficiently a massive number of different datasets and to provide representative transactions for all clusters. Our proposal applies a bisecting strategy to find groups of similar transactions, and gives the possibility to work on a sample of the initial data, making it applicable also in the context of big data. Thanks to these features, *txmeans* is able to outperform existing algorithms both on synthetic and real-world datasets. Finally, we have shown an application scenario on a real retail sale dataset, where we built on top of *txmeans* results a *personal cart assistant*, able to suggest to the customer new items to put in her shopping list.

The main future developments of the work include the improvement of personal recommender systems through the integrated analysis of both individual patterns (already employed in this paper) and collective patterns, obtained by applying *txmeans* on the representative baskets of all customers. Also, we plan to apply *txmeans* to several other contexts. For example, in the analysis of semantically enriched mobility data, *txmeans* might help in the identification of personal point of interests typically visited by people, while in the analysis of individual health indicators it might extract the typical patterns characterizing the health conditions of a person during the daily activities. It would be interesting also applying our algorithm on social media data to infer individual patterns of interests and preferences in terms of music, food, movies, etc. In all these applications the temporal dimension of each transaction might play an important role in the analysis. Unfortunately, the current version of our algorithm does not capture this particular data dimension. Thus, a future research direction would be to extend *txmeans* for considering also the temporal dimension.

ACKNOWLEDGMENTS

This work is partially supported by the European Community H2020 Programnder the funding scheme "INFRAIA-1-2014-2015: Research Infrastructures" grant agreement 654024 (http://www.sobigdata.eu) *"SoBigData"*. We thank UniCoop Tirreno for allowing us to analyze the data and to publish the results.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2001. Using data mining methods to build customer profiles. *Computer* 2 (2001), 74–82.
[2] Periklis Andritsos, Panayiotis Tsaparas, and others. 2004. LIMBO: Scalable clustering of categorical data. In *EDBT*. Springer, 123–146.
[3] Daniel Barbará, Yi Li, and Julia Couto. 2002. COOLCAT: an entropy-based algorithm for categorical clustering. In *CIKM*. ACM, 582–589.
[4] Charles-Edmond Bichot. 2010. Co-clustering Documents and Words by Minimizing the Normalized Cut Objective Function. *MMA* 9, 2 (2010), 131–147.
[5] Mohamed Bouguessa. 2011. A practical approach for clustering transaction data. In *MLDM*. Springer, 265–279.
[6] Fuyuan Cao, Jiye Liang, and Liang Bai. 2009. A new initialization method for categorical data clustering. *ESA* 36, 7 (2009), 10223–10228.
[7] Eugenio Cesario, Giuseppe Manco, and Riccardo Ortale. 2007. Top-down parameter-free clustering of high-dimensional categorical data. *TKDE* 19, 12 (2007), 1607–1624.
[8] Keke Chen and Ling Liu. 2005. The" Best K" for entropy-based categorical data clustering. (2005).
[9] Xiaojun Chen and others. 2016. PurTreeClust: A purchase tree clustering algorithm for large-scale customer transaction data. In *ICDE*. IEEE, 661–672.
[10] Inderjit S. Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*. 269–274.
[11] Guojun Gan and Jianhong Wu. 2004. Subspace clustering for high dimensional categorical data. *SIGKDD Explorations Newsletter* 6, 2 (2004), 87–94.
[12] Fosca Giannotti, Cristian Gozzi, and Giuseppe Manco. 2002. Clustering transactional data. In *PKDD*. Springer, 175–187.
[13] Saikat Guha, Rajeev Rastogi, and Kyuseok Shim. 1999. ROCK: A robust clustering algorithm for categorical attributes. In *ICDE*. IEEE, 512–521.
[14] Riccardo Guidotti, Roberto Trasarti, and Mirco Nanni. 2015. TOSCA: TwO-Steps Clustering Algorithm for Personal Locations Detection. In *SIGSPATIAL*. ACM, 38:1–38:10.
[15] Zhexue Huang. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *DAMI* 2, 3 (1998), 283–304.
[16] Robert E Kass and Larry Wasserman. 1995. A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion. *Journal of the american statistical association* 90, 431 (1995), 928–934.
[17] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. 2004. Towards parameter-free data mining. In *KDD*. 206–215.
[18] JWKJW Kotrlik and CCHCC Higgins. 2001. Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research. *ITLPJ* 19, 1 (2001), 43.
[19] Tao Li, Sheng Ma, and Mitsunori Ogihara. 2004. Entropy-based criterion in categorical clustering. In *ICML*. 68.
[20] Christopher D Manning, Prabhakar Raghavan, and others. 2008. *Introduction to information retrieval*. Vol. 1. Cambridge university press.
[21] Andreas Mild and Thomas Reutterer. 2003. An improved collaborative filtering approach for predicting cross-category purchases based on binary market basket data. *JRCS* 10, 3 (2003), 123–133.
[22] Dan Pelleg, Andrew W Moore, and others. 2000. X-means: Extending K-means with Efficient Estimation of the Number of Clusters.. In *ICML*, Vol. 1.
[23] Steffen Rendle and others. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. ACM, 811–820.
[24] Gideon Schwarz and others. 1978. Estimating the dimension of a model. *The annals of statistics* 6, 2 (1978), 461–464.
[25] Claude Elwood Shannon. 2001. A mathematical theory of communication. *SIGMOBILE* 5, 1 (2001), 3–55.
[26] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in AI* 2009 (2009), 4.
[27] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, and others. 2006. *Introduction to data mining*. Vol. 1. Pearson Addison Wesley Boston.
[28] Lyle H Ungar and Dean P Foster. 1998. Clustering methods for collaborative filtering. In *AAAI*, Vol. 1. 114–129.
[29] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2009. Information theoretic measures for clusterings comparison: is a correction for chance necessary?. In *ICML*. ACM, 1073–1080.
[30] Ke Wang, Chu Xu, and Bing Liu. 1999. Clustering transactions using large items. In *CIKM*. ACM, 483–490.
[31] Pengfei Wang, Jiafeng Guo, and others. 2014. Modeling retail transaction data for personalized shopping recommendation. In *CIKM*. 1979–1982.
[32] Yongqiao Xiao and Margaret H Dunham. 2001. Interactive clustering for transaction data. In *DaWaK*. Springer, 121–130.
[33] Tengke Xiong, Shengrui Wang, André Mayers, and Ernest Monga. 2012. DHCC: Divisive hierarchical clustering of categorical data. *Data Mining and Knowledge Discovery* 24, 1 (2012), 103–135.
[34] Hua Yan, Keke Chen, and Ling Liu. 2006. Efficiently clustering transactional data with weighted coverage density. In *CIKM*. ACM, 367–376.
[35] Yinghui Yang and others. 2005. GHIC: A hierarchical pattern-based clustering algorithm for grouping Web transactions. *TKDE* 17, 9 (2005), 1300–1304.
[36] Yiling Yang, Xudong Guan, and Jinyuan You. 2002. CLOPE: a fast and effective clustering algorithm for transactional data. In *TKDE*. ACM, 682–687.
[37] Ching-Huang Yun, Kun-Ta Chuang, and Ming-Syan Chen. 2006. Adherence clustering: an efficient method for mining market-basket clusters. *Information systems* 31, 3 (2006), 170–186.
[38] Mohammed J Zaki and Markus Peters. 2005. CLICKS: Mining subspace clusters in categorical data via K-partite maximal cliques. In *ICDE*. 355–356.