Short Paper: A Parameter-free Customer Clustering Algorithm on Transaction Data

Xiaolei Di
School of Software Engineering
Tongji University,Shanghai, China
xldi@tongji.edu.cn

Weixiong Rao
School of Software Engineering
Tongji University,Shanghai, China
wxrao@tongji.edu.cn

ABSTRACT

Transaction data is a special kind of categorical data, such as the items purchased in a shopping cart, the web pages visited in a browsing session and the movies watched in a time period. Customer clustering, i.e., partitioning customers into groups based on their transaction data, has been widely used to analyze customer behaviors and make personalized promotion and/or recommendation strategies. However, existing works either suffer from the ineffectiveness issue and cannot accurately measure customer similarities, or require non-trivial efforts to tune the hyperparameters such as the number of clusters. To overcome these issues, we propose a parameter-free clustering algorithm, namely FTCTreeClust, to segment customers according to the similarity of the transaction data bought by customers. We represent the transaction data of each customer by a novel FTCTree structure which incorporates the inherent features such as product category, frequency and purchase time. We define a distance metric to calculate the FTCTree similarity. Next, we propose a parameter-free clustering algorithm to generate the centroid tree for every cluster in order to summarize the preference of clustered customers. The preliminary experiment on a real transaction data set validates that FTCTreeClust outperforms state-of-the-arts.

1 INTRODUCTION

Transaction data is a special kind of categorical data, such as the items purchased in a shopping cart, the web pages visited in a browsing session and the movies watched in a time period [6]. For example, the items purchased by customers represent their purchase behaviors and preferences. Customer clustering, i.e., partitioning customers into groups based on their transaction data, has been widely used to analyze customer behaviors and make personalized promotion and/or recommendation strategies.

To perform customer clustering, literature works frequently measure customer similarity by the overlapping of items purchased by customers and computer the intersection, match similarity and Jaccard similarity [4, 6]. However, the overlapping-based measurement can't distinguish users' similarity well because it misses the inherent information in the items. Let us consider the transaction data purchased by two customers: *{black tea, banana}* and *{green tea, iPhone X}*. Since *black tea* and *green tea* are frequently with two different item IDs, the Jaccard similarity of the customers could be zero. Here, the problem is that the *black tea* and *green tea* in general belong to the same category *tea*, simply using the Jaccard similarity above does not work well to measure the customer similarity. The previous work [3] developed a purTree structure where each leaf represents the purchased product and internal node represents category. Though purTree leveraged the category information to better measure the customer similarity, it fully overlooked the frequency and purchase time of the purchased items. Since consumers are more interested in the items that are frequently and recently purchased, we could leverage these information to better capture the customers' purchase behaviors.

In terms of the hyperparameter K, i.e., the number of clusters, the typical approach [3] is to define a function or metric by which we choose the best one among the set of pre-defined numbers. However, this approach does not automatically decide the number K during clustering. Though the two previous works Large item [8] and Clope [9] iteratively optimize a global criterion function to either assign the transaction record into an existing cluster or create a new one. However, they both still require extra parameters such as the minimum support and repulsion, respectively. It is rather hard to tune the two parameters.

To overcome these issues, we propose a parameter-free clustering algorithm, namely FTCTreeClust, to segment customers according to the similarity of the transaction data bought by customers. We represent the transaction data of each customer by a FTCTree structure based on the inherent features such as product category, frequency and purchase time. We define a distance metric to measure the distance of FTCTrees. The parameter-free clustering algorithm can generate the centroid tree for every cluster and perform the customer clustering in a hierarchical divisive manner. As a summary, we make the following contributions.

- To represent the transaction data purchased by customers, we propose a novel FTCTree structure which incorporates product category, frequency, and purchase time. The FTCTree distance used to measure the similarity between FTCTrees can effectively overcome the curse of high-dimensionality.
- The proposed FTCTreeClust algorithm automatically estimates the number of clusters and uses a centroid tree to summary the purchase preference for each cluster.
- The preliminary experiment on a real transaction data set validates that FTCTreeClust outperforms state-of-the-arts.

The rest of the paper is organized as follows. Section 2 gives the problem setting, and Section 3 describes the design of FTCTreeClust. After that, Section 4 reports our experiments, and Section 5 finally concludes this paper.

2 PROBLEM DEFINITION

Definition 2.1. **Transaction data**: A transaction record t ={Tid, Cid, PurTime, Item, Category} contains transaction id, customer id, purchase time, purchased item, and the category list organizing the item. Let Category = $\{c_1, ..., c_k\}$ be a set of k level categories where $c_i (1 \leq i \leq k)$ represents the i-th level category.

For example, for the first record $Tid = 1$ in Table 1, the item *black tea* is with the category list *{food, drink, tea}*. A higher level indicates a finer granularity category. For the real dataset in our experiment, each product item contains a 4-level category list.

© 2020 Copyright held by the owner/author(s). Published in Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), March 30-April 2, 2020, ISBN XXX-X-XXXXX-XXX-X on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

Table 1: Three Transaction Records of a Customer

Tid	Cid	PurTime	Item	Category
1	001	20190112	black tea	[food, drink, tea]
2	001	20190110	green tea	[food, drink, tea]
3	001	20180919	T-shirt	[clothes, children clothes, shirt]

Table 2: Purchase Time and Level

Time	Level
purTime $\leq 2^0$ month	4
2^0 month < purTime $\leq 2^1$ month	3
2^1 month < purTime $\leq 2^2$ month	2
purTime > 2^4 mouth	1

Definition 2.2. **Purchase Tree** [3]: A purchase tree (purTree in short) organizes the product items bought by a customer. The root of a purTree is an empty node, each leaf node represents an item, and an internal node represents the category used to organize the purchased items (resp. coarser categories).

In our real dataset, each product item has the equal 4-level of categories. Thus, all leaf nodes in a *purTree* have equal depth. Nevertheless, *purTree* can be generally extended to a tree of unequal length without degrading the overall approach.

Problem 1. *Given a set of customers $\mathcal{U} = \{U_1, ..., U_M\}$ and the corresponding transaction sets $\mathcal{T} = \{T_1, ..., T_M\}$, we want to represent the customers $\mathcal{U}$ by the FTCTrees $\mathcal{R} = \{R_1, ..., R_M\}$ and segment the customers $\mathcal{U}$ into K disjoint sets $C = \{C_1, ..., C_k\}$, such that the customers within the same clusters have a higher similarity than those from different clusters.*

3 DESIGN OF FTCTREECLUST

In this section, we first construct the FTCTree structure. Next, we define the distance between FTCTrees to measure customer similarity. Finally, we present the FTCTreeClust algorithm.

3.1 FTCTree Construction

Given a set of transactions purchased by a customer, denoted by $T = \{t_1, ..., t_n\}$, we construct a FTCTree R to represent T. Figure 1(c) gives an example of the FTCTree structure. Unlike purTree [3], each node in FTCTree maintains the product/category information together with the purchase frequency and latest purchase time. Moreover, FTCTree allows data decaying on historical transaction records. It makes sense because both recent changes at a fine scale and long-term changes at a coarse scale items are meaningful for product recommendation and analysis of purchase behaviours. Thus, FTCTree keeps different granularities of data decaying according time distance. That is, the most recent timestamp corresponds to the finest granularity, and the more distant timestamp corresponds to a coarser granularity. Typically there exist three types of time dimension models: (1) natural tilted time frame model, (2) logarithmic tilted time frame model, and (3) progressive logarithmic tilted time frame model. We adopt the logarithmic tilted time frame model for transaction data. It is mainly because supermarkets tend to count sales by week, month and quarter rather than by hour or quarter. In addition, FTCTree is not snapshot, and the progressive logarithmic tilted time frame model is not available for FTCTree.

Table 2 illustrates how data decay performs on historical transaction records in FTCTree by the logarithmic tilted time frame model. This time frame is structured in multiple granularities

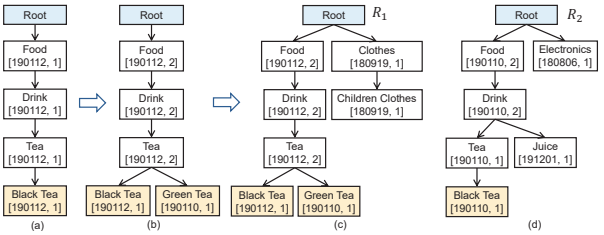

Figure 1: FTCTree Construction

(or levels) according to a logarithmic scale. In this way, FTCTree can precisely represent the customer's recent purchase behavior and meanwhile save storage space for historical transaction data. Note that besides the frequency and recent timestamp, FTCTree could incorporate more inherent features of products, if available.

After introducing the FTCTree structure, we construct a FTCTree from the transaction records T purchased by a customer. Here, we take three transaction records of a customer in Table 1 for illustration. The construction first initiates a tree with an empty root and next inserts the nodes for the first record. Following the logarithmic tilted time frame in Table 2, the purchase time of *black tea* allows the 4-level categories. According this level 4, we insert the 4-level categories into the FTCTree as the internal nodes and the product item as a leaf node. As shown in Figure 1(a), the FTCTree structure contains three internal nodes (the categories of *Food*, *Drink*, and *Tea*), and one leaf node (the item *Black Tea*). In addition, we insert the purchase time and frequency as the attributes of each node into the tree structure.

Next, the three categories of the 2nd record, i.e., *Food*, *Drink*, and *Tea*, are duplicate to those of the 1st one. Thus, we only need to update the associated attributes, and meanwhile create a new leaf node to the FTCTree. Figure 1 (b) gives the updated *FTCTree* structure.

Finally, since the interval between the purchase time time 20180919 of the 3rd record and the most recent one 20190112 is between 2–4 months, we follow Table 2 to adopt the 2-level coarse categories *clothes* and *children clothes*. Figure 1(c) gives the final FTCTree.

3.2 FTCTree Distance

Given two FTCTrees R_1 and R_2, we compute the distance between R_1 and R_2 by the idea of Jaccard distance. That is, we need to create the intersection tree (IR) and union tree (UR) between R_1 and R_2. Unlike FTCTree, each node in UR and IR maintains the category and frequency only, but not purchase time. The reason is as follows. Since the data decaying mechanism on purchase time has been applied to FTCTree, the constructed FTCTree can implicitly identify the purchase time, and we thus do not maintain purchase time in IR and UR.

Now given IR and UR, we leverage the ratio between IR and UR and compute the average ratio of IR over UR as the distance. Given the two example FTCTrees in Figure 1(c-d), we show the intersection tree IR and union tree UR in Figure 2.

Based on the IR and UR, we compute the distance $Dist(IR, UR)$ between IR and UR as follows. First for a node $v \in IR$, we define the similarity $Sim(v, UR)$ between v and UR. Next, for all nodes V_l of a certain level l in IR, we compute the similarity $Sim(V_l, UR)$ between V_l and UR. Finally, we compute $Dist(IR, UR)$ with help of the level-wise normalized similarity.

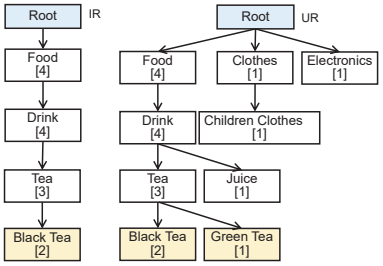

Figure 2: Intersection Tree and Union Tree

$$Sim(v, UR) = \frac{freq(v)}{\sum_{u \in UR} freq(u)} \quad (1)$$

$$Sim(V_l, UR) = \frac{1}{|V_l|} \sum_{v \in V_l} Sim(v, UR) \quad (2)$$

$$Dist(IR, UR) = 1 - \sum_{l=1}^{H(IR)} Sim(V_l, UR) \times w_l \quad (3)$$

In the equations above, to compute $Sim(v, UR)$, we need to find those nodes $u \in UR$ who share the same parent node as v and then compute $Sim(v, UR)$ by the ratio between the frequency of v and the accumulated frequencies of all such nodes u. Next, for a certain level l in IR and all nodes $V_l \in I$ having the level l, we compute $Sim(V_l, UR)$ by the average of the similarities $Sim(v, UR)$ for all nodes $v \in V_l$. Finally, for each level $1 \le l \le H(IR)$, we compute $Dist(IR, UR)$ by a normalized similarity $Sim(V_l, UR) \times w_l$ where the weight $w_l = \frac{l}{\sum_{h=1}^{H(IR)} h}$, where $H(IR)$ is the total height of the IR tree.

For example in Figure 2, for the node *Food* in IR, we compute the similarity $Sim(Food, UR) = \frac{4}{4+1+1} = \frac{2}{3}$. For the level $l = 2$, the similarity $Sim(V_2, UR) = \frac{3}{4}$, and finally $Dist(IR, UR) = 1 - (\frac{2}{3} \times \frac{1}{10} + \frac{4}{4} \times \frac{2}{10} + \frac{3}{4} \times \frac{3}{10} + \frac{2}{3} \times \frac{4}{10}) = \frac{29}{120}$.

3.3 FTCTreeClust

We now present the parameter-free FTCTreeClust algorithm to segment the customers $\mathcal{R} = \{R_1, ..., R_m\}$ in a hierarchical divisive manner. The algorithm initially assumes that all FTCTrees belong to a single cluster. In each successive iteration, via the Bayesian information criterion (BIC) [7], we determine whether or not to split a certain cluster into two smaller clusters. We repeat the process until every cluster cannot be split eventually.

During the process, we use the algorithm *GetCT* (see Alg. 1) to extract a representative centroid tree (*ct* for short) for each cluster. The key idea of *GetCT* is to use the union tree of all member FTCTrees in the cluster to be the initial *ct*. At each iteration, we delete the nodes with the least purchase frequency in turn and sum the distances between every FTCTree and *ct*. Lastly, the centroid tree with the minimum distance is taken as the final centroid tree.

Algorithm 1 gives the pseudocode to extract the centroid tree *ct* in a cluster of FTCTrees. First, we create a union tree *utree*. Then we initiate $freq = 1$ and $mindist = +\infty$. After that, the two variants $freqStep$ and $freqEnd$ indicate the growth step of $freq$ at each iteration and the stopping criterion, respectively. At each iteration (lines 4-11), we update the union tree *utree* by deleting those nodes whose frequency values are smaller than $freq$ (line 5) and next calculate the sum of distance between *FTCTree* and *utree* (line 6). If the current sum is less than the sum in the previous iteration, we use *ct* to retain the corresponding union tree as the current centroid tree (lines 7-9). Finally when the stopping condition is met, we then return the current centroid tree as the final one.

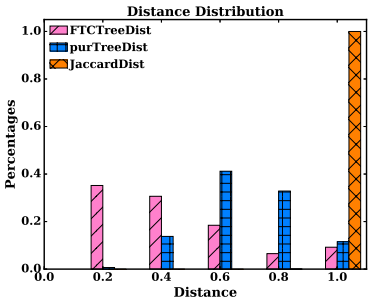

Figure 3: Distance Distribution

Algorithm 1 GetCT: extract centroid tree of a cluster

Input: C: a cluster of FTCTrees
Output: ct: centroid tree of the cluster

1: $utree$ = unionTree(C);
2: $freq = 1$; $mindist = +\infty$;
3: $freqStep$ = AvgFreq($utree$), $freqEnd$ = MaxFreq($utree$);
4: **while** $freq \le freqEnd$ **do**
5: $utree = update(utree, freq)$;
6: $dist = \sum_{FTCTree \in C} Dist(FTCTree, utree)$;
7: **if** $dist < mindist$ **then**
8: $mindist = dist, ct = utree$;
9: **end if**
10: $freq = freq + freqStep$;
11: **end while**
12: **return** ct;

4 EVALUATION

In this section, we evaluate our approach against seven counterparts on a real transaction dataset from a large China supermarket. The dataset includes 16393 transaction records range from February 1, 2016 to July 31, 2016. Table 1 gives three examples of transaction records.

4.1 FTCTree Distance

We compare the proposed FTCTree distance against Jaccard distance and purTree distance in terms of the distribution of computed distance on our dataset. As shown in Figure 2, almost all Jaccard distance values are highly concentrated within the interval $[0.9, 1.0]$. This distance distribution indicates that almost all customers are with almost the exactly same distance and hard to segment them. The purTree distance could lead to better results and is distributed across the interval $[0.2, 1.0]$. Compared with purTree distance, the proposed FTCTree distance is instead diversely and evenly distributed across the interval $[0.2, 1.0]$, and is thus comfortable to segment customers into clusters, leading to better clustering quality.

4.2 Clustering Quality

We use Silhouette Coefficient (*SC*) and Compactness (*CP*) to measure clustering quality. Firstly, *SC* involves the two averages $a(i)$ and $b(i)$, where $a(i)$ computes the the average distance from sample i to other samples in the same cluster, and $b(i)$ is the

average distance from sample i to all samples in another cluster C_j. When SC is bigger and closer to 1, a sample is well matched to its own cluster and poorly matched to neighboring clusters. Secondly, CP is the average distance from each member $t_i \in C_k$ in the cluster C_k to the cluster centroid ct_k of C_k. A *smaller* CP indicates *better* cluster performance.

$$SC = \frac{1}{n}\sum_{i=1}^{n}\frac{b(i)-a(i)}{max\{a(i),b(i)\}}$$

$$CP = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{t_i \in C_k} \parallel t_i - ct_k \parallel$$

Table 3: Clustering Quality

		Distance	K	SC	CP
1	FTCTreeClust	FTCTree dist	15	0.28513	0.20242
2	PurTreeTxmeans	PurTree dist	1	-	-
3	Practical [2]	Jaccard dist	13	0.02022	-
4	Txmeans [6]	Jaccard dist	1	-	-
5	Tkmeans [4]	Jaccard dist	15	0.01556	0.86761
6	Coolcat [1]	Jaccard dist	15	0.00036	-
7	Rock [5]	Jaccard dist	15	-0.00049	-
8	Clope [9]	Jaccard dist	1	-	-

In Table 3, we evaluate our work FTCTreeClust against seven counterparts. The first four approaches are parameter-free and the other four need to predefine the number K of clusters or other parameters. For Tkmeans and Practical, we replace the *basket* with the product list purchased by customers. From this table, we have the following finding.

- Tree-based clustering methods include purTreeTxmeans and FTCTreeClust. Here, we improve Txmeans by replacing the originally used vectors with the purTree structure, and thus name this improved approach by purTreeTxmeans. FTCTreeClust achieves significantly better SC and CP values than purTreeTxmeans.
- Among three centroid-based methods purTreeTxmeans, Tkmeans [4] and FTCTreeClust, FTCTreeClust outperforms Tkmeans in terms of greater SC and smaller CP.
- Parameter-free methods include Practical [2], Txmeans, and the proposed FTCTreeClust. Note that Txmeans cannot segment customers because it is originally used to segment the transaction data of a certain customers into multiple clusters of transaction records. In particular, the Jaccard distance used by Txmeans is insufficient for achieving good clustering result (that is verified by Figure 3). Practical leads to better SC than Txmeans but is still worse than FTCTreeClust. Since Practical does not generate centroid trees and Txmeans has only one cluster, these tree approaches do not have the corresponding SC values.
- In terms of the remaining methods such as Coolcat [1], Rock [5] and Clope [9], for fairness, we predefine $K = 15$ that is just the one automatically tuned by FTCTreeClust. Nevertheless, these works still suffer from much smaller SC than FTCTreeClust.

Next, we randomly choose 318 customers and accumulate their transaction data by month. After that, we calculate the silhouette coefficient (SC) when the 318 customers are clustered based on the accumulated transaction data. As shown in Figure4, FTCTree achieves the biggest SC among all clustering algorithms. Moreover, the sc values consistently grows with more transaction data.

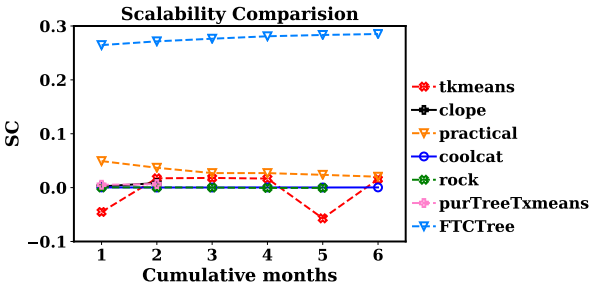

Figure 4: Scalability Comparision

Instead, the SC values of other approaches are significantly small on various transaction size. Especially, clope and purTreeTxmeans can't work on more than two-month transaction data.

5 CONCLUSION

In this paper, to study the customer clustering problem, we design a FTCTree structure to represent the transaction data purchased by a customer, measure the FTCTree distance by the proportion of intersection trees to union trees, and give a novel parameter-free clustering method FTCTreeClust. Our preliminary experiment on a real transaction dataset indicates that FTCTree distance is much better than two other distance metrics and FTCTreeClust outperforms the state-of-the-arts in terms of much better SC and CP. As the future work, we continue the work on the parameter-free clustering algorithm on new measurement of customer similarity and evaluate the algorithm on more real world datasets.

REFERENCES

[1] Daniel Barbará, Yi Li, and Julia Couto. 2002. COOLCAT: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*. ACM, 582–589.

[2] Eugenio Cesario, Giuseppe Manco, and Riccardo Ortale. 2007. Top-down parameter-free clustering of high-dimensional categorical data. *IEEE Transactions on knowledge and data engineering* 19, 12 (2007), 1607–1624.

[3] Xiaojun Chen, Yixiang Fang, Min Yang, Feiping Nie, Zhou Zhao, and Joshua Zhexue Huang. 2018. PurTreeClust: A Clustering Algorithm for Customer Segmentation from Massive Customer Transaction Data. *IEEE Trans. Knowl. Data Eng.* 30, 3 (2018), 559–572.

[4] Fosca Giannotti, Cristian Gozzi, and Giuseppe Manco. 2002. Clustering transactional data. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 175–187.

[5] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 2000. ROCK: A robust clustering algorithm for categorical attributes. *Information systems* 25, 5 (2000), 345–366.

[6] Riccardo Guidotti, Anna Monreale, Mirco Nanni, Fosca Giannotti, and Dino Pedreschi. 2017. Clustering individual transactional data for masses of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 195–204.

[7] Dan Pelleg, Andrew W Moore, et al. 2000. X-means: extending k-means with efficient estimation of the number of clusters.. In *Icml*, Vol. 1. 727–734.

[8] Ke Wang, Chu Xu, and Bing Liu. 1999. Clustering transactions using large items. In *Proceedings of the eighth international conference on Information and knowledge management*. ACM, 483–490.

[9] Yiling Yang, Xudong Guan, and Jinyuan You. 2002. CLOPE: a fast and effective clustering algorithm for transactional data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 682–687.