Why did my Consumer Shop? Learning an Efficient Distance Metric for Retailer Transaction Data

Yorick Spenrath[1 [0000-0003-0908-9163]](✉), Marwan Hassani[1 [0000-0002-4027-4351]], Boudewijn van Dongen[1 [0000-0002-3978-6464]], and Haseeb Tariq[2 [0000-0003-0756-3714]]

[1] Eindhoven University of Technology, {y.spenrath, m.hassani, b.f.v.dongen}@tue.nl
[2] BrandLoyalty, haseeb.tariq@brandloyalty-int.com

Abstract. Transaction analysis is an important part in studies aiming to understand consumer behaviour. The first step is defining a proper measure of similarity, or more specifically a distance metric, between transactions. Existing distance metrics on transactional data are built on retailer specific information, such as extensive product hierarchies or a large product catalog. In this paper we propose a new distance metric that is retailer independent by design, allowing cross-retailer and cross-country analysis. The metric comes with a novel method of finding the importance of categories of products, alternating between unsupervised learning techniques and importance calibration. We test our methodology on a real-world dataset and show how we can identify clusters of consumer behaviour.

Keywords: Distance metric · Transaction categorization · Clustering · Optimization

1 Introduction

The analysis of consumer transaction data is an important task in creating value out of shopper interactions with online and physical stores. In recent years, the literature has extended rapidly on analyzing transactions to provide retailers with recommendations on a variety of topics [1, 7, 15, 25, 26]. One important part of such an analysis is to be able to tell how transactions of a single consumer and between several consumers relate. This paper is part of a larger project on consumer journey analysis and optimization. We intend to mine consumer behaviour from transactional data provided by retailers, to improve effectiveness of promotional programs. The starting point of this project is classifying each individual visit of a consumer to the store; by analyzing the corresponding receipt, and determining the *purpose* of their visit.

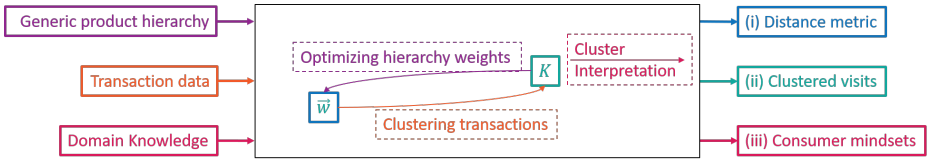

Fig. 1. Overview of the contributions in this paper

While receipts can be hand-labelled using domain knowledge, the vast quantity of data makes this infeasible. For the same reason, supervised learning strategies will be subject to overfitting small quantities of labelled data. To overcome this, we resort to unsupervised learning strategies. Our starting point is a dataset of receipts; indicating which product categories (see Figure 3) are bought during a trip to the retailer. For meaningful comparisons we require a distance metric between two receipts. First, this allows us to present the domain experts with representative examples in the dataset, and as such reduce the overfitting in supervised learning methods. Second, this allows us to use neighbourhood-based machine learning models such as k-medoids [17, 21], spectral clustering [16] or hierarchical clustering [20], all of which can be used in an unsupervised setting as well. In this paper we present a novel framework for learning a distance metric that measures the similarity between consumer receipts. Our introduced framework learns the weights of the different components of the distance metric by optimizing the clusters of receipts. The overview of our contribution is presented schematically in Figure 1.

In this paper, we make the following contributions: (i) we consider relations between products on a generic hierarchy which is fine-tuned by assigning different weights to different parts of the hierarchy. To find the best weights, we use clustering, but for this clustering, the weights serve as input again, i.e. (ii) we provide a fixed-point algorithm to find both, the weights and the clustering, together. Finally, (iii) we provide extensive experimental evaluation of our work using domain expert knowledge in a setting where huge quantities of unlabelled data is available. Part of this domain expert knowledge are the mindsets of consumer behaviour which we explain next.

Table 1. The 5 shopper mindsets identified by domain experts.

`BULK`	Buying large quantities of specific products to optimize discount.
`PLANNED`	Plan ahead and buy everything needed for the week in one go.
`FRESH`	Smaller trips with the key motivation of being flexible in when and what to eat, in particular for fresher fruits, vegetables and meats.
`QUALITY`	Going to specific stores for specific products, like going to the butcher for meat.
`ON THE GO`	Also smaller trips, but with an on-the-go character.

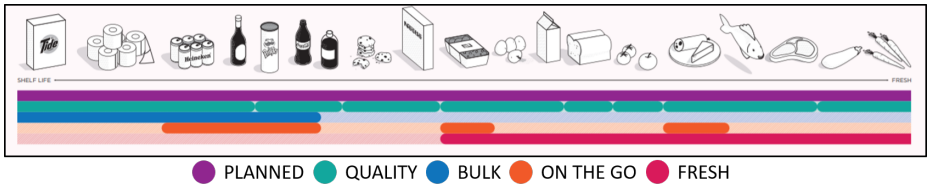

Fig. 2. Examples of products bought in each mindset.

One important challenge of most clustering algorithms is determining the number of clusters k one expects. Setting the value of k needs careful consideration; finding the best number of clusters is a difficult task in many unsupervised learning projects [10, 13, 19]. Considering that our end goal is to be able to tell for each receipt (and accompanying visit) *why* a shopper visits the retailer; it makes sense to base k on the number of visit types we expect. Earlier research[3] by domain experts has indicated five main reasons why shoppers visit retailers. Each of these 'mindsets' are an explanation of what moves consumers to a store, and where decisions are made. We have summarized these mindsets in Table 1, and a visual example of the products in each mindset is represented in Figure 2.

The rest of this paper is organised as follows. We first discuss our framework in Section 2. Following this; we discuss experiments on a real-world dataset in Section 3. We then discuss how our solution relates to existing literature on transaction distance metrics in Section 4. We conclude the paper in Section 5.

2 A Framework For Learning The Distance Metric

In this section we discuss our framework for learning the distance metric. We first present the idea and formalization of the metric in Sections 2.1 and 2.2. We present a novel idea for fine-tuning the weights of the distance metric in Section 2.3 and formalize it in Section 2.4

2.1 Weighing A Product Hierarchy

The concept of a hierarchy is fundamental in our approach. This hierarchy relates categories of products to each other. An example of a hierarchy is presented in Figure 3. The hierarchy consists of super-categories (3 in this example), each of which consists of categories (2 in each super-category in the example). By design, the hierarchy is retailer independent and can as such be applied to all consumer transactional data from supermarkets.

We make an abstraction from the actual receipt: instead of using all information on the products, their quantities and their price; we only consider which

[3] The full article on this study is found at `https://medium.com/icemobile/mindset-moments-a-new-way-to-pinpoint-purchasing-decisions-18c58a574c02`.

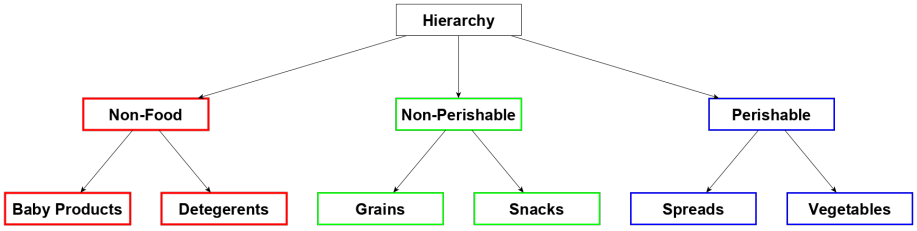

Fig. 3. Example of a retailer hierarchy. This hierarchy consists of three super-categories, each of which consists of two categories.

(super-)categories are included in the receipt. This abstraction allows for numerous improvements in computation time while, as we show in Section 3, it is still expressive enough to make a sensible distance metric and discover clusters of consumer behaviours. For our explanation we consider a small example of two receipts. Receipt a has two Non-Food categories, Baby Products and Detergents, and one Non-Perishable category, Grains. Receipt b has the categories Baby Products, Snacks and Spreads. The receipts, along with the distance metric computations are shown in Figure 4.

We make a total of four comparisons between these two receipts. Each comparison is a Jaccard[4] distance on a specific subset of the hierarchy. The first three comparisons are within each super-category. For the super-category *Non-Food* we have one common category (Baby Products) and two total categories (Baby Products and Detegergents) over the two receipts. The Jaccard distance for *Non-Food* is then $1 - \frac{1}{2} = \frac{1}{2}$. We can similarly analyze *Non-Perishable* and *Perishable*, they both have a Jaccard distance of 1. Despite having the same Jaccard distance, we do not capture the fact that both receipts do contain a *Non-Perishable* product, whereas this does not hold for *Perishable*. To overcome this we make a fourth comparison on the super-category level. The receipts have two super-categories in common, and three in total. As such, the *inter-super-category* distance is $1 - \frac{2}{3} = \frac{1}{3}$. We finally combine the four values using a weighted average. The weight of each distance allows domain expert knowledge to put emphasis on certain super-categories or can be fine-tuned to available transaction data.

[4] Given two sets $\mathcal{S}_1$ and $\mathcal{S}_2$, the Jaccard distance [23] between them is defined as $1 - \frac{|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|}$.

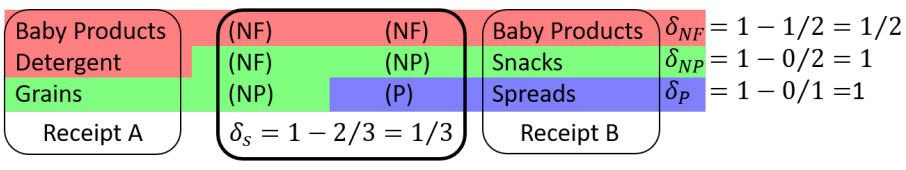

Fig. 4. Example receipts, along with computations for the distance metric.

2.2 Formalization

We now generalize the given example in a formal manner, all notations are summarized in Table 2. Let $\mathcal{C} = C_1 \cup C_2 \cup \ldots \cup C_h$ be a set of *categories*. Each C_i is a *super-category* containing categories. We further have that $C_i \cap C_j = \emptyset$ for all $i \neq j$, i.e. each category is in exactly one super-category, and $C_i \neq \emptyset$ for all i, i.e. each super-category is non-empty. The set $\{C_1, C_2, \ldots C_h\}$ is called a *hierarchy*. Note that the hierarchy is a partition over $\mathcal{C}$. The details of the used hierarchy are presented in Section 3.1. Without loss of generality, our target metric in this work considers a product hierarchy of height 2.

Let $\mathcal{D} \subseteq \{0,1\}^{|\mathcal{C}|}$ be a set of receipts. A *receipt* is a description of the categories purchased in a *visit*. More specifically, for a receipt $\vec{a} \in \mathcal{D}$ we have $a_c = 1$ if a product of category c was bought, and $a_c = 0$ otherwise. For a receipt $\vec{a}$ we also need to know which super-categories were bought. For this purpose we define $s(\vec{a}, i)$ which tells us if any category in C_i was included in $\vec{a}$, formally:

$$s(\vec{a}, i) = \begin{cases} 0 & \forall c \in C_i : a_c = 0 \\ 1 & otherwise. \end{cases} \tag{1}$$

[removed] Here was a story about how we have $\mathcal{R}$ and $\mathcal{D}$. I don't think it adds anything to the paper; is slightly confusing and mainly costs quite some space.

This function extends to two receipts, for $\vec{a}, \vec{b} \in \mathcal{D}$, we define $s(\vec{a}, \vec{b}, i)$ which tells us if any category in C_i was included in $\vec{a}$ or $\vec{b}$, formally:

$$s(\vec{a}, \vec{b}, i) = \begin{cases} 0 & \forall c \in C_i : a_c = b_c = 0 \\ 1 & otherwise. \end{cases} \tag{2}$$

Returning to the example of Section 2.1, the hierarchy consists of:

- C_1 =*Non-Food* = {*Baby Products, Detergents*}
- C_2 =*Non-Perishable* = {*Grains, Snacks*}
- C_3 =*Perishable* = {*Spread, Vegetables*}.

Furthermore[5], $\vec{a} = (1,1,1,0,0,0)$ and $\vec{b} = (1,0,0,1,1,0)$. Some examples of s include $s(\vec{a}, 3) = 0$, $s(\vec{b}, 2) = 1$, and $s(\vec{a}, \vec{b}, i) = 1$ for all i.

Given receipts $\vec{a}$ and $\vec{b}$, we define a total of $h+1$ distances between them. The first h each apply to a single super-category, these are *intra-super-category* distances as they make a comparison within a super-category. For a super-category C_i, the distance between $\vec{a}$ and $\vec{b}$ is defined as

$$\delta_i(\vec{a}, \vec{b}) = \begin{cases} 1 - \frac{\sum_{c \in C_i} \vec{a}_c \cdot \vec{b}_c}{\sum_{c \in C_i} \vec{a}_c + \vec{b}_c - \vec{a}_c \cdot \vec{b}_c} & \text{if } s(\vec{a}, \vec{b}, i) = 1 \\ 0 & \text{if } s(\vec{a}, \vec{b}, i) = 0 \end{cases} \tag{3}$$

Put differently, the distance is the Jaccard distance between the set of categories in C_i included in the receipts. Because the Jaccard distance between empty sets is invalid, we assign the value 0 if both receipts are missing super-category C_i. We explain the choice of this value below. We next define the *inter-super-category*

[5] We order (super-)categories alphabetically in $\vec{a}$, $\vec{b}$ and s

distance $\delta_s(\vec{a}, \vec{b})$. It is a Jaccard distance on the super-categories: comparing which super-categories are included in $\vec{a}$ and $\vec{b}$. For this we can make use of s as defined in Equations 1 and 2.

$$\delta_s(\vec{a}, \vec{b}) = 1 - \frac{\sum_{i=1\ldots h} s(\vec{a}, i) \cdot s(\vec{a}, i)}{\sum_{i=1\ldots h} s(\vec{a}, \vec{b}, i)}. \tag{4}$$

We define the distance between $\vec{a}$ and $\vec{b}$ as the weighted average of δ_s and all δ_i for which either $\vec{a}$ or $\vec{b}$ has products from C_i. Put differently, we ignore the super-categories for which neither receipt includes categories. This is because there is no concept of (dis)similarity for that super-category. This decision is partly inspired by [8]. Let w_s be the weight for δ_s in Equation 4 and let w_i be the weight for each δ_i in Equation 3. Let $\vec{w} = (w_s, w_1, w_2, \ldots w_h)$. Our assignment of 0 for invalid intra-super-category distances allows us to write:

$$\delta_{\vec{w}}(\vec{a}, \vec{b}) = \frac{w_s \cdot \delta_s(\vec{a}, \vec{b}) + \sum_{i=1\ldots h} w_i \cdot \delta_i(\vec{a}, \vec{b})}{w_s + \sum_{i=1\ldots h} w_i \cdot s(\vec{a}, \vec{b}, i)}. \tag{5}$$

Note that any scalar multiple of $\vec{w}$ results in the same distance metric. When considering values for $\vec{w}$ we add the restriction that the weights need to sum up to 1, though Equation 5 can still be computed without this restriction.

Table 2. Symbols and parameters

Symbol	Description
C	categories
C_i	super-category
h	number of super-categories
$\mathcal{D}$	set of receipts
$\vec{a}$	(a_c) for $c \in C$. We have $a_c = 1 \Leftrightarrow$ category c was bought in receipt $\vec{a}$
$s(\vec{a}, i)$	whether super-category i was bought in receipt $\vec{a}$ (1) or not (0)
$s(\vec{a}, \vec{b}, i)$	whether super-category i was bought in in either $\vec{a}$ or $\vec{b}$ (1) or not (0)
δ_x	distance of super-category C_i (δ_i) or the inter-super-category (δ_s)
$\vec{w}$	$(w_s, w_1, w_2, \ldots w_h)$, weights of each distance
K	$\{K_1, K_2, \ldots K_k\}$ clustering (partition) over $\mathcal{D}$
$\Phi(K, \vec{w})$	an internal evaluation measure, a quantitative value of the quality of a clustering K given weights $\vec{w}$

[removed] w^* is no longer a thing

2.3 Finding Optimal Weights

To find $\vec{w}$ we use a clustering over $\mathcal{D}$ and an internal evaluation measure [11]. Given a clustering, we want to find the weights that minimize[6] the internal evaluation measure on the clustering.

Let $K = \{K_1 \cup K_2 \cup \ldots \cup K_k\}$ be a partition over $\mathcal{D}$. K is a clustering of the receipts. Using a clustering and a value of $\vec{w}$ we can define an internal evaluation measure $\Phi(\vec{w}, K)$ which describes the quality of clustering K with respect to weights $\vec{w}$. Following the concept of internal evaluation measures, Φ decreases if receipts within a cluster are closer together (lower value for Equation 5) and if points in different clusters are further apart. For a given K, we want to find $\vec{w}$ that minimizes $\Phi(\vec{w}, K)$. While our approach can deal with any internal evaluation measure that considers compactness and separation [11], in this paper we use the one defined in [7], which was used there for a similar studies:

$$\Phi(\vec{w}, K) = \log\left\{\sum_{K_i \in K} \frac{1}{2|K_i|} \sum_{\vec{a},\vec{b} \in K_i} \delta_{\vec{w}}(\vec{a}, \vec{b})\right\} - \log\left\{\sum_{\vec{a},\vec{b} \in \mathcal{D}} \delta_{\vec{w}}(\vec{a}, \vec{b})\right\}. \quad (6)$$

A large part of the computation of Equation 6 consists of summing over the distance between all points. This can be costly, especially because we need to do this for every $\vec{w}$ we wish to evaluate. We reduce the time complexity by precomputing parts of Equation 6, the details are found in the repository discussed in Section 3.

The problem however, is that we *do not* have such a clustering K; as finding it requires a distance metric on $\mathcal{D}$ or a large set of labelled data. This means we need to find the solutions to two dependent problems: finding a clustering using the distance metric with $\vec{w}$; and finding the optimal $\vec{w}$ using a clustering. The solution to this is presented in the next section.

2.4 Finding Fixed Points

The target is to find *fixed points* that solve the two problems simultaneously. In other words, we need to find a combination $\Lambda := (\vec{w}, K)$ such that: (1) a clustering algorithm applied to $\mathcal{D}$ with a distance metric as defined by $\vec{w}$ and Equation 5 finds clustering K, and (2) Equation 6 with K is minimized by $\vec{w}$. A dataset may have multiple *fixed points* that satisfy these conditions. In this section, we explain first how we can find any of those fixed points, and then explain which of them is *the* fixed point of a dataset $\mathcal{D}$.

Finding *a* fixed point We alternate between solving (1) the clustering given a set of weights and (2) the optimization given a clustering, until the solutions do not change. The starting point for this is a random set of weights $\vec{w}^{(0)}$. The

[6] Without loss of generality, we discuss minimization of the internal evaluation measure, the problem and solution is similar for internal evaluation measures that need to be maximized.

details of finding $\vec{w}$ and K simultaneously are presented in Algorithm 1. All input parameters are summarized in Table 3. The algorithm requires the dataset, an initial set of weights, and k as the number of clusters to find. Furthermore, we need two parameters for the stopping criteria: ϵ to check if weights vary significantly between two consecutive iterations, and n_{max} to set a maximum number of iterations. Next, we need two separate functions. `cluster` takes a dataset, a distance metric and an initial set of medoids to find the clusters and their medoids with the lowest internal distance. `minimize` takes a function and provides a numerical minimization. We finally need a parameter r to break cyclicity as explained below.

We initiate the first set of k medoids (as needed by `cluster`) in Line 1, using the `kmeans++` algorithm [2]. We use this to find an initial clustering in Line 2. In Line 4 through Line 10 we find the optimal weights and update the clustering. The weights are updated in Line 6. If the difference between weights of consecutive iterations is smaller than ϵ or if we reached the maximum number of iterations, we terminate the algorithm by saving the final weight as $\vec{w}$ and the final clustering as K. If neither stopping criterion is met we move on to the clustering step. During initial experiments, it turned out that sometimes the algorithm gets stuck in a cycle of several iterations (returning to a combination of weights and clusters which was found at an earlier iteration). To solve this, we increase all weights randomly by up to $r \leq \epsilon$. We then use these adapted weights for the clustering computation.

Selecting *the* fixed point We apply the above procedure n_{rep} times, seeding the algorithm with a different, random set of weights $\vec{w}^{(0)}$ each repetition. Each time this results in one fixed point of the dataset. A fixed point can be found multiple times, formally let the *support* of a fixed point Λ^X be the fraction of the n_{rep} runs that ended with Λ^X. We then define Λ, *the* fixed point of $\mathcal{D}$, as

Algorithm 1: Procedure to find $\vec{w}^*$, given an initial $\vec{w}^{(0)}$

input : A dataset $\mathcal{D}$, an initial $\vec{w}^{(0)}$, $\epsilon \geq 0$, $r \geq 0$, n_{max}, k, `cluster`, `minimize`
output : The optimal weights $\vec{w}^*$ and medoids m

1 $m^{(-1)} = \texttt{kmeans++}\left(\mathcal{D}, \delta_{\vec{w}^{(0)}}, k\right)$
2 $K^{(0)}, m^{(0)} = \texttt{cluster}\left(\mathcal{D}, \delta_{\vec{w}^{(0)}}, m^{(-1)}\right)$
3 $n = 0$
4 **while** *True* **do**
5 $\quad n \leftarrow n + 1$
6 $\quad \vec{w}^{(n)} \leftarrow \texttt{normalize}\left(\texttt{minimize}\left(\Phi(K^{(n-1)}, \vec{w})\right)\right)$
7 $\quad$ **if** $\forall i = 0 \ldots h : \left|\vec{w}_i^{(n)} - \vec{w}_i^{(n-1)}\right| < \epsilon \vee n = n_{max}$ **then**
8 $\quad\quad$ **Return** $\vec{w}^{(n)}$ and $K^{(n)}$
9 $\quad \vec{w}^{(n')} \leftarrow \texttt{normalize}\left(\vec{w}^{(n)} + U(0, r)\right)$
10 $\quad K^{(n)}, m^{(n)} \leftarrow \texttt{cluster}\left(\mathcal{D}, \delta_{\vec{w}^{(n')}}, m^{(n-1)}\right)$

Table 3. Parameters for Algorithm 1

Symbol	Description
$\vec{w}^{(0)}$	initial weight vector
ϵ	maximum change of $\vec{w}$ for termination
r	random change applied to $\vec{w}$ before clustering to avoid cyclicity
n_{max}	maximum number of iterations
k	the number of clusters to find in the clustering step

the fixed point with the highest support. This entire procedure is schematically presented in Figure 5.

3 Experimental Evaluation

In this section we discuss an experimental evaluation of our method. We describe the general settings for all experiments in Section 3.1, and discuss the experiments and their the results in Sections 3.2 through 3.4. The entire code, experimental evaluation, and the used datasets are provided at `https://github.com/YorickSpenrath/PKDD2020`.

3.1 Experimental Settings

Our hierarchy consists of 29 categories, divided over 7 super-categories, shown in Table 4. We mapped all products of the retailer onto this generic product hierarchy with the help of domain expert knowledge by aligning the lowest level product groups from the original retailer to one of the 29 categories. We set ϵ to 0.01, which is less than one-tenth of a weight if all weights would be distributed equally ($\frac{1}{8}$). We set r to one-tenth of ϵ at 0.001, to not have it interfere with stability but at the same time still have effect on breaking cyclicity. We set n_{max} to 200 in order to prematurely end the algorithm before it consumes too much computation time.

For k we base our choice on the number of mindsets as discussed in Section 1. By design; our methodology does not account for the quantity and price of products; and it is therefore not possible to distinguish the mindset `BULK` (Table 1) from the other mindsets. As such we are explicitly searching for $k = 4$ clusters.

For `cluster` we use the Voronoi [17] algorithm, which we implemented using Numba [14]; favouring lower time complexity over a better clustering, different to

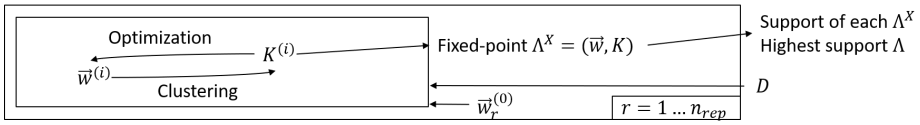

Fig. 5. Overview of the proposed algorithm.

Table 4. Hierarchy used in the experiments. (N)P stands for (Non-)Perishable

Super-Category	Categories
Non Food	Pet articles, Personal care, Non food, Baby products, Washing and cleaning
Perishable A	Bread, Desserts, Milk, Fruits and vegetables P, Eggs, Meat and fish P, Cheese
Perishable B	Prepared meals P, Spread P, Carbs P, Soft drinks P, Condiments P
Non Perishable A	Condiments NP, Grains, Carbs NP, Meat and fish NP, Fruits and vegetables NP, Prepared meals NP
Non Perishable B	Snacks, Sweets
Non Perishable C	Spread NP, Hot drinks
Non Perishable D	Alcohols and tobacco, Soft drinks NP

for instance [21]. For `minimize` we use the differential evolution method proposed by [22] and implemented in SciPy [24]. We finally execute Algorithm 1 a total number of $n_{rep} = 1000$ times.

3.2 Weights Learning

Apart from finding the optimal weights, we also want to verify that it is independent of the used dataset. As such, we apply the method of Section 2.3 to 100 datasets each having $|\mathcal{D}_i| = 1000$ visits. This computes the support of each of the fixed points of $\mathcal{D}_i$. Although these fixed points will have different medoids for each dataset, we can still compare the weights found for each $\mathcal{D}_i$. We compute the average and confidence interval of the support of each set of weights; the 5 weights with the highest average support are reported in Table 5.

We can see that the most supported set of weights is a clear winner: it is found almost two and a half times as often as the second most supported set of weights. This set of weights gives most importance on the Non Perishable B super-category; the one consisting of Sweets and Snacks. We further learn that Perishable B and the inter-super-category distance are also important for make clusters in our data.

3.3 Qualitative Cluster Analysis

The previous section focussed on the final weights; which are in fact only one part of our fixed points. In this section we analyze the contents of the discovered clusters using domain expert knowledge. The goal is to provide descriptive statistics on the contents of the clusters: which (super-)categories are characteristic for which clusters, and which (super-)categories are mostly missing.

In order to do so, we need a way to combine the clusters from each $\mathcal{D}_i$. This is schematically presented in Figure 6. Formally, (1) let $\Lambda_i = (\vec{w}_i, K_i)$ be the fixed point for $\mathcal{D}_i$ and let m_i be the k medoids of K_i. We (2) combine all medoids as

Table 5. The most frequently found weights, showing the percentage of repetitions that found this combination of weights, averaged over the 100 datasets, with their 95%-confidence intervals.

support	43.5 ± 1.2%	16.2 ± 1.1%	7.5 ± 0.6%	4.0 ± 0.5%	3.2 ± 0.5%
Non Food	0.00	0.00	0.11	0.18	0.28
Non Perishable A	0.05	0.10	0.03	0.03	0.08
Non Perishable B	0.44	0.20	0.25	0.25	0.09
Non Perishable C	0.04	0.07	0.04	0.12	0.12
Non Perishable D	0.01	0.33	0.25	0.14	0.01
Perishable A	0.06	0.00	0.01	0.01	0.00
Perishable B	0.17	0.13	0.07	0.03	0.30
w_s	0.22	0.16	0.24	0.24	0.11

$M = \bigcup_{i=1}^{100} m_i$ and all weights as $\vec{w} = \frac{1}{100} \sum_{i=1}^{100} \vec{w}_i$. We define the latter because the weights of the fixed points are identical up to the precision presented in Section 3.2. We then (3) use `cluster` to create a clustering K over M using the distance metric with weights $\vec{w}$. (4) All receipts in the same cluster in K are given the same label, and subsequently (5) for each $\mathcal{D}_i$, each cluster of K_i is given the label as their respective medoid. For each of the 400 clusters we compute how often a (super-)category is included in a receipt, and aggregate these values over clusters with the same labels. The results of this are presented in Table 6

We analyzed the table together with the same domain experts that initially researched the mindsets of Figure 2 and created the hierarchy of Table 4. Despite the abstraction of receipts to their categories (rather than quantities and prices), we can still distinguish properties of the four mindsets in the clusters. K_1 consists primarily of super-category Perishable A: fresh fruits, vegetables, meat and fish. This is typical for the `FRESH` mindset. Similarly; it has a relatively low inclusion of the Non Perishable categories. Compared to the other clusters, K_2 has a rather low inclusion of Perishable A, while having a high inclusion of Non Perishable D: alcohols, tobacco and non-perishable soft-drinks. One explanation for these two values is a `QUALITY` mindset, where people prefer specific stores like a butcher or greengrocer for the products from Perishable A, while preferring the retailer for their alcohols and tobacco. The value for Non Perishable A also supports this; as such non-perishable products from these categories are typically not found at butchers and groceries. However; within the `QUALITY` mindset; visits that focus specifically on meats or vegetables (but not both) do also exist. It appears this type of `QUALITY` mindset was not found. The difference between K_3 and K_4 is

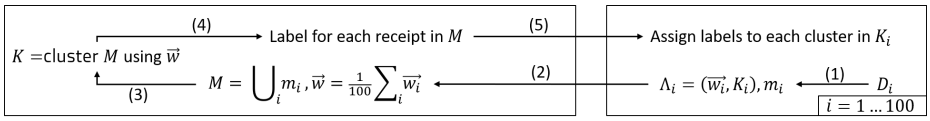

Fig. 6. Overview of the cluster analysis learning experiment.

Table 6. Descriptive Statistics of each cluster. The abbreviations are `CONNP`: Condiments NP, `FnVP`: Fruits and vegetables P, `MIL`: Milk, `MnFP`: Meat and fish P, `PRENP`: Prepared meals NP, `SNA`: Snacks, `SOFNP`: Soft drinks NP, `SWE`: Sweets.

Cluster	K_1	K_2	K_3	K_4	K_{All}
Size	259.6 ± 8.3	188.1 ± 14.9	270.3 ± 9.0	233.0 ± 4.0	1000.0 ± 0.0
Non Food	22.5 ± 1.4%	23.9 ± 3.4%	44.1 ± 1.1%	31.7 ± 0.8%	31.7 ± 0.3%
Non Perishable A	32.8 ± 2.0%	15.7 ± 1.5%	64.0 ± 1.6%	42.2 ± 0.8%	43.6 ± 0.3%
Non Perishable B	0.1 ± 0.1%	0.2 ± 0.2%	74.5 ± 5.5%	99.6 ± 0.5%	45.0 ± 0.3%
Non Perishable C	6.5 ± 0.4%	5.8 ± 0.8%	15.0 ± 0.7%	10.0 ± 0.4%	9.8 ± 0.2%
Non Perishable D	35.7 ± 2.6%	66.4 ± 9.7%	53.5 ± 1.2%	40.2 ± 0.9%	44.6 ± 0.3%
Perishable A	70.8 ± 2.8%	28.7 ± 2.6%	71.5 ± 1.8%	68.7 ± 0.7%	67.0 ± 0.3%
Perishable B	39.1 ± 5.2%	41.1 ± 10.1%	52.5 ± 3.1%	41.5 ± 0.8%	43.2 ± 0.3%
category 1	`FnVP`	`SOFNP`	`SNA`	`SWE`	`FnVP`
	39.2 ± 1.3%	38.1 ± 5.1%	72.9 ± 5.4%	99.3 ± 0.5%	41.3 ± 0.3%
category 2	`MnFP`	`AnT`	`FnVP`	`FnVP`	`SWE`
	27.9 ± 1.0%	34.0 ± 5.5%	50.0 ± 1.7%	42.9 ± 0.7%	33.2 ± 0.3%
category 3	`SOFNP`	`PREP`	`MnFP`	`MnFP`	`MnFP`
	21.6 ± 1.5%	29.5 ± 7.9%	41.9 ± 1.6%	33.7 ± 0.7%	32.0 ± 0.3%

unfortunately less obvious, as both have characteristics regarding the mindsets of `ON THE GO` and `PLANNED`. In contrast to the other two clusters; K_3 and K_4 have a relative high inclusion on most super-categories. It further makes sense that K_3 and K_4 are separated from the other two because of their high inclusion for Non-Perishable B. They get separated from each-other because of the focus on the underlying category (snacks for K_3 and sweets for K_4, which together make up the Non Perishable B super-category). Although not shown due to space restrictions, $K_3's$ sweets at 34.4% and $K_4's$ snacks at 7.2% are much lower, making up the seventh and twenty-first largest category respectively. One way to explain this difference is that K_3 is more of an `ON THE GO` mindset; where the consumer gets some snacks for the evening. This is further supported by the high value of Non Perishable D; the drinks that go with these snacks. At the same time, $K_3's$ inclusion is higher on almost every super-category, which would be more fitting for a `PLANNED` mindset. Future research should focus on including quantities and prices of products, which is expected to help in distinguishing between K_3 and K_4.

3.4 Competitor Analysis

We conclude our experimental analysis with a comparison of our approach to the more naive approach of setting equal weights for each super-category. We do this comparison using the same internal evaluation measure of Equation 6. We apply several clustering algorithms using our distance metric, but setting all weights equal to $\frac{1}{8}$ (i.e. without using our two-problem framework). We use the algorithms as implemented in Scikit-Learn [18], by setting the number of clusters

index	Φ
Our framework	-7.917 ± 0.002
Voronoi [21]	-7.720 ± 0.003
Spectral [16]	-7.735 ± 0.003
HAC - complete [20]	-7.698 ± 0.004
HAC - average [20]	-7.698 ± 0.005
HAC - single [20]	-7.605 ± 0.000

Table 7. Comparison of our approach to using various clustering algorithms and our distance metric with uniform weight.

to 4 and by using default values for the other parameters. We compute Φ for each $\mathcal{D}_i$ as used in the previous experiments; and report the mean and the 95% confidence interval for each approach.

The internal evaluation measure (which needs to be minimized) clearly indicates that our approach with learned weights performs better than the alternatives that do not learn weights; there is also no overlap in the confidence intervals. This shows the added benefit of learning the weights in improving the found clustering in terms of the internal evaluation measure.

4 Related Work

In this section we present some related work and how our approach fits into the literature.

Receipt distance metrics have been considered in different publications. Most metrics are based on either specific products of a retailer, or a large product hierarchy. Both these artefacts make the distance metrics more specific towards a single retailer.

In [15], the authors discuss several metric types on transaction data. They first consider affinity items; where two products that can for example substitute each-other (like two coke cans of different brands) are considered affinity products. Receipt distance is roughly based on the number of affinity products between them. In a sense; our method uses a similar concept taken to the extreme: we consider products in the same category to be equivalent. The authors then consider product distance, where the distance between two receipts is based on the closest or furthest two products, amongst others. A disadvantage is that it requires a distance between products, which is not suitable to our global target of developing retailer independent analytics.

The ROCK clustering algorithm for categorical attributes [9] uses transaction data as main use case. The clustering does not directly depend on the distance between datapoints, but on the number of common neighbour points. Contrary to our (super)-categories, ROCK does not assume relations between products, but uses the frequency of individual products. The disadvantage of the latter is that this makes inter-retailer comparisons infeasible. The clustering of categorical

features proposed in [25] uses a similar approach. Their target is to have transactions that contain certain products frequently bought together to be in the same cluster. This approach does not focus on finding a distance metric but more on finding the clusters. Other work that computes clusters through common items can be found in [1].

The product hierarchy we use in our approach is balanced: all leafs of the tree are at the same depth. In [26], the authors do not restrict this and develop an approach for dealing with unbalanced product trees. The advantage of this is that it allows retailers focussed towards one category of products to have a deeper hierarchy on that category, while having a more shallow hierarchy for other categories. Our approach implicitly solves this by considering a generic hierarchy, which would effectively transform an unbalanced product tree in to the hierarchy such as presented in Table 4.

The work presented in [7], using earlier work in [4–6], has a similar approach to ours; in the sense that a set of weights over a hierarchy is optimized while clustering the transactional data. Fundamentally different however, is that the approach uses the full product hierarchy of a retailer, making cross-retailer and cross-country analysis more difficult.

5 Conclusion and Future Work

In this paper we presented a novel method for determining the distance between consumer transactions, guided by consumer behaviour using k-medoid clustering. The framework as-is is able to self-optimize and explain consumer behaviour, despite being based on a high level of abstraction of consumer transaction data. We showed that our approach not only finds sensible and separable clusters of consumer behaviour, but also provide reasonable stability in doing so.

Nonetheless, extensions to the algorithm are considered for future work. There is potential to improve on the final clusters; but also on more accurately matching consumer behaviour types. First and foremost is using quantities instead of inclusion; this gives a more descriptive distance metric at the cost of increased computation time. We have explicitly set $k = 4$. While this makes sense based on the original mindset study, it is interesting to see what happens for higher values of k, seeing if new or 'sub-mindsets' are found. Another way to approach this by applying density based clustering methods [3, 12] instead. In the clustering step, we traded quality for efficiency by deploying Voronoi-clustering over PAM clustering, another optimization focus of our approach lies there. In the inter-super-category distance of Equation 4 we treated every super-category with the same weight; i.e. both receipts having *Non-Food* products is treated the same as both receipts having *Non-Perishable* products. Adding weights within this distance, with or without using quantities instead of inclusion, is another next step in our research. Finally, it will be interesting to see if we can find the same quality of results regardless of the retailer and respective product hierarchy.

Acknowledgements

The authors kindly thank BrandLoyalty's domain experts, in particular Anna Witteman, Hanneke van Keep, Lenneke van der Meijden, and Steven van den Boomen, for their contributions to the research presented in this paper.

References

1. Aggarwal, C., Procopiuc, C., Yu, P.: Finding localized associations in market basket data. IEEE Transactions on KDE **14**(1), 51–62 (2002).
2. Arthur, D., Vassilvitskii, S.: K-Means++: The Advantages of Careful Seeding. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1027–1035. SODA '07, Society for Industrial and Applied Mathematics, USA (2007).
3. Campello, R., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 7819 LNAI, pp. 160–172. Springer Verlag (2014).
4. Chen, X., Fang, Y., Yang, M., Nie, F., Zhao, Z., Huang, J.Z.: PurTreeClust: A Clustering Algorithm for Customer Segmentation from Massive Customer Transaction Data. IEEE Transactions on Knowledge and Data Engineering **30**(3), 559–572 (mar 2018).
5. Chen, X., Huang, J.Z., Luo, J.: PurTreeClust: A purchase tree clustering algorithm for large-scale customer transaction data. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE). pp. 661–672 (may 2016).
6. Chen, X., Peng, S., Huang, J.Z., Nie, F., Ming, Y.: Local PurTree Spectral Clustering for Massive Customer Transaction Data. IEEE Intelligent Systems **32**(2), 37–44 (mar 2017).
7. Chen, X., Sun, W., Wang, B., Li, Z., Wang, X., Ye, Y.: Spectral Clustering of Customer Transaction Data With a Two-Level Subspace Weighting Method. IEEE Transactions on Cybernetics **49**(9), 3230–3241 (sep 2019).
8. Gower, J.C.: A General Coefficient of Similarity and Some of Its Properties. Biometrics **27**(4), 857–871 (1971)
9. Guha, S., Rastogi, R., Shim, K.: Rock: a robust clustering algorithm for categorical attributes. Information Systems (2000).
10. Hamerly, G., Elkan, C.: Learning the k in K-Means. In: Proceedings of the 16th International Conference on Neural Information Processing Systems. pp. 281–288. NIPS'03, MIT Press, Cambridge, MA, USA (2003),
11. Hassani, M., Seidl, T.: Using internal evaluation measures to validate the quality of diverse stream clustering algorithms. Vietnam Journal of Computer Science **4**(3), 171–183 (Aug 2017).
12. Hassani, M., Spaus, P., Seidl, T.: Adaptive multiple-resolution stream clustering. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 8556 LNAI, pp. 134–148. Springer Verlag (2014).
13. Jain, A.K.: Data clustering: 50 years beyond K-means. Pattern Recognition Letters **31**(8), 651–666 (jun 2010).
14. Lam, S.K., Pitrou, A., Seibert, S.: Numba. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15. pp. 1–6. LLVM '15, ACM Press, New York, New York, USA (2015).

15. Lu, K., Furukawa, T.: Similarity of Transactions for Customer Segmentation. In: Quirchmayr, G., Basl, J., You, I., Xu, L., Weippl, E. (eds.) Multidisciplinary Research and Practice for Information Systems. pp. 347–359. Springer Berlin Heidelberg, Berlin, Heidelberg (2012).
16. Ng, A.Y., Jordan, M.I., Weiss, Y.: On Spectral Clustering: Analysis and an Algorithm. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. pp. 849–856. NIPS'01, MIT Press, Cambridge, MA, USA (2001),
17. Park, H.S., Jun, C.H.: A simple and fast algorithm for K-medoids clustering. Expert Systems with Applications **36**(2 PART 2), 3336–3341 (mar 2009).
18. Pedregosa, F., et al.: Scikit-learn: Machine Learning in {P}ython. Journal of Machine Learning Research **12**, 2825–2830 (2011)
19. Pelleg, D., Moore, A.W.: X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters. In: Proceedings of the Seventeenth International Conference on Machine Learning. pp. 727–734. ICML '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000),
20. Rokach, L., Maimon, O.: Clustering Methods, pp. 321–352. Springer US, Boston, MA (2005).
21. Schubert, E., Rousseeuw, P.J.: Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 11807 LNCS, pp. 171–187. Springer (oct 2019).
22. Storn, R., Price, K.: Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization **11**(4), 341–359 (1997).
23. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., USA (2005)
24. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods **17**, 261–272 (feb 2020).
25. Wang, K., Xu, C., Liu, B.: Clustering Transactions Using Large Items. In: Proceedings of the Eighth International Conference on Information and Knowledge Management. pp. 483–490. CIKM '99, ACM, New York, NY, USA (1999).
26. Wang, M., Hsu, P., Lin, K.C., Chen, S.: Clustering Transactions with an Unbalanced Hierarchical Product Structure. In: Song, I.Y., Eder, J., Nguyen, T.M. (eds.) Data Warehousing and Knowledge Discovery. Lecture Notes in Computer Science, vol. 4654, pp. 251–261. Springer Berlin Heidelberg, Berlin, Heidelberg (2007).