Marchal, Samuel; Szyller, Sebastian

Detecting organized eCommerce fraud using scalable categorical clustering

Published in:
2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA

DOI:
10.1145/3359789.3359810

Published: 01/01/2019

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Marchal, S., & Szyller, S. (2019). Detecting organized eCommerce fraud using scalable categorical clustering. In 2019 Annual Computer Security Applications Conference (ACSAC '19), December 9–13, 2019, San Juan, PR, USA (pp. 215–228). ACM. https://doi.org/10.1145/3359789.3359810

Detecting organized eCommerce fraud using scalable categorical clustering

Samuel Marchal
Aalto University
samuel.marchal@aalto.fi

Sebastian Szyller
Aalto University
sebastian.szyller@aalto.fi

ABSTRACT

Online retail, eCommerce, frequently falls victim to fraud conducted by malicious customers (fraudsters) who obtain goods or services through deception. Fraud coordinated by groups of professional fraudsters that place several fraudulent orders to maximize their gain is referred to as *organized fraud*. Existing approaches to fraud detection typically analyze orders in isolation and they are not effective at identifying groups of fraudulent orders linked to organized fraud. These also wrongly identify many legitimate orders as fraud, which hinders their usage for automated fraud cancellation. We introduce a novel solution to detect organized fraud by analyzing orders in bulk. Our approach is based on clustering and aims to group together fraudulent orders placed by the same group of fraudsters. It selectively uses two existing techniques, *agglomerative clustering* and *sampling* to *recursively* group orders into small clusters in a reasonable amount of time. We assess our clustering technique on real-world orders placed on the Zalando website, the largest online apparel retailer in Europe[1]. Our clustering processes 100,000s of orders in a few hours and groups 35-45% of fraudulent orders together. We propose a simple technique built on top of our clustering that detects 26.2% of fraud while raising false alarms for only 0.1% of legitimate orders.

KEYWORDS

online fraud; fraud detection; eCommerce; categorical clustering

1 INTRODUCTION

Online retail, also known as eCommerce, represents an important share of the retail business. About 17.5% of all sales made in the United States consists of eCommerce transactions, which accounts for several trillions of dollars every year [39]. The expansion of online retail is driven by its two main characteristics: 24/7 accessibility and scalability to a potentially unlimited number of customers. However, these features also increase the exposure to *frauds* in which malicious customers, *fraudsters*, obtain physical goods or services through deception. It is estimated that 3 to 5% of online orders constitute fraud, which accounts for over $50B in value every year [41]. Fraud represents a direct monetary loss that can significantly reduce the business valuation of online retailers [51] and it must be mitigated.

Cancellation of fraudulent orders in a timely manner prevents this monetary damage. To be effective, fraud cancellation requires reliable means of detection. Cancelling legitimate orders decreases customer loyalty, degrades brand image/reputation and causes a shortfall in revenue estimated to over $100B every year [12]. Because of this reliability requirement, current approaches to canceling fraud rely on *screening* which is a manual process done by human analysts [35]. Thus, screening is a costly process applied to a limited number of orders and which can prevent only a limited amount of fraud. Screening can be facilitated using automated analysis techniques that produce additional fraud indicators [31]. Nevertheless, these techniques are not accurate enough to provide standalone and automated fraud cancellation [35].

The type of fraud that has been rising is *organized fraud* [27]. 95 professional fraudsters performing organized fraud were arrested in 2018 for committing fraud exceeding €8M in value [14]. In organized fraud, a small group of fraudsters coordinate *fraud campaigns* against a chosen online retailer. Fraud campaigns span a limited period of time during which several fraudulent orders are placed for goods delivered in a limited geographical area. The online retailer, Zalando, lost €18.5M to organized fraud in one quarter of 2015 [51]. Since then, Zalando invested systematically into their fraud detection systems, which reduced fraud to very low numbers. Nevertheless, throughout the market, many current automated techniques for fraud detection analyze orders in isolation [8, 10, 17, 27, 43]. However, detecting organized fraud profits from a global view of all orders placed in a given period of time.

Goal and contributions. We want to design a solution to detect organized eCommerce fraud. We propose analyzing orders in bulk rather than in isolation, to identify similarities among fraudulent orders that belong to the same fraud campaign. Similar fraudulent orders can be grouped together by applying *clustering* on relevant attributes. Orders in the same fraud campaign have common characteristics highlighted by identical *categorical attributes*, e.g., delivery address, customer name, payment method, etc. Consequently, we propose applying unsupervised *categorical clustering* on these attributes to identify organized eCommerce fraud.

We introduce a novel approach for hierarchical categorical clustering: *recursive agglomerative clustering*. This approach is specifically designed to group fraudulent orders placed on online retail stores. It combines the benefits of (1) *agglomerative clustering* to generate small clusters each potentially representing a fraud campaign and (2) *sampling* to process a large number of orders in a reasonable amount of time. Clusters obtained using our clustering approach have two applications: **A1** prioritizing orders that must be analyzed through screening and **A2** automatically canceling frauds by deciding that all orders in a cluster are fraudulent if at least one order in the cluster is fraudulent (e.g., older known fraud).

We assess the real-world effectiveness of our approach using 6M orders placed on the Zalando website [49], the largest online apparel retailer in Europe. We claim the following contributions:

[1]Disclaimer: The views and opinions expressed in this article are those of the authors and do not necessarily reflect the official position of Zalando Payments GmbH.

- a novel clustering technique for categorical data (Sect. 3.2). It is a recursive clustering approach combining agglomerative clustering and sampling to generate a large number of clusters from medium-size datasets (100,000s of samples).
- two strategies for weighting categorical attributes (Sect. 4), which facilitate the selection of optimal hyperparameters for categorical clustering (Sect. 6.3).
- the evaluation of our clustering technique showing its effectiveness at grouping fraudulent orders (Sect. 7). It generates clusters mixing a small number of fraudulent and legitimate orders (0.8%) while grouping a large portion of fraudulent orders (42.1%) together. Its computation time is much lower than existing clustering techniques and is able to process, e.g., 15,000 orders in 3 minutes.
- the demonstration that generated clusters can be used to automatically detect 26.2% of real-world fraud perpetrated against Zalando, while causing only 0.1% false alarms for legitimate orders incorrectly identified as fraud (Sect. 8).

2 DETECTING ECOMMERCE FRAUD

We focus on the detection of fraudulent orders committed against online retailers, which we simply name *frauds* from now on. In this paper, we tackle the specific use case of detecting fraud perpetrated against Zalando [49]. Zalando is an online apparel retailer operating in 17 markets, having 28 million active customers and generating over €5B in revenue yearly [52]. We believe this use case is representative of many large online retailers. We focus on detecting fraud where fraudsters obtain physical goods through delivery with no intent of paying for them. The order can either be in payment default or paid with illegally acquired means of payment. In both cases, the retailer suffers monetary losses.

2.1 Fraud detection and cancellation

A typical[2] fraud detection pipeline [11, 13, 35] is depicted in Fig. 1. An order is represented by a set of numerical features and categorical attributes. This information is automatically validated to confirm the order which serves as a preliminary step for the fraud detection process. Features and attributes representing an order are fed to several *scoring functions* that automatically produce fraud indicators. These functions can use additional background information (e.g., from customer history) and they typically rely on human defined heuristics and supervised machine learning (ML) models [8, 37, 43]. Fraud indicators and raw order information are provided to a screening process that decides if the order is legitimate and should proceed or if it is a fraud and it should be canceled. Cancellations are usually performed based on a combination of machine learning systems and human expert knowledge.

Fraud detection is a time constrained process that must happen after an order is placed and before it is processed for shipping. This typically gives only a few hours to detect fraud and only a small fraction of orders can be inspected by human experts. On the other hand, many automated scores are computed on each order

[2]This pipeline is chosen for the sake of generalizability. The particular fraud detection setup at Zalando is not taken into account in this paper and it does not perfectly match this typical pipeline.

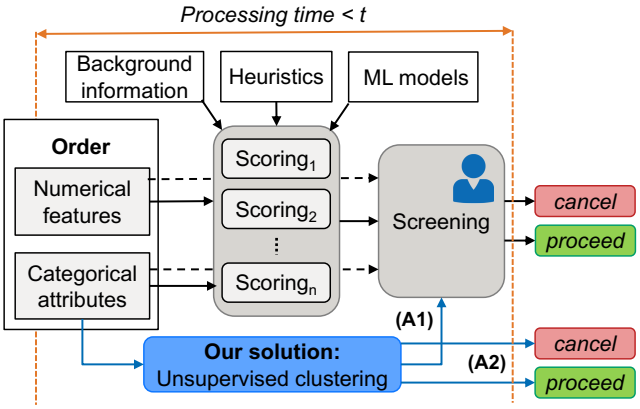

Figure 1: Fraud detection pipeline. Final cancellation is decided by human analysts during screening. Our solution supports screening (A1) and automatically cancel frauds (A2).

independently, which can decrease their efficiency at detecting organized fraud.

2.2 Preventing organized fraud

Fraud can be either isolated events occasionally performed by individuals or organized by criminal groups of professional fraudsters [27]. Organized fraud relies on coordinated events, *fraud campaigns*, that target a specific online retailer. During a fraud campaign, several orders are placed over a limited period of time (e.g., one month), by a small group of fraudsters having several electronic identities each. Most orders are fraudulent and all orders are delivered in a restricted geographical area (e.g., the same city) where the criminal group operates. Also, fraud campaign typically uses payment methods that are known to be vulnerable to fraud [51]. We propose to prevent organized fraud by identifying fraud campaigns.

Following that fraud belonging to a same campaign has similar characteristics, we propose to group similar orders together in order to identify fraud campaigns. In this study, we restrict ourselves to certain categorical attributes, i.e., delivery address, customer name, payment method, etc. Numerical features have already been extensively used for fraud detection [9, 10, 48] and we want to investigate the capabilities of categorical attributes in their own right. Since we do not have a priori knowledge about orders that belong to a fraud campaign, we propose to take an unsupervised clustering approach to group similar orders and apply it to categorical attributes of orders. Ideally it would generate one cluster per fraud campaign, containing all frauds of this campaign but no legitimate order. Also, most legitimate orders should have low similarity between each other and thus, be less likely to be grouped into clusters.

Figure 1 depicts the deployment of our clustering approach in the fraud detection pipeline. It has two applications.

A1 Prioritizing screening. Clustered orders can be screened with high priority to detect a large number of frauds with a minimal effort. We expect frauds to be clustered at a significantly higher rate than legitimate orders. Comparing orders in the same cluster provides human analysts with new information that may facilitate the cancellation.

A2 **Automated fraud cancellation.** Fraud detection can be applied to clusters of orders rather than to individual orders. Several orders provide aggregated information that may depict fraudulent behavior more reliably than isolated orders. An automated process can decide if the whole cluster is fraudulent and cancel the orders.

While a significant share of frauds is organized and may be associated with a fraud campaign, frauds can also be isolated events. We focus on detecting organized fraud only and our approach is not designed to detect isolated fraud cases. Our approach is complementary to the extensive prior work addressing the detection of fraud in isolation [2, 9, 17, 29].

2.3 Attributes representing orders

In the following, we represent each order by 37 categorical attributes, which have discrete values with no intrinsic ordering. These attributes were selected for the sake of generalizability. They belong to 5 categories of information that is generally provided by a customer placing an order on any online retailer.

- Customer A_{cust} (9 attributes): related to the electronic identity of the customer, e.g., email address, IP address, etc.
- Delivery A_{del} (3 attributes): related to the means used for order delivery, e.g, pickup point, delivery type, etc.
- Shipping A_{ship} (7 attributes): related to identity and location (address) of the person receiving the order.
- Payment A_{pay} (11 attributes): related to payment method, e.g., bank transfer, credit card suffix, etc.
- Billing A_{bill} (7 attributes): related to identity and location (address) of the person paying the order.

Many of our attributes contain Personally Identifiable Information (PII) which were anonymized prior to perform any data analysis. The clustering approach we introduce and the experimental results we obtain use these anonymized attribute values.

2.4 Challenges in clustering fraud campaigns

Clustering orders that belong to fraud campaigns requires to address several challenges related to (a) categorical clustering, (b) fraud detection and (c) the application domain of online retail.

C1 **Imbalanced attribute cardinality.** Categorical attributes representing orders take a different number of values (cardinality), from two values to millions. High cardinality prevents the numerical encoding of attributes. Imbalance makes it difficult to quantify the similarity between two orders.

C2 **Imbalanced classes.** The ratio of fraudulent to legitimate orders is highly imbalanced, typically around 1/50 [41]. Probability of clustering legitimate orders is much higher than that of clustering frauds, which is undesirable.

C3 **No ground truth for fraud campaign.** There is no information which fraud corresponds to which fraud campaign. Only ground truth for individual orders is available.

C4 **Scale of the data.** Large online retailers receive 100,000s of orders per day. Zalando receives 300,000 orders on average every day [52]. Most existing categorical clustering methods [15, 20, 50] have a high complexity and they cannot process data of such a scale in a reasonable amount of time.

2.5 Requirements

We define the following requirements for a clustering approach to detect fraud campaigns:

R1 **Generate small clusters**. There are many more legitimate orders than frauds (**C2**). Also, a fraud campaign typically contains a low number (e.g., 10s-100s) of orders. In order to group frauds, clustering must generate a large number of small clusters, each potentially corresponding to a single fraud campaign.

R2 **Minimize cluster impurity**. The cluster impurity must be low. Generated clusters must be composed either only of legitimate orders or only of frauds.

R3 **Maximize clustered fraud**. Frauds isolated in singletons (clusters with one component) are not linked to any fraud campaign and they cannot be detected by our method. We must maximize the rate of detected fraud.

R4 **Minimize execution time**. Online retailers receive 100,000s of orders per day. Our approach must be able to process such amount of data (**C4**) in a reasonable amount of time allowing for cancellation (e.g., a few hours).

Regarding **R1**, we do not have ground truth for fraud campaigns (**C3**) and we cannot evaluate the "*goodness*" of our clusters with respect to grouping frauds from the same campaign. We ensure clustering *goodness* by requiring a minimum similarity between elements belonging to the same cluster, which is typical in clustering [30]. This guarantee is provided by enforcing a maximum distance between elements that compose the same cluster. Keeping this distance below a threshold is our criterion for cluster goodness.

3 RECURSIVE AGGLOMERATIVE CLUSTERING

We introduce *Recursive Agglomerative Clustering* (RecAgglo) a novel approach for categorical clustering. It combines the benefits of two existing techniques [30]: *agglomerative clustering*, which is able to generate small clusters (**R1**) and *sampling*, which reduces the time complexity of clustering methods (**R4**). These two techniques are selectively applied to recursively divide a large set of samples into small clusters which eventually meet our goodness criterion. The code for the RecAgglo algorithm is publicly available [33].

3.1 Agglomerative clustering and sampling

Agglomerative clustering is a bottom up hierarchical clustering approach. Each element is initially placed into a singleton cluster. Pairs of clusters having the smallest distance to each other are then sequentially merged into larger clusters until all elements are in a single cluster. The distance between two clusters is defined using a *linkage* method. For instance, *single linkage* uses the minimum distance between any two points in each cluster. The algorithm produces a dendrogram which represents consecutive merges. Using the dendrogram, a desired clustering (set of clusters) can be chosen using different criteria, e.g., *'distance'*: the maximum distance d_{max} between elements in a cluster, *'maxclust'*: the maximum number of clusters c_{max} to generate. In contrast to many clustering techniques [5, 22], agglomerative clustering does not generate a predefined number of clusters. It generates clusters by grouping the

most similar singletons first and it can create many small clusters which meet our goodness criterion as defined by the *'distance'* d_{max}. Elements that cannot be assigned to any cluster while meeting this criterion remain isolated in singletons. Agglomerative clustering requires computing pairwise distances between elements ($O(n^2)$ complexity) and does not scale to large datasets. The AggloClust algorithm is presented in App. A.1. It takes a set c of elements to cluster and a distance d_{max} as inputs.

The sampling algorithm is applied on top of existing clustering techniques. It selects a random sample of reference elements from a set of n elements. These reference elements are clustered and the remaining ones (not sampled) are assigned to the initially formed clusters. This reduces the number of distance computations between elements from $n \times n$ to the sample size $\times n$. If the sample size is in the order of $O(log(n))$, the complexity of the base clustering algorithm is reduced by the same factor.

We use sampling to reduce the complexity of agglomerative clustering to $O(n \times log(n))$ by using a sample size in the order of $O(log(n))$. Our algorithm for agglomerative clustering with sampling SampleClust is detailed in App. A.2. It takes 3 inputs: a set c of elements to cluster, ρ_s a multiplying factor to $\sqrt{|c|}$ defining the sample size and ρ_{mc} the maximum number of clusters to generate (using the *'maxclust'* criterion for cluster generation). ρ_s and ρ_{mc} are parameters to be defined according to the desired computation time. Sampling deprives agglomerative clustering of its ability to generate many small clusters since the number of clusters is bounded by the sample size. Also, clusters generated using sampling do not meet any goodness criterion defined by the maximum *'distance'* d_{max}.

3.2 Our RecAgglo algorithm

We introduce a scalable approach to generating clusters that meet our goodness criterion, i.e., the distance between elements in the same cluster is lower than d_{max}. Our solution ❶ recursively divides large clusters into smaller ones using SampleClust. When clusters are small enough, ❷ it runs AggloClust to generate a clustering in which each cluster meets the *'distance'* criterion d_{max}. All resulting clusters are recursively aggregated to form the final clustering C_{res} composed of clusters that meet our goodness criterion.

Our approach RecAgglo is defined in Algorithm 1. It takes as inputs an initial clustering C and a set of parameters: δ_a (for RecAgglo), d_{max} (for AggloClust), ρ_s and ρ_{mc} (for SampleClust). δ_a is a parameter defined according to computation time restrictions. RecAgglo loops over clusters $c \in C$ to split them into smaller clusters.

If the size of c is larger than a threshold δ_a, c is split using SampleClust. We use the *'maxclust'* criteria to generate clusters small enough to be eventually processed using AggloClust. The resulting clustering C_s does not meet our goodness criterion yet and it is re-processed using RecAgglo. We observed that SampleClust may not be able to split an input set c given a specific 'maxclust' factor ρ_{mc}. We address this in two ways. Firstly, we re-try SampleClust with a lower value $\rho_{mc} = 1.01$ providing the ability to generate more clusters. Secondly, we fall back to plain agglomerative clustering given that the cluster size is still reasonably low ($4 \times \delta_a$). Both these measures were determined empirically. Alternatively, ρ_{mc} could be progressively decreased or the multiplying factor of δ_a can be changed. If both measures fail to split c in clusters, elements in c are added to the set *remain* for further processing. Alternatively, we

Algorithm 1 Recursive agglomerative clustering

```
Let C = c_1, ..., c_n denote a clustering of |C| = n clusters,
c = v_1, ..., v_m, a cluster of |c| = m elements v,
δ_a, the threshold for using AggloClust,
d_max, the maximum distance for cluster fusion,
ρ_s, the multiplying factor for the sample size,
ρ_mc, the dividing factor for maxclust number.

function RecAgglo(C, δ_a, d_max, ρ_s, ρ_mc)
    C_res ← ∅
    remain ← ∅
    max_s ← 4 × δ_a

    for c : c ∈ C do                          ▷ Loop to split existing clusters
        if |c| > δ_a then                     ▷ Cluster sampling
            C_s ← SampleClust(c, ρ_s, ρ_mc)
            if |C_s| > 1 then                 ▷ Recursive clustering
                C_loop ← RecAgglo(C_s, δ_a, d_max, ρ_s, ρ_mc)
            else if ρ_mc > 1.01 then          ▷ Recursive clustering alt.
                ρ_mc ← 1.01                   ▷ Set higher maxclust
                C_loop ← RecAgglo(C_s, δ_a, d_max, ρ_s, ρ_mc)
            else if |c| < max_s then          ▷ Fall back agglomerative
                C_loop ← AggloClust(c, d_max)
            else                              ▷ No split possible / to re-cluster
                remain ← remain ∪ c
            end if
        else if |c| > 1 then                  ▷ Agglomerative clustering
            C_loop ← AggloClust(c, d_max)
        else                                  ▷ Elements to re-cluster
            remain ← remain ∪ c
        end if
        C_res ← C_res ∪ C_loop                ▷ Add new clusters to result
    end for

    # Clustering non-clustered elements (remain)
    if |remain| > δ_a then                    ▷ Cluster sampling
        C_sample ← SampleClust(remain, ρ_s, ρ_mc)
        if |C_sample| > 1 then                ▷ Recursive clustering
            C_end ← RecAgglo(C_sample, δ_a, d_max, ρ_s, ρ_mc
        else                                  ▷ Elements are singletons
            C_end ← {remain}
        end if
    else if |remain| > 1 then                 ▷ Agglomerative clustering
        C_end ← AggloClust(remain, d_max)
    else                                      ▷ Elements are singletons
        C_end ← {remain}
    end if
    C_res ← C_res ∪ C_end
    return C_res
end function
```

obtain a resulting clustering C_{loop} from RecAgglo or AggloClust that meets our goodness criterion.

If the size of c is lower than δ_a but larger than 1, c is split using AggloClust with parameter d_{max}. If c is a singleton, it is added to the *remain* set for later processing. During each iteration, we add the new clustering C_{loop} to the final clustering C_{res} or we complement the *remain* set of elements for reprocessing.

The *remain* set contains clustered elements resulting from SampleClust and singletons - obtained due to lack of sufficiently similar elements in the drawn sample. Thus, we try to re-cluster these remaining elements, following the same steps as previously. We use SampleClust (if $|remain| > \delta_a$), AggloClust (if $\delta_a \geq |remain| > 1$) or keep a singleton (if $|remain| = 1$). If SampleClust is successful at splitting *remain*, we recursively run RecAgglo on the resulting clustering. However, in contrast to the previous process, we do not apply alternative measures if SampleClust fails and just keep all elements as singletons.

The resulting clustering C_{end} is added to C_{res} which is our final clustering where all clusters meet our goodness criterion. It is worth noting that many of these clusters may be singletons.

3.3 RecAgglo properties

Achieving cluster goodness: RecAgglo uses agglomerative clustering to generate the final clustering C_{res}. Consequently, any cluster in C_{res} of two or more elements meets our goodness criterion defined by the maximum distance d_{max}. These clusters are smaller than δ_a and meet **R1** for a sensible choice of δ_a.

Computational complexity: The complexity of RecAgglo depends on its recursive nature and SampleClust complexity. AggloClust runs on sets of size with the static upper bound δ_a. Its running time is bounded by a constant. The maximum complexity of SampleClust during the initial run is $O(n \times log(n))$ and it decreases during subsequent recursions. In the worst-case scenario, we require at most n recursions to obtain the final clustering. This makes the worst-case complexity of $O((n \times log(n))^n)$ for RecAgglo. This theoretical complexity is completely untractable and RecAgglo cannot scale to large datasets in theory. However, we show in Sect. 7.3 that its actual complexity is sub-quadratic when clustering sets containing up to 100,000s orders. In this setting, RecAgglo is faster than most categorical clustering algorithms.

Non-optimal solution: RecAgglo is non-deterministic and it does not produce a globally optimal clustering. This is due to the stochastic nature of the sampling process used SampleClust. We show in Sect. 7.3 that clusters that we obtain during different runs are consistent. Also, their goodness is close to the one of clusters generated using plain agglomerative clustering, while improving on the basic sampling method for clustering.

Hybrid clustering (using numerical features): Numerical features can be input to a clustering algorithm for continuous data (e.g., K-means, DBScan, etc.) in order to generate clusters in a standalone manner. The resulting clustering (cluster indexes) can be used as an additional categorical attribute that is input to RecAgglo in a cluster aggregation fashion [16].

4 ATTRIBUTE WEIGHTING STRATEGIES

Hamming distance and *Jaccard Index* are the most widely used metrics for computing the distance between two elements u and v represented using categorical attributes [30]. We use *Hamming* distance in our clustering algorithms since it is fast to compute. It counts the number of different attribute values between two elements:

$$Hamming(u, v) = \frac{1}{d} \sum_{i=1}^{d} w_i \times (u_i \neq v_i) \quad (1)$$

By default, *Hamming* like many other metrics gives the same weight to every attribute ($w_i = 1$). However, different attributes might not contribute equally to quantifying the similarity between u and v, or to produce "good" clusters. For instance, if attributes having high cardinality are matching, this may indicate a higher similarity than if attributes having low cardinality are matching. We propose two novel strategies for weighting attributes which capture these aspects and help addressing **C1**. The first strategy is based on feature cardinality while the second uses labels from known frauds and legitimate orders.

4.1 Cardinality driven attribute weight

We define a function to compute the weight w_i for an attribute a_i based on its cardinality. The cardinality $card_i$ is the total number of values attribute a_i can take. The rationale for weighting attributes based on cardinality is the following: the probability of two elements u and v having equal value for an attribute i is inversely proportional to the attribute cardinality $card_i$ for uniformly distributed attribute values. The goal of this weighting strategy is to give larger weights to attributes having high cardinality.

Cardinality of the attributes in our dataset is not bounded since values may be added as new orders are made, e.g., new customers signing up. Thus, we use the inverse normalized richness index [24] as the basis for weight computation: $R_i^{-1} = \frac{n_i}{card_i}$. n_i is the number of instances in a given set of size N for which a_i is not null . R_i^{-1} is a positive decreasing function of the attribute cardinality.

We use a sigmoid function ($\frac{x}{|1+x|}$) to scale R_i^{-1} to $[0, 1]$. We normalize its value over this range using the median value of R_i^{-1} computed over all 37 attributes: $median(R^{-1})$. Finally, we scale our weight to an intended range that controls the maximum difference between attribute weights. We chose the range $[1, 3]$ - we do not discard any attributes and a given attribute can have at most 3 times higher weight than any other. We compute cardinality driven weights as follows:

$$w_i^{\#} = 1 + 2 \times \left(1 - \frac{R_i^{-1}}{median(R_i^{-1}) + R_i^{-1}}\right), w_i^{\#} \in [1, 3] \quad (2)$$

4.2 Label driven attribute weight

We define the second function to compute weights using ground truth fraud labels of past orders. These weights are computed in order to satisfy two requirements for our method: **R3** maximizing clustered fraud and **R2** minimizing cluster impurity.

We start by clustering a set of orders using default attribute weights ($w_i = 1$) using *Hamming* distance. We obtain clusters of three types: (a) pure clusters c_f containing only frauds, (b) pure clusters c_l containing only legitimate orders and (c) mixed cluster c_m. c_m clusters violate **R2** and their number must be minimized. c_f clusters contribute to **R3** and their number must be maximized.

We aim to emphasize the importance of attributes that help generating c_f clusters and de-emphasize the importance of attributes that do not by scaling their weight accordingly. The higher the weight, the more important the attribute. We compute the contribution of an attribute a_i towards generating a cluster c using the Simpson index [45]. It is defined as $\lambda_i(c) = \sum_{j=1}^{card_i} p_j^2$, where p_j is the probability of encountering the attribute value v_j in c: the ratio of elements having v_j for a_i. High Simpson index indicates that a low number of different values v is present in the cluster. This means a_i has significantly contributed towards generating this cluster.

Using the Simpson index we define two metrics $Adv_{f/l}$ in Eq. (3) and $Adv_{p/m}(a_i)$ in Eq. (4). $Adv_{f/l}(a_i)$ measures the *advantage* of the attribute a_i in generating pure fraudulent clusters c_f rather than pure legitimate clusters c_l. $Adv_{p/m}(a_i)$ quantifies the *advantage* of the attribute a_i in generating pure clusters c_f and c_l instead of mixed clusters c_m. High $Adv_{f/l}$ helps achieving **R3** and a high $Adv_{p/m}$ helps achieving **R2**. The normalization term $norm_{adv}(a_i)$ ensures that $Adv_{f/l}(a_i) + Adv_{p/m}(a_i) \in [0, 2]$ which allows us to keep the final weight in the range $[1, 3]$

$$Adv_{f/l}(a_i) = \frac{\overline{\lambda_i(c_f)} - \overline{\lambda_i(c_l)}}{norm_{adv}(a_i)} \tag{3}$$

$$Adv_{p/m}(a_i) = \frac{\overline{\lambda_i(c_f)} + \overline{\lambda_i(c_l)} - 2 \times \overline{\lambda_i(c_m)}}{2 \times norm_{adv}(a_i)} \tag{4}$$

We compute our label driven weights using both these advantages as follows:

$$w_i^* = 1 + Adv_{f/l}(a_i) + Adv_{p/m}(a_i), w_i^* \in [1, 3] \tag{5}$$

5 PERFORMANCE METRICS AND DATASETS

We discussed that RecAgglo meets **R1** by design. We empirically evaluate the remaining requirements **R2** (minimize cluster impurity), **R3** (maximize clustered fraud) and **R4** (minimize execution time).

5.1 Performance metrics

We evaluate **R2** by computing the cluster *impurity* measure I, which is used to evaluate the quality of a clustering [16]. We give the label of the majority class to each cluster and all samples that do not belong to this class are counted as the impurity. For a clustering containing k clusters of sizes $s_1, \ldots, s_k$ and the sizes of the majority class in each clusters $m_1, \ldots, m_k$, the impurity index is defined as:

$$I = \frac{\sum_{i=1}^{k}(s_i - m_i)}{\sum_{i=1}^{k} s_i} = \frac{\sum_{i=1}^{k}(s_i - m_i)}{n} \tag{6}$$

We evaluate **R3** by calculating the clustered fraud rate (*CFR*) which is the ratio of clustered frauds to the total number of frauds. For the count of frauds in each cluster $f_1, \ldots, f_k$ and the total number of frauds F, *CFR* is defined as:

$$CFR = \frac{\sum_{i=1}^{k}(f_i)}{F} \tag{7}$$

We evaluate **R4** by measuring the computation time t of the clustering for the given dataset.

Our objectives are to minimize the impurity I and computation time t while maximizing the *CFR*.

5.2 Datasets

We use several datasets composed of real fraud and legitimate orders placed on the Zalando website in 2017 and 2018. Zalando receives on average 29 million orders per quarter [52]. Our ground truth fraud labels are obtained based on actual payment status of the order 12 weeks after it is placed. Orders without a label are considered legitimate.

The datasets presented in the following are sampled from the original order data. They differ in size and ratio of legitimate to fraudulent orders. We use them for different experiments that we describe as follows.

Small datasets with artificial distribution. We sample two small datasets TrainF-15K and TestF-15K that are used for selecting hyperparamters of agglomerative clustering (Sect. 6) and for comparing the performance of several categorical clustering techniques (Sect. 7.3) respectively. These sets are small enough for most categorical clustering techniques to run in a reasonable amount of time (<10 hours). Also, they contain enough frauds to generate many fraudulent clusters that we can use to compute sensible impurity I and *CFR* metrics. Frauds are artificially over-sampled (1 fraud / 2 legitimate) compared to a real-world distribution. Each dataset consists of 10 disjoint subsets, each composed of 10,000 legitimate orders and 5,000 frauds.

Large datasets with artificial distribution. We sample two larger datasets TrainG-30K and TrainG-100K that are used for selecting hyperparamters of RecAgglo (Sect. 7.1). These also have an artificial distribution where frauds are over-sampled compared to the real-world distribution. The imbalance is larger and more realistic in these datasets though (1 fraud / 5 legitimate and 1 fraud / 19 legitimate). TrainG-30K consists of 10 disjoint subsets, each composed of 25,000 legitimate orders and 5,000 frauds. TrainG-100K consists of 5 disjoint subsets, each composed of 95,000 legitimate and 5,000 fraud. The composition of datasets with artificial distribution is presented in detail in App. B.

Real-world datasets. Finally, we select real-world datasets that will be used to evaluate the actual effectiveness of RecAgglo at clustering fraud in Sect. 8. These datasets consist of all Zalando Fashion Store orders placed between April 1st and May 5th 2018 (35 days) in Germany (DE-real), Switzerland (CH-real), the Netherlands (NL-real), Belgium (BE-real) and France (FR-real). These datasets contain more than 6 million orders in total, with a realistic fraud/legitimate order ratio (well below 1% before fraud cancellation).

Each of these datasets is complemented with a background dataset containing only frauds placed between January 1st and March 31st 2018 (90 days). These datasets are respectively named DE-bg, CH-bg, NL-bg, FR-bg BE-bg.

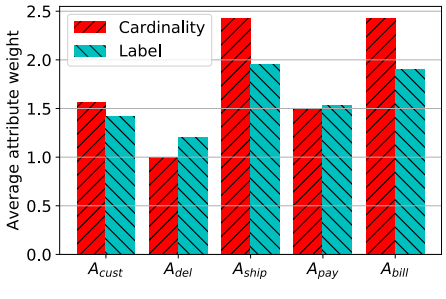

Figure 2: Average weights of five attribute categories using cardinality and label driven weights. Shipping and billing attributes are the most important in generating clusters.

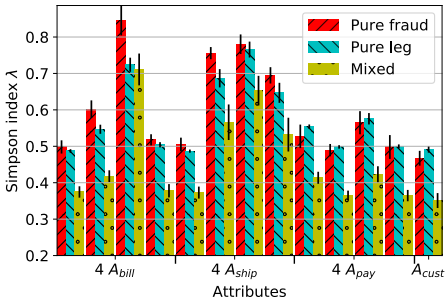

Figure 3: Simpson index λ for 13 attributes providing the best advantage in generating fraudulent and pure clusters. Billing, shipping and payment attributes provide the best advantage.

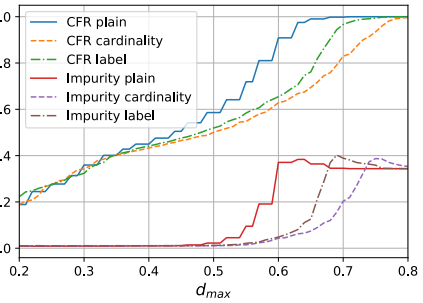

Figure 4: Increase in Impurity and *CFR* with d_{max} for three attribute weighting strategies. A fine-grained choice of Impurity/CFR tradeoff is possible using cardinality and label driven weights.

6 WEIGHTING STRATEGIES EVALUATION

Agglomerative clustering is the basis for RecAgglo. It uses three hyperparameters: a distance metric, a linkage method and the maximum distance for cluster fusion d_{max}. Recall that we selected *Hamming* as a distance metric because of its low computation cost. We selected the *single* linkage method based on evaluation described in App. C.1. We want to select the optimal weighting strategy and a distance d_{max} which minimize the impurity I and maximize the CFR. We compare the default weighting strategy ($w_i = 1$) to the cardinality $w_i^{\#}$ and label driven weights w_i^* we introduced in Sect. 4.

6.1 Weight computation

We select a random sample of 2M orders placed in France in 2017 to compute our cardinality driven weights using Eq. (2). In this subset, we obtain a minimum inverse normalized richness index $min(R_i^{-1}) = 1.606$ for one of the attributes in the A_{cust} category. It means that the same value repeats less than twice (on average) for over 2M samples. On the other hand, we obtained $max(R_i^{-1}) = 1,000,000$ for one of the attributes in A_{del} meaning it has only two possible values. These statistics highlight the imbalance in the cardinality of attributes representing orders (**C1**), which justifies cardinality based weighting strategy. We computed the value for $median(R_i^{-1}) = 149$ that we use to calculate the weights of all 37 attributes.

$$w\#_i = 1 + 2 \times (1 - \frac{R_i^{-1}}{149 + R_i^{-1}})$$

We start by clustering each set TrainF-15K-i using agglomerative clustering and default weights $w_i = 1$ to compute our label driven weights. We select the maximum distance for cluster fusion $d_{max} = 0.56$, which generates a clustering with impurity $I = 0.095$ and $CFR = 0.719$ (App. C.1). The majority of fraud is clustered (72%) and there is a significant number of mixed clusters as depicted by the high impurity (9.5%). We compute the Simpson index λ_i for each attribute a_i in each generated cluster. We aggregate these results to compute the mean $\overline{\lambda_i}$ for pure fraudulent clusters (c_f), pure legitimate clusters (c_f) and mixed clusters (c_m). Using these statistics we compute our advantage metrics (Eq. (3) and (4)) and by extension our final label driven weights.

6.2 Attribute importance

Figure 2 depicts the average cardinality and label driven weights of attributes in each category: A_{cust}, A_{del}, A_{pay}, A_{ship} and A_{bill}. Despite the different rationale and implementation for our two weighting strategies, we see they give similar high and low weights to the same attributes. Attributes in A_{ship} and A_{bill} have the largest weights according to both strategies. These attributes differ between customers (A_{ship}) and between orders (A_{bill}), which explains their high cardinality and their large cardinality driven weights.

Figure 3 depicts the averaged Simpson index for the 13 attributes providing the highest *advantage*. We observe that A_{bill} attributes give the best advantage for generating fraudulent rather than legitimate clusters (higher Simpson index in pure fraudulent clusters). A_{ship} attributes also provide a small advantage towards that goal, while A_{pay} and A_{cust} do not. Different values of the Simpson index depict the advantage of each attribute and are captured by our label driven features. It can be seen that A_{ship} and A_{bill} attributes have the highest weights (Fig. 2). On the other hand, all 13 attributes contribute to generating pure clusters (lower Simpson index for mixed clusters). A_{cust} and A_{del} contribute the least to our *advantages* and they have the lowest weight in Fig. 2.

High Simpson index values for A_{bill} and A_{ship} attributes indicate that fraudulent orders have more similar billing and shipping information than legitimate orders. On the other hand, there is no significant difference for A_{cust} and A_{pay} attributes. These results might indicate that fraudsters tend to use several user accounts and payment methods with similar billing and shipping information. Consequently, our generated clusters have characteristics that are typically associated with fraud campaigns as presented in Sect. 2.2.

6.3 Weighting strategies performance

We clustered the 10 TrainF-15K-i datasets using default attribute weights, cardinality and label driven weights. Each weighting strategy provides a similar I/CFR tradeoff that is detailed in App. C.2.

Nevertheless, label driven weights provide a slightly better *CFR* than other strategies for the same impurity value and we select it for the remaining experiments. A more interesting property of cardinality and label driven weights can be observed in Fig. 4. Both these strategies offer a smoother increase of impurity and *CFR* while varying d_{max}. In contrast, default weights have abrupt changes and long plateaus providing the same performance. In this setting it is difficult to select an optimal d_{max} that provides desired impurity and *CFR* values. Cardinality and label driven weights can be used to effectively fine-tune d_{max} in order to achieve desired performance characteristics. Using Fig. 4, we select $d_{max} = 0.5$ with label driven weights, which results in an average impurity $I = 0.012$ for a $CFR = 0.52$. With this value, impurity remains low (about 1%) while more than half of the frauds are clustered.

Hamming distance with label driven weights, single linkage and distance $d_{max} = 0.5$ are used in RecAgglo in all following experiments.

7 RECAGGLO PERFORMANCE EVALUATION

We evaluate the performance of RecAgglo in terms of impurity, *CFR* and computation time when clustering real online orders. We compare this performance to several state-of-the-art categorical clustering techniques to show RecAgglo is best suited for this task.

7.1 Hyperparameter setting

RecAgglo requires defining δ_a and SampleClust requires defining ρ_s and ρ_{mc} (cf. Sect. 3). We compute optimal hyperparameter values with the primary goal of minimizing computation time and the secondary goals of minimizing impurity I and maximizing *CFR*. We set $\delta_a = 1,000$ with computation time being the only consideration in mind. Agglomerative clustering takes 25s to process 1,000 samples. Consequently, this is an upper bound for the computation time of AggloClust in RecAgglo. The upper bound for the fall back agglomerative clustering is given for $4,000$ $(4 \times \delta_a)$ elements to cluster and takes 388s.

We perform a grid search over $\rho_s = \{0.25, 0.5, 1, 2\}$ and $\rho_{mc} = \{1.01, 1.5, 2, 3, 4, 6, 10\}$ to select hyperparameter values for SampleClust. We run RecAgglo with every hyperparameter combination on TrainG-30K (10 runs) and TrainG-100K (5 runs) computing I, *CFR* and computation time t. A detailed analysis of the grid search results on TrainG-30K is presented in App.C.3. It shows that a too low ρ_{mc} value (e.g., 1.01) or a high ρ_s value significantly increases the computation time of RecAgglo. We selected $\rho_s = 0.5$ and $\rho_{mc} = 6$ as these hyperparamters provide the best tradeoff with $t = 4,110s$, $CFR = 34.0\%$ and $I = 3.0\%$ on TrainG-100K. The sample size for sampling is $0.5 \times \sqrt{n}$ and the maximum number of clusters to generate is $maxclust = n/6$.

7.2 Experimental setup

We use four categorical clustering algorithms to compare the performance of RecAgglo: AggloClust, SampleClust, Kmodes and ROCK. We ran experiments on a consumer grade laptop with 8GB of RAM and Intel Core i5 (2.7GHz) processor. We already presented AggloClust and SampleClust in Sect. 3.1.

Kmodes [22] is an extension of the *Kmeans* algorithm for categorical data. It starts by selecting k random points as starting "modes" and calculates the distance between each element-mode pair. It assigns each element to the cluster that has the closest pairwise distance to its mode. Modes and clusters are updated over several iterations. Kmodes uses *Hamming* as distance metric and its time complexity is $O(n \times k)$. We use the PyPi implementation of Kmodes [42].

ROCK [20] is a clustering algorithm that uses a concept of "neighbor". Two elements are neighbors if the distance between them is lower than a threshold θ. Then, if two elements have enough common neighbors they are placed in the same cluster. ROCK uses the *Jaccard index* as distance metric and its time complexity is $O(n^2 \times log(n) + n^2)$. We adjust the ROCK implementation from [38] to accommodate for categorical data.

Table 1: Impurity, *CFR*, and computation time for 4 categorical clustering algorithms. Results are averaged over 10 runs on TestF-15K (15,000 samples). *: results for ROCK are computed on 5,000 randomly picked samples. RecAgglo generates the clusters with the lowest impurity in a short time.

Algorithm	I (%)	*CFR* (%)	time
RecAgglo$_{\delta_{max}=0.5}$	0.8	42.1	185s
AggloClust$_{\delta_{max}=0.5}$	1.2	51.9	1h31
SampleClust$_{\delta_{max}=0.5,\rho_s=0.5}$	1.1	1.2	38s
SampleClust$_{\delta_{max}=0.6,\rho_s=0.5}$	3.3	2.2	38s
SampleClust$_{\delta_{max}=0.6,\rho_s=2}$	2.4	7.7	158s
Kmodes$_{k=1,000}$	20.5	99.8	20m
Kmodes$_{k=5,000}$	14.1	91.6	1h31
Kmodes$_{k=12,000}$	10.5	39.2	7h44
*ROCK$_{\theta=0.55,t=0.45}$	7.1	51.4	3h08
*ROCK$_{\theta=0.45,t=0.40}$	0.9	30.3	1h48

7.3 Performance analysis

We cluster the 10 TestF-15K subsets (15,000 orders each) and average I, *CFR* and t over the 10 runs. ROCK was not able to cluster 15,000 elements in a reasonable amount of time and its results are computed on a random sample of 5,000 elements from TestF-15K. These results are summarized in Tab. 1 Kmodes is not able to generate a clustering with low impurity ($I > 0.1$) despite the high number of clusters we try to generate (up to k =12,000 as generated by AggloClust). SampleClust generates a clustering with low impurity in short time. However, it clusters only a very small number of frauds ($CFR < 0.08$). ROCK produces a clustering with low impurity $I = 0.009$ and higher $CFR = 0.303$ but its computation time on 5,000 samples is prohibitive (>1h30). AggloClust is the algorithm providing the best trade-off between impurity and *CFR*. It clusters over half of the frauds with an impurity close to 1%. Its computation time of 1h31 is prohibitive for a fairly small dataset.

RecAgglo has the second lowest computation time (3 minutes) and the clusters it generates have the lowest impurity of all algorithms ($I = 0.008$). RecAgglo achieves high $CFR = 0.421$, which is only 10 percentage points lower than the *CFR* of AggloClust. RecAgglo meets **R2** with its low impurity and **R3** with its relatively high *CFR*.

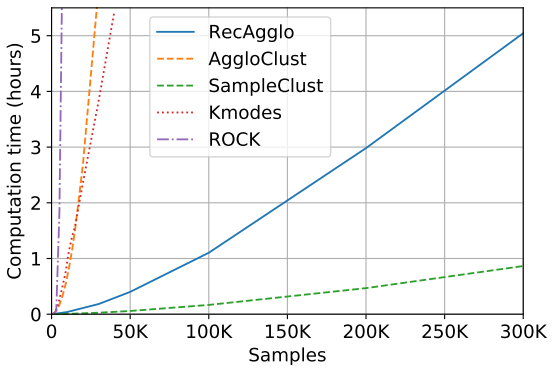

Figure 5: Computation time (averaged over 5 runs) vs. sample size for 5 clustering algorithms. Only SampleClust and RecAgglo can scale to large datasets.

We timed the clustering of an increasing set of orders from DE-real (up to $n = 300,000$) to assess the scalability of these algorithms. We report the running time averaged over 5 runs in Fig. 5. We use settings from Tab. 1 providing the lowest computation time: SampleClust ($\delta_{max} = 0.5, \rho_s = 0.5$) and ROCK ($\theta = 0.45, t = 0.40$). For Kmodes, we use $k = min(n/2, 5000)$, bounding k to 5,000 to limit the complexity of Kmodes (function of k).

ROCK, AggloClust and Kmodes cannot process 50,000 orders in less than 5 hours. Their complexity is at least quadratic, which prevent scaling to large datasets. SampleClust is the fastest algorithm, it clusters 300,000 samples in less than one hour. Finally, we see that RecAgglo scales to medium-size datasets despite its high worst-case complexity (cf. Sect. 3.3). In our specific use case of clustering orders, RecAgglo is much faster than algorithms with quadratic complexity because it requires only 2-5 recursions (instead of n in the worst case) to obtain a final clustering that meets our goodness criterion. It is able to cluster 100,000 orders in one hour and its computation time increases almost linearly with the set size: one hour more per additional 50,000 orders. RecAgglo processes 300,000 samples in 5 hours, which is the number of orders received by Zalando daily. RecAgglo addresses **C4** and it meets the low computation time requirement **R4**: it can be used in a real-world deployment setting to cluster frauds.

8 REAL-WORLD FRAUD DETECTION

We assess the performance of RecAgglo for our applied cases of preventing organized fraud: **A1** prioritizing screening and **A2** automated fraud cancellation. We take the following real-world scenario and we evaluate it using our real-world datasets. We take a set of unlabeled orders O_u placed over one day (e.g., in a 24h window). We have access to a set of labeled fraudulent orders O_f from an earlier time period (e.g., more than one day old). Our goal for **A1** is to provide human operators with clusters containing a maximum number of frauds from O_u. Our goal for **A2** is to use the resulting clusters to automatically and reliably detect a maximum number of organized frauds. This empirical evaluation assesses the performance of RecAgglo in its own right. It assumes a deployment in parallel of any existing detection system (as depicted in Fig. 1) and does not take into account the fraud detection pipeline that Zalando has already developed. The evaluation of the incremental value of RecAgglo with respect to the existing Zalando pipeline is not in the scope of this paper.

Table 2: Performance of RecAgglo at clustering real-world orders from 5 countries. Different delays for obtaining fraud labels (1 day/30 days). Cluster impurity (I), ratio of unlabeled (CFR_u) and overall (CFR) frauds clustered. Ratio of legitimate orders clustered (CLR). Frauds are clustered 3-7 times more than legitimate orders. Clusters do not mix legitimate orders and frauds.

	1 day delay for labels				30 days delay for labels			
Dataset	I	CFR	CFR_u	CLR	I	CFR	CFR_u	CLR
DE-real	0.9	51.9	43.4	9.6	0.8	52.8	33.3	9.6
CH-real	2.9	49.8	45.8	17.7	2.6	52.1	37.3	17.8
NL-real	1.2	34.8	34.8	7.9	1.0	33.1	26.0	7.9
BE-real	1.2	49.8	41.8	13.0	1.1	48.6	29.9	13.0
FR-real	0.3	50.4	44.1	7.1	0.2	47.7	32.1	7.1
Overall	1.3	49.7	43.5	10.9	1.1	50.2	33.7	11.0

8.1 Prioritizing screening

To effectively prioritize screening we must provide human analysts with a manageable amount of orders that contains a large portion of frauds. If we screen only clusters generated by RecAgglo, these clusters must contain a large amount of fraud (maximize CFR) and a low amount of legitimate orders (minimize CLR). CLR is the ratio of clustered legitimate orders, equivalent to CFR but for legitimate orders. Considering our real-world scenario, we cluster orders in $O_u \cup O_f$ using RecAgglo. We cluster the merged sets together since frauds from O_u can be related to frauds from O_f. We want our clusters to maximize CFR_u, which is the CFR computed solely on O_u, i.e., the portion of fraudulent orders form O_u that are clustered. Recall that we want our clusters to have low impurity I and to keep a low CLR to reduce the workload of human analysts.

We take orders from each day in the datasets DE-real, CH-real, NL-real, BE-real and FR-real to create 35 sets O_u per country. We create associated sets O_f of frauds from a prior period of 60 consecutive days using DE-bg, CH-bg, NL-bg, FR-bg and BE-bg. We consider two scenarios for composing O_f that depict the delay in obtaining fraud labels. The first scenario assumes a one day delay meaning that O_f contains frauds from the period of 60 days ending when O_u starts. The second scenario assumes a 30 days delay meaning that O_f contains frauds from a period ending 30 days before O_u ends. We selected 30 days because it is a sufficient delay for preliminary identification of orders being in payment default. We cluster the 35 resulting sets $O_u \cup O_f$ for each country and present averaged I, CFR, CFR_u and CLR per country in Tab. 2.

RecAgglo produces clusters having low impurity which varies depending on the country. Notably, there is a 10-fold difference between Switzerland ($I = 2.9\%$) and France ($I = 0.3\%$). This shows that legitimate orders from, e.g., Switzerland or Belgium, are more similar to frauds than this is the case in France or Germany. The CLR is 3- to 7-times lower than the CFR showing that RecAgglo

Table 3: Recall, precision and false positive rate (FPR) for automated fraud detection in 5 countries. One quarter of frauds are detected while generating a few false alarms (0.1%). Only 35.3% of detected frauds are actual frauds.

Dataset	$Recall_{clust}$	$Recall_{final}$	$Precision$	FPR
DE-real	59.8	26.2	35.9	0.1
CH-real	72.3	33.2	17.0	0.3
NL-real	60.4	20.5	29.3	0.1
BE-real	63.6	26.5	34.4	0.2
FR-real	65.8	30.0	71.4	0.1
Overall	62.6(±10.6)	26.4(±5.5)	35.3(±6.3)	0.1(±0.03)

clusters more frauds than legitimate orders. The CFR_u is around 35-45%, which is slightly lower than the CFR but consistently larger than the CLR (7-18%). For screening prioritization **A1**, around 90% of legitimate orders could be discarded from manual analysis while preserving around 40% of frauds that could be detected. RecAgglo can be used to prioritize screening.

We see that the increased delay in obtaining fraud labels decreases the CFR_u by around 10 percentage points. There is a strong time dependence between orders belonging to the same cluster and to the same fraud campaign. Obtaining fraud labels in a timely manner is crucial to maximize our ability to detect organized fraud. It is worth noting that while we ran RecAgglo once per day in this experiment, it can be re-ran continuously as new orders are received (using a sliding window containing one day of orders O_u). We recall that RecAgglo processes 100,000 orders in one hour.

8.2 Automated fraud cancellation

We devise a simple technique to automatically detect and cancel frauds using clusters generated by RecAgglo. We detect an unlabeled order from O_u as fraud (1) if it is clustered and (2) if at least one known fraud from O_f belongs to this cluster. We call this technique *label propagation* - known fraud propagates its label to the whole cluster it belongs to.

We apply this technique to the clusters generated in Sect. 8.1. For each country in Tab. 3, we report *Recall*, *Precision* and False Positive Rate (*FPR*) averaged over 35 clustering results (35 days). *Recall* is the ratio of correctly detected frauds (TP) over the total number of frauds ($TP + FN$) and it represents the ability to detect frauds. $Recall_{clust}$ is computed only on clustered frauds, while $Recall_{final}$ is computed on all frauds. $Recall_{final} = Recall_{clust} \times CFR_u$ is the actual rate of detected frauds. *Precision* is the ratio of correctly detected frauds (TP) over the total number of detected frauds ($TP + FP$), i.e., the reliability of fraud detection. FPR is the ratio of legitimate orders incorrectly detected as fraud (FP) over the total number of legitimate orders ($TN + FP$). It corresponds to the error rate for legitimate orders.

This simple automated detection technique would cancel over one quarter ($Recall_{final} = 26.4\%$) of fraud, which likely represents 62.6% ($Recall_{clust}$) of all organized fraud. It also generates few false alarms ($FPR = 0.1\%$) for legitimate orders. Despite this very low $FPR = 0.1\%$, the precision remains low because of the large imbalance between fraud and legitimate orders ($\#fraud \ll \#legitimate$). Our average precision of 35.4% means that 2/3 orders detected as fraud are actually legitimate orders. This low precision is prohibitive for automated fraud cancellation **A2** but it can prioritize screening **A1**. Human analysts could cancel over 25% of frauds with little effort required.

We investigated further the characteristics of legitimate orders that our *label propagation* technique incorrectly identified as fraud. We computed the ratio of these orders that belong to four legitimate categories namely, (1) *fully* and (2) *partially returned* to the retailer (where a customer does not pay for all items and returns some of them), (3) *partly unpaid* (where items in the order remain unpaid while delivered) and (4) *canceled* by the customer. We observed that 94.7% of the false positives that degrade the Precision of *label propagation* belong to one of these four categories. The majority of the false positives (63.9%) are returned orders while 24.3% are partly unpaid orders.

8.3 Evading fraud detection

Recent research in adversarial machine learning [25, 40] has shown that machine learning-based systems can be evaded by manipulating their inputs [18]. Fraudsters can evade our fraud detection approach by making their orders less similar to one another. This would result in RecAgglo not being able to group together frauds from the same campaign. RecAgglo quantifies the similarity between orders by computing the Hamming distance. If two attributes have different values, the distance between two orders increases and their similarity decreases. Our attributes have a string representation. The inequality between two attribute values can be obtained by modifying a single character from one of them. Such a modification (e.g., typo in street address) may have no impact on successfully placing and receiving an order (fraud purpose), while its similarity to other orders could be greatly reduced. An adversary can modify order attributes that are input to RecAgglo to evade fraud detection.

This limitation can be addressed by using only attributes that are resilient to adversarial manipulations in RecAgglo [32]. Resilient attributes are those for which manipulations inherently defeat the fraud purpose. For instance, a small modification to a credit card number makes payment information inconsistent, which causes rejection of payment and of the order. Credit card number is an attribute resilient to adversarial manipulations. Alternatively, we can use a metric more fine-grained than binary equality for string comparison. The edit distance can accurately quantify the similarity between two strings and it accommodates small adversarial modifications. An adversary would have to make large modifications to reduce the similarity between two orders, which makes evasion harder. Nevertheless, the edit distance is expensive to compute and it will increase the running time of RecAgglo clustering.

9 RELATED WORK

9.1 Categorical clustering

Categorical clustering faces two main challenges: scalability and guarantee of convergence. Generic algorithms such as Kmodes [22], *ROCK* [20], *CACTUS* [15] (greedy hierarchical grouping of tuples) or *CLICKS* [50] (representing the dataset as a graph and finding disjoint vertices) provide good guarantees of convergence and many recent clustering algorithms [6, 16] are built upon these techniques.

The main alternative approaches use information-theoretic criteria to assess the quality of the clustering. For instance, *COOLCAT* [5] searches locally to find clusters with the lowest entropy, while *LIMBO* [4] produces a hierarchical summary of the data that preserves as much information as possible. While generalizable to any kind of categorical data, these algorithms have high complexity and they do not scale to large datasets.

To improve scalability, the data can be either partitioned into chunks that are clustered individually and then combined into global clusters [6] or it can be transformed into representations that make processing faster, e.g., Merkle trees [28]. These approaches are typically problem specific (images [28], streamed data [19]) and they are not applicable to categorical data. A more generic solution produces several clusterings using subsets of features and aggregates the results to obtain global clusters [16]. Similarly, low-dimensional clusters can be generated using dissimilarity matrices and then combined using an ensemble method to get the final clustering [3]. These methods use sampling, as we do, but they take a local approach, trying to reduce the dimension of the input space to improve scalability.

9.2 Fraud detection

Fraud detection is a sparse subject relevant to many domains: credit card fraud [10, 17], tax evasion [7], online dating [46], erotic content [23], advertising [36], among others. Despite efforts to systemize it [8, 37, 43], there is no commonly accepted means of comparing different detection techniques or application scenarios.

Many solutions try to detect frauds in isolation using supervised classifiers such as neural networks [2, 9, 17, 29] and ensemble of decision trees [10]. Improvements have been proposed to incorporate the time component in the process of eCommerce fraud detection by identifying changes in the underlying distribution of orders (also known as *concept drift*) [31]. Finally, one can analyze the temporal activity of users from a small set of features to predict possible account take-over [21]. These solutions analyze orders in isolation or in a group related to a specific user and they are not suitable for identifying organized fraud that involves many users. Also, they require labeled data that is not available for fraud campaigns.

Graph-based methods can be used to take a global view at the fraud detection problem [1, 34, 44, 47]. Social network analysis methods [1, 47], the PageRank algorithm [44] and sets of graph-derived features [34] have been used to spot frauds in the payment network and to identify the key links between frauds. However, these methods require having access to (or constructing) a graph with a well-defined notion of the vertex and edge (e.g. credit card and merchant vertices and interaction as an edge). In our case, we do not have a proper link between orders that we could use to build a graph. Instead, we try to identify similarities through clustering. Moreover, building a graph with a static set of assumptions could hinder the performance of the graph analysis method if the nature of fraud changes overtime.

A few works also proposed to use clustering to identify fraudulent activity, as we do. However, these methods either use only numerical features as input [48] or they do not scale to data of such a scale as ours (100,000s samples) [26].

10 CONCLUSION

We introduced a novel clustering solution (RecAggLo) to detect organized fraud and we evaluated it on 6M real-world orders placed on the Zalando website. We showed RecAggLo is able to process 100,000s of orders in a few hours and it groups over 40% of fraudulent orders together. The algorithm can be deployed and used to efficiently prioritize screening **A1**. We further proposed a simple technique named *label propagation* that uses our generated cluster to automatically detect 26.2% of fraud while raising false alarms for only 0.1% of legitimate orders. In spite of its high accuracy, *label propagation* incorrectly identifies many legitimate orders as fraud (35% precision). Considering our definition of what "fraud" is, *label propagation* cannot be used for automated fraud cancellation **A2**. Nevertheless, we observed that 95% of the legitimate orders incorrectly identified as fraud by *label propagation* are either returned, partly unpaid or canceled orders.

Canceling legitimate orders from good customers can create a lot of harm to a business [12]. It generates a bad customer experience with a detrimental effect on customer satisfaction and customer lifetime value, which may ultimately decrease the profitability of an online retail service. "Fraud" has a subjective definition that is different for different online retailers. The effectiveness and deployability of a fraud detection system are evaluated according to this definition. Hence, the suitability of our solution to prioritize screening **A1** and automatically cancel fraud **A2** depends on the categorization of returned (fully/partly), unpaid (fully/partly) and canceled orders for a given online retailer.

ACKNOWLEDGMENTS

This research was funded by a research donation from Zalando Payments GmbH. It is supported by the Academy of Finland through the SELIoT Project (Grant 309994). We thank Zalando employees and N. Asokan for interesting discussions and valuable feedback.

REFERENCES

[1] Leman Akoglu, Mary Mcglohon, and Christos Faloutsos. 2009. *Anomaly detection in large graphs (CMU-CS-09-173).* Technical Report.
[2] E. Aleskerov, B. Freisleben, and B. Rao. 1997. CARDWATCH: a neural network based database mining system for credit card fraud detection. In *Proceedings of the IEEE/IAFE Computational Intelligence for Financial Engineering (CIFEr).* 220–226.
[3] Saeid Amiri, Bertrand S. Clarke, and Jennifer L. Clarke. 2018. Clustering Categorical Data via Ensembling Dissimilarity Matrices. *Journal of Computational and Graphical Statistics* 27, 1 (2018), 195–208.
[4] Periklis Andritsos, Panayiotis Tsaparas, Renee J. Miller, and Kenneth C. Sevcik. 2004. LIMBO: Scalable clustering of categorical data. In *9th International Conf. on Extending DataBase Technology.* 123–146.
[5] 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.* 582–589.
[6] Malika Bendechache, Nhien-An Le-Khac, and M. Tahar Kechadi. 2016. Efficient Large Scale Clustering Based on Data Partitioning. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (2016), 612–621.
[7] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. 2015. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *International conference on financial cryptography and data security.* Springer, 227–234.
[8] Richard J. Bolton and David J. Hand. 2002. Statistical Fraud Detection: A Review. *Statist. Sci.* 17, 3 (08 2002), 235–255.
[9] R. Brause, T. Langsdorf, and M. Hepp. 1999. Neural Data Mining for Credit Card Fraud Detection. In *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence.* IEEE Computer Society, 103–106.
[10] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. 2018. Scarff: a scalable framework for streaming credit card fraud detection with spark. *Information fusion* 41 (2018), 182–194.

[11] Thomas Chmielewski and Denise James. 2009. Mortgage fraud detection systems and methods. US Patent 7,546,271.

[12] Clearsale. 2017. Are False Declines Hurting Your Online Reputation? last accessed June 3, 2019. https://blog.clear.sale/false-declines-hurting-your-online-reputation

[13] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. 2018. Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. *IEEE Transactions on Neural Networks and Learning Systems* 29, 8 (2018), 3784–3797.

[14] Europol. 2018. 95 e-commerce fraudsters arrested in international operation. last accessed June 3, 2019. https://www.europol.europa.eu/newsroom/news/95-e-commerce-fraudsters-arrested-in-international-operation

[15] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. 1999. CACTUS&Mdash;Clustering Categorical Data Using Summaries. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*. 73–83.

[16] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. 2007. Clustering Aggregation. *ACM Trans. Knowl. Discov. Data* 1, 1, Article 4 (March 2007).

[17] Jon Ander Gomez, Juan Arvalo, Roberto Paredes, and Jordi Nin. 2018. End-to-end Neural Network Architecture for Fraud Scoring in Card Payments. *Pattern Recogn. Lett.* 105, C (April 2018), 175–181.

[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[19] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. 2003. Clustering Data Streams: Theory and Practice. *IEEE Trans. on Knowl. and Data Eng.* 15, 3 (2003), 515–528.

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

[21] Hassan Halawa, Matei Ripeanu, Konstantin Beznosov, Baris Coskun, and Meizhu Liu. 2018. Forecasting Suspicious Account Activity at Large-Scale Online Service Providers. *CoRR* abs/1801.08629 (2018). arXiv:1801.08629

[22] Zhexue Huang. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery* 2, 3 (1998), 283–304.

[23] Alice Hutchings and Sergio Pastrana. 2019. Understanding eWhoring. *arXiv* (2019). arXiv:cs.CR/1905.04576

[24] Lou Jost. 2006. Entropy and diversity. *Oikos* 113, 2 (2006), 363–375.

[25] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.

[26] Marie-Jeanne Lesot and Adrien Revault d'Allonnes. 2012. Credit-Card Fraud Profiling Using a Hybrid Incremental Clustering Methodology. In *Scalable Uncertainty Management*, Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 325–336.

[27] Michael Levi. 2008. Organized fraud and organizing frauds: Unpacking research on networks and organization. *Criminology & Criminal Justice* 8, 4 (2008), 389–419.

[28] Ting Liu, Charles Rosenberg, and Henry A Rowley. 2007. Clustering billions of images with large scale nearest neighbor search. In *2007 IEEE Workshop on Applications of Computer Vision*. 28–28.

[29] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. 2002. Credit Card Fraud Detection Using Bayesian and Neural Networks. (08 2002), 261–270.

[30] Oded Maimon and Lior Rokach. 2005. *Data mining and knowledge discovery handbook*. Springer.

[31] Huiying Mao, Yung-wen Liu, Yuting Jia, and Jay Nanduri. 2018. Adaptive Fraud Detection System Using Dynamic Risk Features. *arXiv preprint arXiv:1810.04654* (2018).

[32] Samuel Marchal, Giovanni Armano, Tommi Gröndahl, Kalle Saari, Nidhi Singh, and N Asokan. 2017. Off-the-hook: An efficient and usable client-side phishing prevention application. *IEEE Trans. Comput.* 66, 10 (2017), 1717–1733.

[33] Samuel Marchal and Sebastian Szyller. 2019. Recursive Agglomerative Clustering (RecAgglo) for categorical data. last accessed August 30, 2019. https://github.com/SSGAalto/recagglo

[34] Ian Molloy, Suresh Chari, Ulrich Finkler, Mark Wiggerman, Coen Jonker, Ted Habeck, Youngja Park, Frank Jordens, and Ron van Schaik. 2017. Graph Analytics for Real-Time Scoring of Cross-Channel Transactional Fraud. In *Financial Cryptography and Data Security*. 22–40.

[35] David Montague. 2014. The Fraud Practice. last accessed June 3, 2019. http://fraudpractice.com/gl-manual.html

[36] Shishir Nagaraja and Ryan Shah. 2019. Clicktok: Click Fraud Detection Using Traffic Analysis. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '19)*. 105–116.

[37] Xuetong Niu, Li Wang, and Xulei Yang. 2019. A Comparison Study of Credit Card Fraud Detection: Supervised versus Unsupervised. *arXiv* (2019). arXiv:cs.LG/1904.10604

[38] Andrei Novikov. 2019. PyClustering: Data Mining Library. *Journal of Open Source Software* 4, 36 (apr 2019), 1230–1230. https://doi.org/10.21105/joss.01230

[39] Aaron Orendorf. 2019. What Is the Future of eCommerce? 10 Insights on the Evolution of an Industry. last accessed May 25, 2019. https://www.shopify.com/enterprise/the-future-of-ecommerce

[40] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. 2018. SoK: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 399–414.

[41] Signifyd & PYMNTS.com. 2017. *Global Fraud Index*. Technical Report. PYMNTS.com.

[42] PyPi. 2019. Kmodes clustering. https://pypi.org/project/kmodes/

[43] Andrei Sorin Sabau. 2012. Survey of Clustering based Financial Fraud Detection Research. *Informatica Economica* 16, 1 (2012), 110–122.

[44] Alex Sangers, Maran van Heesch, Thomas Attema, Thijs Veugen, Mark Wiggerman, Jan Veldsink, Oscar Bloemen, and Daniël Worm. 2018. Secure multiparty PageRank algorithm for collaborative fraud detection. *IACR Cryptology ePrint Archive* 2018 (2018), 917.

[45] Edward H Simpson. 1949. Measurement of diversity. *Nature* 163, 4148 (1949), 688.

[46] Guillermo Suarez-Tangil, Matthew Edwards, Claudia Peersman, Gianluca Stringhini, Awais Rashid, and Monica Whitty. 2019. Automatically Dismantling Online Dating Fraud. (05 2019).

[47] Lei Tang, Geoffrey Barbier, Huan Liu, and Jianping Zhang. 2010. A social network analysis approach to detecting suspicious online financial activities. In *International Conference on Social Computing, Behavioral Modeling, and Prediction*. Springer, 390–397.

[48] Sutapat Thiprungsri and Miklos A Vasarhelyi. 2011. Cluster Analysis for Anomaly Detection in Accounting Data: An Audit Approach. *International Journal of Digital Accounting Research* 11 (2011), 69–84.

[49] Zalando website. 2019. last accessed June 3, 2019. https://www.zalando.com

[50] Mohammed J. Zaki, Markus Peters, Ira Assent, and Thomas Seidl. 2005. CLICKS: An Effective Algorithm for Mining Subspace Clusters in Categorical Datasets. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 736–742.

[51] Zalando. 2015. Half-year report January-June 2015. last accessed August 16, 2019. https://corporate.zalando.com/sites/default/files/media-download/zalando_se_half-year_report_2015_e_s.pdf

[52] Zalando. 2019. *Annual Report 2018 - Key figures*. Technical Report. Zalando. https://corporate.zalando.com/en/investor-relations/key-figures-2018

A CLUSTERING ALGORITHMS DETAILS

A.1 Agglomerative clustering

The agglomerative clustering algorithm is detailed in Algorithm 2. DistanceMatrix compute a $|c_i| \times |c_j|$ matrix of the distance between elements of c_i and c_j. *Hamming* distance and *Jaccard Index* are the most widely used [30] methods for measuring the distance of categorical data. We select *Hamming* distance since it is fast to compute. It counts the number of different attribute values between two elements u and v (cf. Eq. 1).

The linkage matrix computed using LinkageMatrix describes the successive cluster fusions to go from singletons to a single cluster (dendrogram). Cluster fusion is done according to the distance between two clusters, which is defined by a *linkage* method. *Single linkage* uses the minimum distance between any two points in each cluster. *Complete linkage* takes the maximum distance between any two points from each cluster. *Ward linkage* [30] takes the increase in the sum of square obtained by merging two clusters rather than by keeping them separate. We use the single linkage method here. We compute the final clustering using Cluster based on the linkage matrix and the *'distance'* criterion parametrized by d_{max}.

A.2 Agglomerative clustering with sampling

The algorithm for agglomerative clustering with sampling is presented in Algorithm 3. We modify the basic sampling algorithm to create a maximum number of clusters. Rather than initially clustering sampled elements, we use each of them as the basis for a new cluster (no initial clustering). Our sampling algorithm randomly

Algorithm 2 Agglomerative clustering

Let $c = v_1, \ldots, v_m$ denote a set of $|c| = m$ elements v,
d_{max}, the maximum distance for cluster fusion.

function AggloClust(c, d_{max})
    $D \leftarrow$ DistanceMatrix(c, c, 'Hamming')
    $LM \leftarrow$ LinkageMatrix(D, 'single')
    $C_{res} \leftarrow$ Cluster(LM, d_{max}, 'distance')
    return C_{res}
end function

selects n samples from c (where n is a factor ρ_s of $\sqrt{|c|}$). Thus, we compute the distance matrix between the sample and all elements in c. We use the single linkage method to compute the linkage matrix since we only know the distance to a single element in each cluster (because of sampling). Finally, it generates a clustering C_{res} using the maximum number of clusters criterion (*'maxclust'*) where c_{max} is computed as a factor ρ_{mc} of the set size $|c|$. This criterion is selected with the goal of splitting a large set of elements into smaller clusters but without providing any guarantee of goodness for the resulting clustering.

Algorithm 3 Agglomerative clustering with sampling

Let $c = v_1, \ldots, v_m$ denote a set of $|c| = m$ elements v,
ρ_s, the multiplying factor for the sample size,
ρ_{mc}, the dividing factor for maxclust number.

function SampleClust(c, ρ_s, ρ_{mc})
    $n \leftarrow \rho_s \times \sqrt{|c|}$
    $c_{max} \leftarrow m \div \rho_{mc}$
    $s \leftarrow$ RandomSample(c, n)
    $D \leftarrow$ DistanceMatrix(s, c, 'Hamming')
    $LM \leftarrow$ LinkageMatrix(D, 'single')
    $C_{res} \leftarrow$ Cluster(LM, c_{max}, 'maxclust')
    return C_{res}
end function

B DATASETS COMPOSITION

Small datasets with artificial distribution: We selected TrainF-15K and TestF-15K from Zalando orders passed in France over 2017. In each dataset, we simulate an artificial distribution where frauds are over-sampled compared to a real-world distribution (2 legitimate / 1 fraud). TrainF-15K consists of 10 disjoint subsets TrainF-15K-i, each composed of 10,000 legitimate and 5,000 frauds. The 5,000 frauds are randomly selected from a continuous period of 1-1.5 month. The 10,000 legitimate orders are randomly selected from a period of 1-2 days. For each subset i, the period from which legitimate orders are selected is included into the period from which frauds are selected. TestF-15K consists of 10 disjoint subsets TestF-15K-i selected the same way as for TrainF-15K (10,000 legitimate and 5,000 frauds) TestF-15K and TrainF-15K are disjoint.

Large datasets with artificial distribution. We select TrainG-30K and TrainG-100K from orders passed in Germany over 2017. Each subset of TrainG-30K is composed of 25,000 legitimate and 5,000 frauds. The 5,000 frauds are randomly selected from a continuous period of 1 month. The 25,000 legitimate orders are randomly selected from a period of 1 day contained in the month from which frauds are selected. Each subset of TrainG-100K is composed of 95,000 legitimate and 5,000 frauds. The 5,000 frauds are randomly selected from a continuous period of 1 month. The 95,000 legitimate orders are randomly selected from a period of 1 week contained in the month from which frauds are selected.

C HYPERPARAMETER SELECTION

C.1 Linkage method selection

We want to select a linkage method that minimizes the impurity I and maximizes the clustered fraud rate (CFR). We cluster the 10 TrainF-15K-i datasets using *single* linkage, *complete* linkage and *Ward* linkage. Figure 6 shows the evolution of impurity I according to CFR while varying the maximum distance for cluster fusion d_{max}. Values are averaged over 10 clustering results. We see that for $CFR < 0.6$ all linkage methods have similar impurity values. For $CFR > 0.6$ *Ward* outperforms other linkage methods, while *single* becomes the worst method with high increase in impurity.

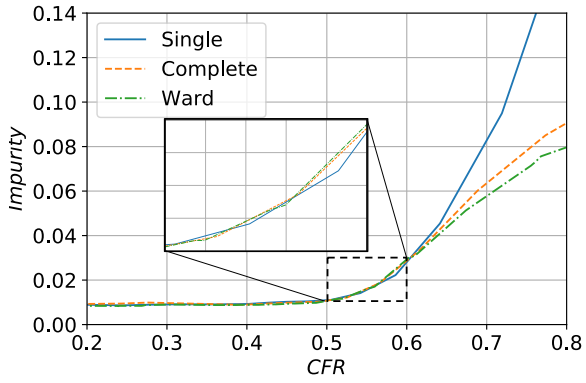

Figure 6: Impurity vs. *CFR* for single / complete / Ward linkage methods. Each linkage method provides the same Impurity/*CFR* tradeoff when the impurity is low: 0.01-0.03.

Our primary goal is to keep the impurity as low as possible. All methods are comparable at providing a high CFR while keeping the impurity low (0.01 - 0.03). *Single* and *complete* linkage are computed using a single distance between two points from two clusters: the closest and the furthest away ones respectively. Thus, they are faster to compute than *Ward*. They are also better suited for clustering with sampling, since the distance between any two points is not available using sampling. We can see from Fig. 6 (zoom) that single linkage provides higher CFR than complete linkage for the same impurity value. Thus, we select *single* linkage as our base linkage technique.

C.2 Weighting strategy selection

Figure 7 shows the evolution of impurity I according to CFR while varying the maximum distance for cluster fusion d_{max}. Values are averaged over 10 clustering results. We see that for $CFR < 0.55$ all weighting strategies have similar impurity values. For $CFR > 0.55$

Table 4: Results of grid search for SampleClust hyperparameters: $\rho_s = \{0.25, 0.5, 1, 2\}$ and $\rho_{mc} = \{1.01, 1.5, 2, 3, 4, 6, 10\}$. Average impurity, *CFR* and computation time computed over 10 runs of RecAgglo on TrainG-30K ($\delta_a = 1,000$). A low performance score means a combination of hyperparameters that maximizes our clustering objectives: low impurity, high CFR and low computation time.

ρ_{mc}	ρ_s	I (%)	*CFR* (%)	time (s)	performance score
1.01	0.25	3.61	28.59	2,070	3.06
	0.5	3.65	29.54	4,722	5.10
	1	3.75	30.13	9,049	9.33
	2	3.68	30.53	16,657	15.90
1.5	0.25	3.43	27.72	461	1.17
	0.5	3.45	28.36	657	1.00
	1	3.38	28.49	1,010	0.89
	2	3.49	28.74	1,365	1.61
2	0.25	3.38	27.45	397	0.98
	0.5	3.39	28.29	620	0.72
	1	3.43	28.48	920	1.09
	2	3.53	28.70	1,300	1.82
3	0.25	3.43	27.42	410	1.30
	0.5	3.42	28.31	629	0.86
	1	3.44	28.82	913	0.91
	2	3.56	28.75	1,262	1.91
4	0.25	3.46	27.67	417	1.29
	0.5	3.51	28.65	632	1.12
	1	3.40	28.71	884	0.72
	2	3.48	28.76	1,265	1.49
6	0.25	3.49	27.56	456	1.57
	0.5	3.44	28.48	682	0.88
	1	3.43	28.76	936	0.88
	2	3.48	28.78	1,350	1.52
10	0.25	3.41	27.89	540	0.99
	0.5	3.44	28.43	720	0.94
	1	3.47	28.81	995	1.12
	2	3.53	28.89	1,456	1.83

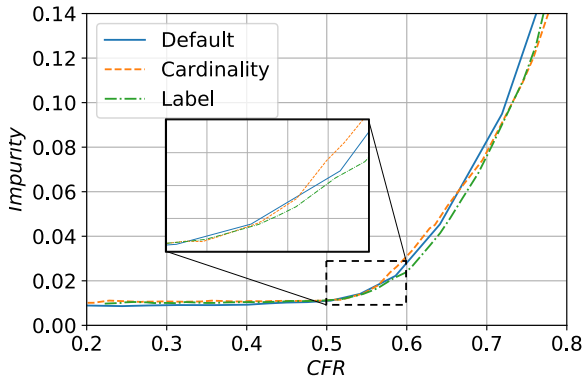

Figure 7: Impurity vs. *CFR* for default / cardinality / label driven attribute weighting. All methods have comparable performance. Label driven weighting provides marginally higher *CFR* for the same impurity.

our label driven weighting becomes better than other strategies providing up to a 2 percentage points higher *CFR* than other strategies while keeping impurity low (0.025). Cardinality driven features provide the best trade-off between *CFR* and impurity (for $I > 0.10$). This is an interesting result but it is not useful since our goal is to minimize impurity. We see that label driven weighting gives the best trade-off between *CFR* and impurity.

C.3 SampleClust hyperparameters selection

Table 4 presents the impurity (I), *CFR* and computation time (t) for each combination of SampleClust hyperparameters ρ_{mc} and ρ_s tested during the grid search. These results are limited to the grid search we performed on TrainG-30K and they depict how our performance metrics vary according to ρ_{mc} and ρ_s values. We also computed a *performance score* to choose the optimal hyperparameter combination. It is the sum of normalized performance metrics $\hat{I}$, $\hat{CFR}$ and $\hat{t}$, which are respectively defined in Eq. (8), (9) and (10)[3]. A low performance score depicts a combination of ρ_{mc} and ρ_s that maximizes our clustering objectives: low impurity, high CFR and low computation time.

$$\hat{I}_{(\rho_s=x,\rho_{mc}=y)} = \frac{I_{(\rho_s=x,\rho_{mc}=y)} - min(I)}{max(I) - min(I)} \tag{8}$$

$$\hat{CFR}_{(\rho_s=x,\rho_{mc}=y)} = \frac{max(CFR) - CFR_{(\rho_s=x,\rho_{mc}=y)}}{max(CFR) - min(CFR)} \tag{9}$$

$$\hat{t}_{(\rho_s=x,\rho_{mc}=y)} = \frac{t_{(\rho_s=x,\rho_{mc}=y)} - min(t)}{max(t) - min(t)} \tag{10}$$

We see that hyperparamater choice heavily impacts the computation time while impurity and *CFR* remain almost constant. Too low ρ_{mc} value (e.g., 1.01) or a high ρ_s value significantly increase the computation time. This is expected since ρ_s defines the sample size used in clustering with sampling. A high ρ_s value means a large sample.

[3]We discarded time results obtained using rho_{mc} from the normalization process in Eq. (10). These are too high and represent outliers. $max(t)$ and $min(t)$ only take $\rho_{mc} = \{1.5, 2, 3, 4, 6, 10\}$ into account.