FDHelper: Assist Unsupervised Fraud Detection Experts with Interactive Feature Selection and Evaluation

Jiao Sun[1†], Yin Li[1†], Charley Chen[1], Jihae Lee[1], Xin Liu[1], Zhongping Zhang[2], Ling Huang[1,3], Lei Shi[4*], Wei Xu[1*]

[1] Institute for Interdisciplinary Information Science, Tsinghua University
[2] Computer Science Department, Boston University [3] AHI Fin-tech Inc.
[4] ACT and BDBC, School of Computer Science and Engineering, Beihang University
{j-sun16,liyin16,ctj12}@tsinghua.org.cn, {lizhihui17,liuxin16}@mails.tsinghua.edu.cn, zpzhang@bu.edu
linghuang@fintec.ai, leishi@buaa.edu.cn, weixu@mail.tsinghua.edu.cn

ABSTRACT

Online fraud is the well-known dark side of the modern Internet. Unsupervised fraud detection algorithms are widely used to address this problem. However, selecting features, adjusting hyperparameters, evaluating the algorithms, and eliminating false positives all require human expert involvement. In this work, we design and implement an end-to-end interactive visualization system, FDHelper, based on the deep understanding of the mechanism of the black market and fraud detection algorithms. We identify a workflow based on experience from both fraud detection algorithm experts and domain experts. Using a multi-granularity three-layer visualization map embedding an entropy-based distance metric ColDis, analysts can interactively select different feature sets, refine fraud detection algorithms, tune parameters and evaluate the detection result in near real-time. We demonstrate the effectiveness and significance of FDHelper through two case studies with state-of-the-art fraud detection algorithms, interviews with domain experts and algorithm experts, and a user study with eight first-time end users.

Author Keywords

Human Computer Interaction, Fraud Detection, Visualization

CCS Concepts

•Human-centered computing → Visualization systems and tools; Information visualization;

INTRODUCTION

Nowadays, many online services are rife with frauds such as fake accounts on forums and video websites, and bots on social networks. Frauds conduct malicious activities, and thus compromise the business value of online services, causing millions of complaints and billion dollars of economic losses each year globally [2].

Online fraud detection is a well-known problem in the research community [18]. Researchers often formulate fraud detection as a binary classification problem that categorizes users into frauds and normal ones based on their log records. User logs usually come from web access records and demographic information provided upon registration. Most log records are categorical features, e.g., the cities, the operating system of their devices, the source IP addresses, timestamps, etc.

We use the following terminology throughout this paper. *Users* refer to the users in the log records, i.e., the subject for fraud detection. They are either *normal* or *fraud*. We refer to people who use FDHelper as *analysts*, which includes both *algorithm experts* and *domain experts*.

Researchers have proposed many fraud detection algorithms based on the grouping behavior of frauds, in particular, unsupervised learning methods [10, 13, 19]. However, it is challenging to design and evaluate these algorithms: 1) the log records contain many dimensions describing user behaviors, and it is hard to select the most relevant dimensions to fraud patterns; 2) the selection of feature sets and algorithms depends heavily on domains and scenarios; 3) there are very few or no fraud labels for training or evaluation. We can not confirm fraud behaviors until a materialized damage suffered, often after a long time. Meanwhile, finding an appropriate algorithm, choosing useful feature sets, and excluding false positives are all critical for a successful fraud detection process. Nevertheless, all of these processes need the involvement of human experts, for which visualization is an indispensable component in any successful fraud detection system.

The visualization for group-based fraud detection is different from traditional methods for clustering [24] and anomaly detection [3]: 1) fraud groups expose various structures under different sets of features; 2) the size of the user base can be huge (e.g., millions of users), making a full-scale visualization impractical; 3) the number of frauds can be even larger than normal users, making the visualization focusing on normal behavior impractical; 4) in addition to detecting frauds, analysts also need to interpret the detection result for verification, evaluation, and comparison of different algorithm configura-

[†]equal contribution; [*]corresponding authors

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6708-0/20/04 ...$15.00.
http://dx.doi.org/10.1145/3313831.3376140

tions for optimization. Finally, the detected fraud groups are characterized for knowledge externalization; 5) the instruction for configuring algorithms needs to be actionable, i.e., we should guide analysts on how to tune the algorithms.

Besides, we highlight the feature selection process in our system. There are three reasons why the interactive feature selection is necessary. 1) It is well known that feature selection is extremely important in the fraud detection process [8]; 2) in many cases, feature selection is a trial-and-error process, even for experts; 3) the visualization provides instant feedback for feature selection. After choosing different features, we run the fraud detection algorithm with new features selected. Algorithm experts can then access the visualization result using these new features. This interactive process dramatically reduces the time spent in refining the algorithm.

In this paper, we present FDHelper, an interactive visualization tool that integrates effective and visualization-friendly fraud detection algorithms with the visualization design. FDHelper integrates *feature selection*, *algorithm parameter tuning* and *algorithm refinement* into an interactive end-to-end visualization system. To the best of our knowledge, it is the first visual fraud detection system that reveals the detected fraud group structure. Analysts can explore the result of fraud detection algorithms at three levels of detail (i.e., the algorithm level, the feature level, and the hyperparameter level). Instead of trying to explain the detail of fraud detection algorithms, we keep the fraud detection algorithm as a black box and only provide user-friendly interfaces for users to interact with. The main contributions of this paper are as follows:

- An analysis of the fraud black market and an identification of user requirements in the interaction process with unsupervised fraud detection methods.
- An interactive visualization tool that enables users to *choose* (algorithm and dataset), *refine* (feature selection and algorithm setting) and *evaluate* (the detection result) during an unsupervised fraud detection process.
- An evaluation of FDHelper through two case studies and an interview with two kinds of fraud detection experts.

RELATED WORK

Frauds and the Black Market

The key to conducting online frauds lies in collecting disposable accounts on the target website. Fraudsters established a professional chain of fraud services through the dark web (aka, the black market) [21] to break into the defense of website providers with low cost (e.g., \$140 $\sim$ \$420 for 1000 mobile SIMs). Compared with legitimate accounts, frauds exhibit unusually similar behaviors in certain aspects, e.g., re-use of phone numbers, similar phone access durations, highly recurrent IP ranges, and the regular frequency of activities.

This resource sharing mechanism enables a fundamental way to detect fraud behaviors based on these resources, so the goal of fraud detection algorithms is efficiently discovering these grouping behaviors. In other words, it is much more effective to hunt down these resource-sharing groups than detecting just a few individual accounts.

Algorithmic Fraud Detection

Nevertheless, it is non-trivial to detect these grouping behaviors. Researchers always take unsupervised detection algorithms because frauds are not labeled. There are two kinds of algorithms to detect such behaviors: feature-vector based like [15] and graph-based. Researchers in [4, 22] use clustering to find synchronized behaviors. For graph-based models, [27, 28, 30] use spectral methods to analyze the graph.

Different from most problems where users can try various machine learning models easily, fraud detection algorithms are way more complicated. Meanwhile, these algorithms normally do not offer end-to-end interfaces for analysts, which makes the deployment of algorithms much harder. FDHelper is compatible with fraud detection algorithms with fraud groups as outputs. It integrates two built-in state-of-the-art fraud detection algorithms: CrossSpot [11] and D-Spot [29], and allows analysts to change and compare algorithms. Besides, the feature selection process is also quite important. FDHelper solves this problem by automatically loading all the potentially useful features and ranking them accordingly, which allows analysts to change and choose feature sets easily.

Visualizing Fraud Detection

In EVA [14], Leite *et al.* focused more on identifying fraudulent events, e.g., unauthorized transactions from the financial data. In the VISFAN [7] and VISFORFRAUD [6], the authors synthesized numerous reports on currency transactions and suspicious activities/transactions together to build a financial activity network. They proposed a complex network visualization design to display such networks. Beyond the financial fraud detection, the use of visualization also extends to many other domains, including occupational fraud detection [1] and computer network intrusion detection [5].

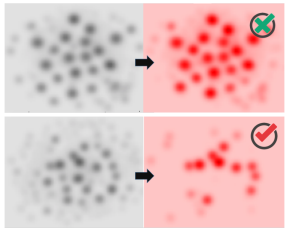
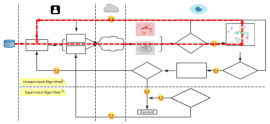
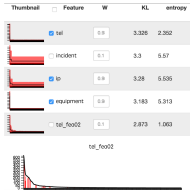
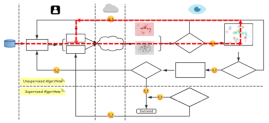
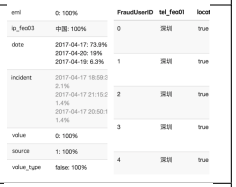
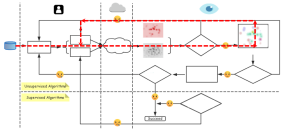
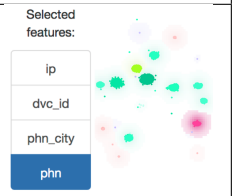
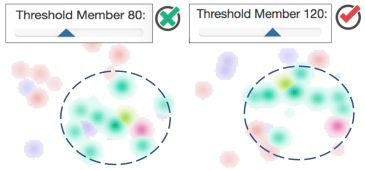
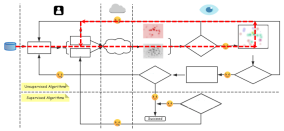
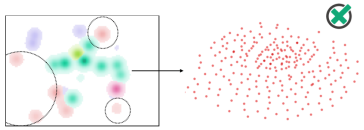
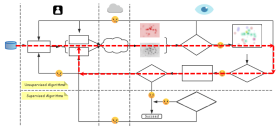
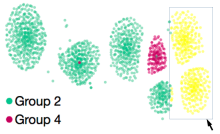
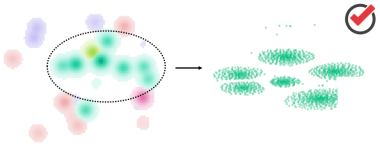
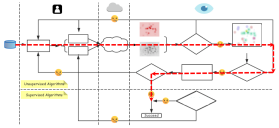
These existing systems either avert the complicated fraud detection algorithms and feature selection process, or use a simple scoring criteria [14] that can not handle the high dimensional data in the real case. Differently, FDHelper utilizes an advanced metric, ColDis, in an end-to-end system and is compatible with different algorithms, and is also scalable to large high-dimensional datasets.

Visualizing Algorithms with Interaction

There has been a long history since visualization was used for facilitating fine-tuning models [16, 17]. Some new works come up to visualize the algorithm adjustment. For example, Google Vizier [9] uses parallel coordinates to analyze searched models. ATMSeer [25] first uses the visualization to assist the automated machine learning process. In terms of Google's FACETS [1], it is similar to our work and has the feature embedding of dataset statistics analysis and visualization. AutoAIViz [26] visualizes the AI model generation process to increase the interpretability. However, none of these systems are designed for fraud detection. Fraud detection problems are different since frauds have unusual synchronized property on a subset of high-dimensional data, and the visualization needs to be flexible on choosing the feature set and to focus on the grouping behavior.

[1] https://pair-code.github.io/facets/

Table 1: Instruction table: How to help yourself out according to the visual patterns in specific situations? (P: Precision / R: Recall)

Situation	Visual Pattern	P/R	Solution	Detailed Action	Related Panel
	Visual Pattern 1: AUL& FUL are similar in both the location and brightness of clusters	↓/↑		Choose the top three features with highest entropies	
	Visual Pattern 2: Too many mixed colors nearby in GL	↓/↑		Delete features with low KL-divergence high entropy or reduce their weights	
	The number of bright clusters is lower than the expected value(e.g. 10)	↑/↓		Keep current features and add some others with low entropy/high KL-divergence	
	Visual Pattern 3.1: Although the overall performance is fine, some local areas still consist of different colors	↓/↑		Increase the *member* or *threshold edge connection threshold*	
	Visual Pattern 3.2: The color of group scattered both in GL and detailed view of this specific group	↓/		If it is common, change feature sets. If few clusters have mixed color check those suspicious groups.	
		↑/		Accept this large group as true positives	

TARGET USERS AND DESIGN PROCEDURE

Target Users. We collaborate closely with a successful fraud detection company, which employs more than sixteen experts. The company has been working with top-tier giants and financial companies on real fraud detection problems for several years already. From our side, we have seven students working on both the algorithm and visualization for fraud detection, and we already published several papers about fraud detection on top conferences. Therefore, we have access to resources from both the academia and industry. During the collaboration, we can not only work with the top-tier experts from the real industry but also their customers. The customers understand frauds perfectly in their scenario and can evaluate the result of machine learning algorithms, but they have little knowledge about the mechanism of algorithms. We regard the experts from the company as a combination of both algorithm and domain experts.

In a word, we have three types of experts evolved: 1) algorithm experts from academia; 2) domain experts from the industry; 3) a team of experts who know both the algorithm and domain from a successful fraud detection company. For simplicity, we categorize them into algorithm experts and domain experts. Mainly domain experts refer to business-side people in the company and their customers.

Design Procedure. We followed the agile development process and went through many iterations. There are three major phases during the process, which are *evaluation metric design*, *display metric design* and *interaction design* phases. Before each phase, we collected experts' answers to carefully designed questions with different focus on evaluation, display or interaction. Then we extracted and refined the information, set up meetings with experts, double-checked the requirements and used the confirmed information to guide the design. During each phase, we developed several prototypes, showed the experts the updated demo every week, modified it according to the advice, and then showed them again.

- Phase 1: design the evaluation metric. We asked *how can we tell the detection result is good or not? What do you usually look at for evaluation? What do you care the most when refining the algorithm?*. After the iterative feedback collection and discussion, we picked the KL-divergence.

- Phase 2: design the display metric. Firstly, we asked the algorithm experts about *which properties should a good fraud detection metric have? What are the state-of-the-art methods researchers are using? What rules do you follow?*. Then we came up with ColDis after many iterations. We also showed other prototypes with other metrics such as Euclidean distance, cosine distance, part of ColDis etc, and showed them to the experts. According to their feedbacks, we decided to use ColDis in Section 5.1.

- Phase 3: customize the interaction. We asked *what are helpful interactions? How many features do you want to see in the system (with options from [0-10] to [all of them])? Which statistical metrics are useful? How fast can you tolerate the system to respond after updating the algorithm? What information do you want to confirm after you fine-tune the algorithm?*. We collected the feedback from both the algorithm experts and domain experts with allowing skips of questions by semi-structured interviews.

We also went through the iterative process for design details, such as the color selection, parameter choice of KDE, projection methods and etc with both algorithm and domain experts. We always picked up the design which gets the most agreements during the discussion.

FraudVis [20] was the preliminary prototype in the early iteration. We gradually came up with metrics and the design of FDHelper after more iterations. We also noted that the feedback did not fully cover what they wanted during our iterative discussion. Therefore, we took notes of their needs, maintained and enriched a requirement list, then concluded them in the Design Requirement Section, which guided the core design of FDHelper.

SYSTEM REQUIREMENTS AND DESIGN

Online Frauds and Grouping Effect

Fraudsters have to reuse resources to lower the cost as we discussed in Section 2.1. The same is true for IP addresses, accounts, and other resources. This similarity is the key to fraud detection, and researchers usually refer to the similarity as "synchronized behavior" [12]. These activities usually are hard to detect since different fraud groups can synchronize on the different sets of features, and there are many features to consider. Unfortunately, synchronized behavior itself does not always indicate a fraud. For example, it is usual for many people to follow President Trump on Twitter, but if many people follow a nobody, it implies a suspicion. We want to capture both *how surprising the similarity is* and *how likely people have such similarities*.

Data Abstraction

The outputs of many unsupervised fraud detection algorithms [11, 29] are groups of fraudulent candidates with synchronized feature values. Intuitively, we illustrate a *group* as a set of detected users who share features. The quality of feature selection highly impacts algorithm performance. Therefore, the input of our visualization system is the algorithm itself and all categorical features. One of the main tasks of our system is to illustrate the unique structure for detected groups, which is also used to evaluate the algorithm performance.

Design Requirements

After the semi-structured interview in Section 3, we summarize their analytical tasks with visualization as below. Besides, we classify all the tasks into two categories: one is observation-based tasks, and the other is action-based tasks.

For **observation-based tasks**, we have:

R1 **Fraud detection overview:** *What is the distribution of frauds in the high-dimensional feature space? How is this distribution similar to or different from the distribution of all users?* Both algorithm and domain experts gain an overview of the quality of the algorithm output.

R2 **Quality evaluation:** *Are the fraud detection results correct? Which group contains more true positives, how is one*

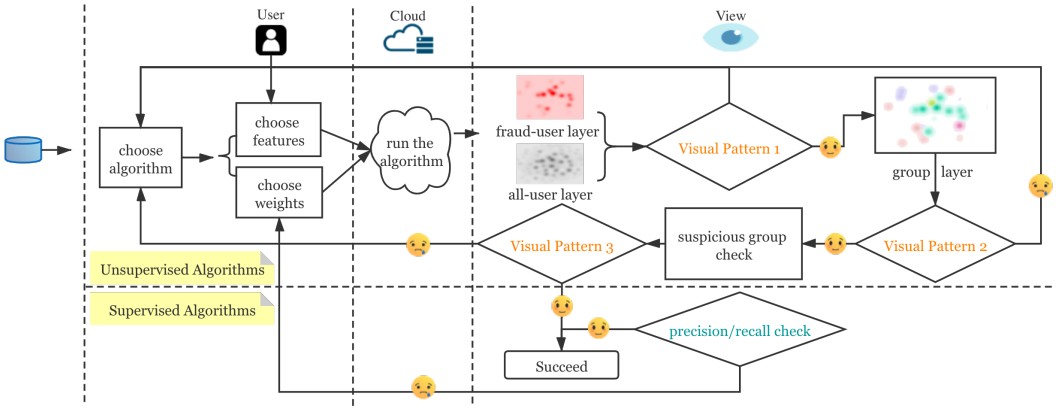


Figure 1: The workflow of FDHelper includes the user configuration part, algorithm in the backend and visualization display.

particular fraud group? Algorithm experts can know better about the quality of the fraud detection output.

R3 **Feature importance analysis:** *Which features contribute the most to the detection result? For one particular fraud group, how important are all the related features?* This task helps algorithm experts select an appropriate feature set for fraud detection.

R4 **Fraud group details:** *How many users are there in each fraud group? What is the distribution of their feature values? What are the feature values of some specific fraud?* Experts can drill down to the details for further inspection.

R5 **Fraud group characterization:** *What common patterns do the users in one fraud group share? How to characterize such pattern for this fraud group?* Domain experts address these questions to form a hypothesis about the collective behavior of the same fraud group. Analysts can validate and learn this hypothesis as domain knowledge.

For **action-based tasks**, we have:

R6 **Interactive feature selection:** *What is the fraud detection result under a given feature set? What will happen if we adjust the weight of features?* Analysts can tune the feature set and weights. Upon applying the new setting, analysts can rerun the algorithm and reevaluate through visualization.

R7 **False positive exclusion:** *Did the algorithm make a mistake on a specific user?* Domain experts marked it as a final process before exporting fraud detection result. FDHelper also helps to carefully examine positives for a better understanding of the algorithm.

FRAUD GROUP PROJECTION

One fraud user can have implicit grouping behavior with some other frauds. Meanwhile, normal users scatter randomly on the feature space. In this section, we propose a novel entropy-based metric ColDis to reveal the characteristics of online frauds.

A Novel Entropy-based Distance Metric: ColDis

A basic distance metric on raw user logs is the Euclidean distance between K-dimensional feature vectors. However, some vital limitations make the Euclidean distance impractical: 1) most features are categorical, using one-hot encoding to compute the Euclidean distance does not make sense; 2) only suspiciously synchronized feature values are useful as discussed in Section 4.1. It is not helpful to discover that 50% of users on the website is female; 3) the combination of one-hot encoding and Euclidean distance leads to an $O(\sum_{k=1}^{K}|V_k|N^2)$ overall complexity in computing the distance matrix among users, which is impractical for a large user base.

We introduce a new distance metric, ***Col**lision **Dis**tance (ColDis)*, which captures the relationship of users on the subset of features relevant to their specific common fraud behavior. The metric is designed to have three desirable properties:

Enhance the rarity and similarity. First, we define the similarity of two users u_i and u_j on the kth feature:

$$\mathscr{S}^k(u_i,u_j)=\sum_{\forall v\in f_k^i\cap f_k^j}-\log(p_k(v)), \tag{1}$$

where p_k denotes the distribution of values on the kth feature and $p_k(v)$ is the probability of taking a value v on this feature. Intuitively, each v in $f_k^i\cap f_k^j$ indicates a collision on the kth feature between u_i and u_j, i.e., having the same value on the kth feature. $-\log(p_k(v))$ corresponds to the Kullback-Leibler Divergence ($KL(\cdot)$) from the all-user distribution on the kth feature ($p_k(\cdot)$) to the distribution of u_i and u_j on the collision value v ($F(\cdot)$). In details, $KL(F||p_k)=-log(p_k(v))$ where $F(x)$ denotes the fixed value distribution of u_i and u_j on the kth feature, $F(x)=1(x=v)$ or $F(x)=0(x\neq v)$.

In some cases, a user can have more than one value/collision on a single feature (e.g., IP subnets used). Eq. (1) sums over all such collisions on the kth feature. The KL divergence also measures the information gain on the kth feature if two users colliding on a value v are detected in the same fraud group. The users with a larger similarity by Eq. (1) will have a better chance to be grouped if we maximize the total information gain on the group structure according to the kth data feature. From another perspective, a higher user similarity also indicates

smaller $p_k(v)$, which suggests more suspicious collisions on rare feature values.

We define the overall similarity between u_i and u_j by the average of their similarities on all K data features.

$$\mathscr{S}(u_i,u_j) = \frac{1}{K}\sum_{k=1}^{K}\mathscr{S}^k(u_i,u_j). \quad (2)$$

Finally, we compute the pairwise ColDis by

$$\mathscr{D}(u_i,u_j) = \begin{cases} \mathscr{S}(u_i,u_j)^{-1} & \mathscr{S}(u_i,u_j) > 0 \\ \mathscr{D}_{max} \times S_{max} & \mathscr{S}(u_i,u_j) = 0 \end{cases}. \quad (3)$$

Magnify the differences between normal users and frauds. The $(\cdot)^{-1}$ operator converts the similarity to a distance metric on the similarity-based affinity graph. $\mathscr{D}_{max}$ denotes the maximal non-zero distance between all pairs of users. S_{max} is a parameter controlling the degree of grouping in the projection. By this metric, the distance between a pair of users is no longer uniformly weighted by all the K features, but biased toward the features having suspicious value collisions between the two users. By projection with ColDis, a fraud group is supposed to be well-separated from other fraud groups and normal users.

Note that ColDis also works for different feature types. For features with numerical value, we apply the similarity definition by Eq. (4), where the closeness between numerical values is used as the degree of value collision.

$$\mathscr{S}^k(u_i,u_j) = -\log(\sum_{\forall f_k^i \le v \le f_k^j} p_k(v)). \quad (4)$$

Unified standard. The ColDis metric helps us to tolerate noise in the raw feature values and focuses on important information that distinguishes normal users from frauds. Given the same feature set of user data, the projection by ColDis will always be the same, allowing us to pre-compute projection results. Different fraud detection algorithms may change the classification result of normal v.s. fraud users, but not the visual layout of their projections. This makes it easier to evaluate different fraud detection algorithms from the ColDis-based visualization, which builds a unified interface for comparison across all these algorithms. We introduce FDHelper over the definition of ColDis, which supports side-by-side comparisons of fraud detection result both among algorithms and within the same algorithm using different feature sets.

OVERVIEW OF FDHelper

Analysts interact with FDHelper in an iterative process. Each iteration contains three steps:

- Given a particular dataset, users choose their desirable features, the corresponding weights and different algorithms in the interactive session;
- Feeding all of these configurations to the algorithm, FDHelper automatically runs the selected algorithm in the backend and prepares the formatted data for the display;
- FDHelper displays the result of the algorithm with ColDis visually in the front-end. With visualization, both domain experts and algorithm experts can evaluate the algorithm performance, and if needed, fine-tune the feature selection and hyperparameters, and iterates through the above steps.

FDHelper Interfaces Overview. Figure 2 illustrates the overall interfaces of FDHelper. We design the system according to the visual fraud detection tasks characterized in Section 4. Both algorithm experts and domain experts start by picking the data set and fraud detection algorithm in the model configuration panel (Figure 2 (a)). Then we display the user data distribution and the detection result from the specified algorithm in the group projection view (R1, the overview task), which includes a multi-layer fraud map for quality evaluation (Figure 2 (b), R2) and a user map for group details (Figure 2 (c), R4). There are three layers in the multi-layer fraud map: the *All-User Layer (AUL)*, *Fraud-User Layer (FUL)* and the *Group Layer (GL)*. Algorithm experts can visually explore the results by selecting important features for fraud detection through the feature view (Figure 2 (d), R3, R6). Domain experts can drill down to the categorical feature distribution and value of the users in the detail view (Figure 2 (e)). The detail view helps to characterize the fraud group behavior (R5), examine potential false positives (R7), and finally extract the learned fraud detection rules.

Connection between fraud patterns, interfaces, and the workflow. To illustrate the connection, Table 1 summarizes each visual pattern and how it relates to the fraud patterns with concrete examples, where we also define the visual patterns.

Visual Patterns for AUL and FUL. After running the algorithm with all features and corresponding positive weights, one observation is that AUL and FUL look quite similar, which indicates that those algorithms give high recall but low precision. It is usually because we use too many features, and the noise confuses the algorithm. Thus, the user needs to improve algorithm precision by removing less important features.

Visual Patterns for GL. If we can get a relatively satisfying result in Visual Pattern 1, we can go to Visual Pattern 2 to check the overall quality of the detected groups. Generally speaking, frauds in the same detected group should have synchronized behavior so if we can keep this property even in the 2D panel, users in the same group should also locate nearby, which indicates a "good" result.

As the most important design in our system, GL can provide us with information on different levels. For suspicious groups in GL, we can project it using the raw feature values and check the quality. A high-quality group appears as a tight cluster. A group with highly overlapping color indicates obvious synchronized behaviors. We accept this group as the correct detection shown in Table 1. Then we can focus on the details of these good groups, such as if there are any outliers. We will introduce the design of the three visual layers in Section 6.1

Multi-Layer Fraud Map Design

The key design of FDHelper is the map in the upper part of the group projection view (Figure 2 (b)). It is composed of multiple layers, including an overview of all online users with

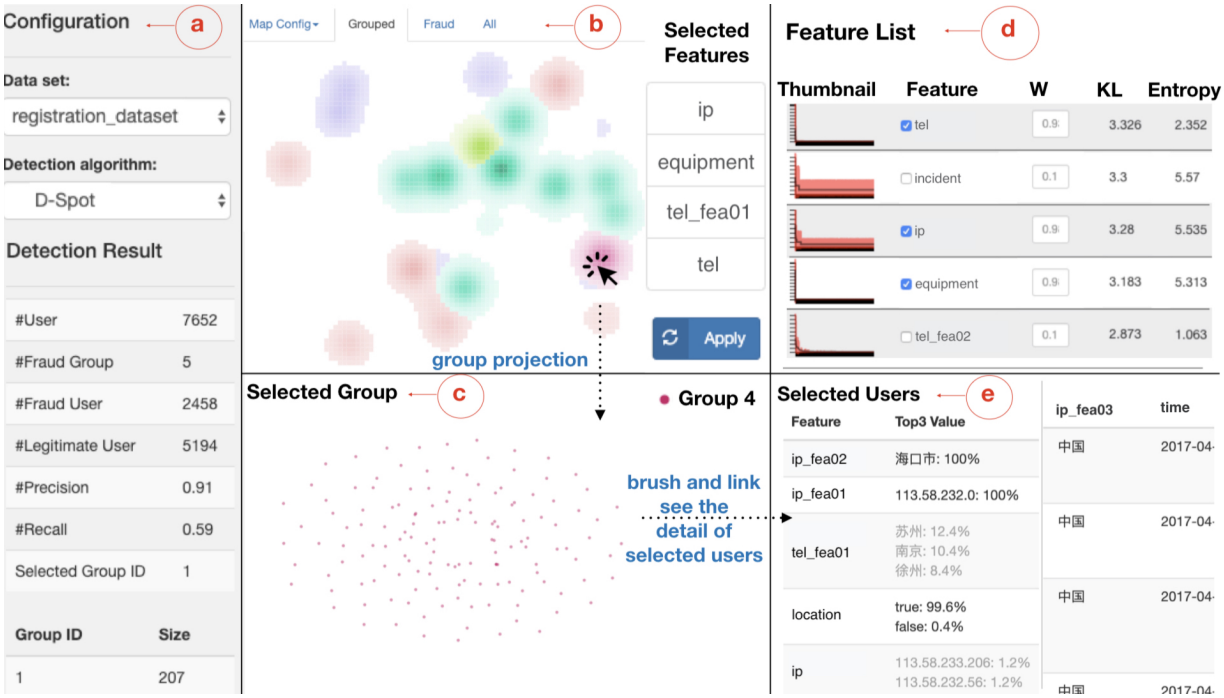


Figure 2: The interfaces of FDHelper. The model configuration panel (a), multi-layer visualization map (b), selected group inspection (c), brush and link selected users for details (e) and adjust the feature sets selection panel (d).

the suspect to potential fraud behavior and a visual interpretation of fraud detection result. In the base layer, we project the high dimensional user profile data into a 2D space using tSNE dimensionality reduction algorithm [23] and the ColDis distance metric.

When the data size gets large, the point-based visualization can lead to severe overlapping, and we can hardly observe the real distribution. Therefore, we apply the Kernel Density Estimation (KDE) method with a Gaussian kernel. We call the resulting metric *User-KDE density* and compute it as:

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sqrt{2\pi}h}e^{-\frac{(x-x_i)^2}{2h^2}}, \tag{5}$$

where N is the number of all the users, x_i is the projected 2D position of the ith user u_i, and h is the bandwidth of KDE initialized by the Gaussian approximation $h = 1.06\sigma N^{-0.2}$, where σ is the standard deviation of all user's positions. As the base layer, we draw the User-KDE density using a black-white color palette. We use a darker color to represent the denser areas. Over the base layer, we design another panel to illustrate the distribution of frauds detected by the algorithm. We compute a Fraud-KDE to represent the distribution of frauds. The coloring of Fraud-KDE is also similar to User-KDE. We fill the denser fraud area with lighter colors, but we use the red color hue to highlight the existence of frauds.

At the top of the fraud map, the group layer further displays the distribution of fraud-user groups. First, we define the *Per-Group-KDE density* that represents the distribution of each fraud group. In the group layer, the color hue indicates the dominating group and the color lightness indicates the Group-KDE density, lighter means that there are only a few members of this group at this location.

On the multi-layer map, we also display the relationship between features and detected frauds. When analysts select a feature from the current selected feature list, we color detected frauds who have the same value for the selected feature with the same hue. Switching from one feature label to another will also help identify the importance of different features. The more synchronized the color appears in the cliques, the more important the corresponding feature is. It helps to improve the feature selection in the fraud detection algorithms (R3).

Interactive Feature Selection

As we mentioned before, the quality of feature selection has a great impact on the algorithm performance. Different features' combinations result in different detection accuracies.

On the right-hand side of *group projection view* (Figure 2 (b)), we list all features in a feature view. In Figure 2 (d), the selected features are arranged in top lines and other unselected ones are attached to the bottom. Each feature line consists of a thumbnail chart on the left and the statistics on the right. The thumbnail chart integrates two lines: black line shows the value distribution of this feature in the entire user data set, and the red line shows the value distribution of all the detected frauds. By comparing these two distributions, algorithm experts can obtain an initial idea about the importance of this feature in separating the frauds (R3). Meanwhile, we compute two information-theoretic metrics, including the average *entropy* of this feature across all fraud groups, and the average *KL divergence* of the value distribution on this feature from all users to each fraud group. The features with lower entropy

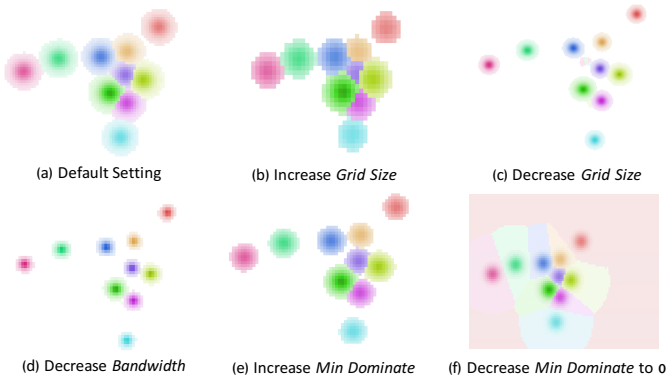


Figure 3: Screenshots of tuning visual design parameters

and higher KL divergence will be more important to currently detected fraud groups. We display these metrics, together with the feature weights as the *feature statistics*.

The *feature view* (Figure 2 (d)) offers several interactions to facilitate the feature selection process. Upon clicking a feature, the thumbnail chart expands to a full feature chart. The full feature chart emphasizes the fraud feature distribution as bar charts. When selecting a fraud group in the group layer, we highlight the feature value distribution of this group on the bar charts. As the number of user features can be quite high, we allow analysts to sort them according to any feature statistics. Algorithm experts interactively select important features and adjust their weights to refine the fraud detection algorithm. By clicking the "Apply" button, we update the parameters and rerun the backend computation. Finally, we display the visualization result as the start of the next iteration of this visual analysis.

There are two kinds of configurable hyperparameters:

Visual design related parameters. As Figure 3 shows, *Bandwidth* is a smoothing parameter of Kernel Density Estimation (KDE) algorithm. *Min Dominate* filters out the grids that are not dominated by any group. *Grid size* controls granularity of density estimation. Intuitively, if a color spreads the entire map, its corresponding group will have a big impact. Increasing *Min Dominate* or decreasing *Bandwidth* can optimize the visual effect under this situation. However, these visual parameters only affect the visual design, but not provide insights about the performance of the algorithm itself.

Min Dominate filters out the grids that are not dominated by any group, which makes the map more clear and meaningful. In the extreme case, when we set it to zero, we get a GL as Figure 3 (f), which provides few insights. The *bandwidth* is a parameter of KDE that exhibits a strong influence on the resulting estimation. Decreasing the *bandwidth* (Figure 3 (d)) and increasing the *min dominate* value (Figure 3 (e)) can both separate the cliques further. They provide a more clear view of the given result but are not scalable to large datasets with many groups. In most cases, we can use the default *grid size* to balance the efficiency and the visual effect.

Algorithm-related parameters. We extract parameters which are most often used so that users can easily access and adjust them. *Member threshold* filters out small groups and thus helps users to focus on the bigger groups. *Edge threshold* filters out loosely connected users. A weak connection implies users in the dataset are not closely synchronized and have less in common. Increasing *member threshold* and *edge threshold* reduces false positives, but a large value also lowers the recall. We present our heuristic of adjusting these parameters in Table 1 (row 4) and in the next section.

EVALUATION

Case Study on E-commerce Website

First, we applied FDHelper onto the e-commercial website. As we mentioned above, in this case, we find the number of fraudsters in the registration process is even greater than the normal users according to the experience of domain experts, which we call "fraudulent registration".

Dataset. We visualize eight thousand registration records in this case, to demonstrate FDHelper and analyze frauds. We have only partially labeled data from our collaborators. They get the label according to both their experience and the subsequent shopping behavior of these users. The labels are accurate but not adequate. In other words, the precision of the labels is 100% while the recall is not.

Tuning process. In the beginning, we use all categorical features with equal weights. Unsurprisingly, the algorithm classifies all users as frauds and thus AUL and FUL look pretty similar, which indicates a high recall but low precision, and it categorizes all the users as frauds. Other case studies also start from this default setting, and we take the same actions according to Table 1 (the first row). Firstly, we sort the features by their entropies in descending order. The bigger entropy it has, the more information the feature can provide.

1) We roughly categorize the top ten potentially useful features into three categories: time-related features, IP-related features, and phone-related features.

2) Given any of these three feature combinations, we can easily get AUL and FUL like Figure 4 (a) and (b), which indicates we can move forward.

3) Figure 4 (c) shows the result for time-related features. Following the Table 1, we find the timestamp of user registration is the most important one here. The same works for Figure 4 (d) and (e) which show the result for IP-related features and phone-related features respectively. We also get the prefixes of IP addresses is the best feature in this category and the two phone-related features are equally representative.

4) When we drilled down the detailed feature values in all of these results, we surprisingly found the *device* used by any given group highly synchronized. It also shows a low entropy and high KL-divergence. Next, we keep the most useful features and adjust weights proportionally while we identify the device feature as the most important one.

5) After applying these features, we get a promising result. However, there are still mixed up groups, e.g. the fourth row of Table 1. We then increase the *edge threshold*. It results a better result that we show in Figure 4 (f). Also, since we

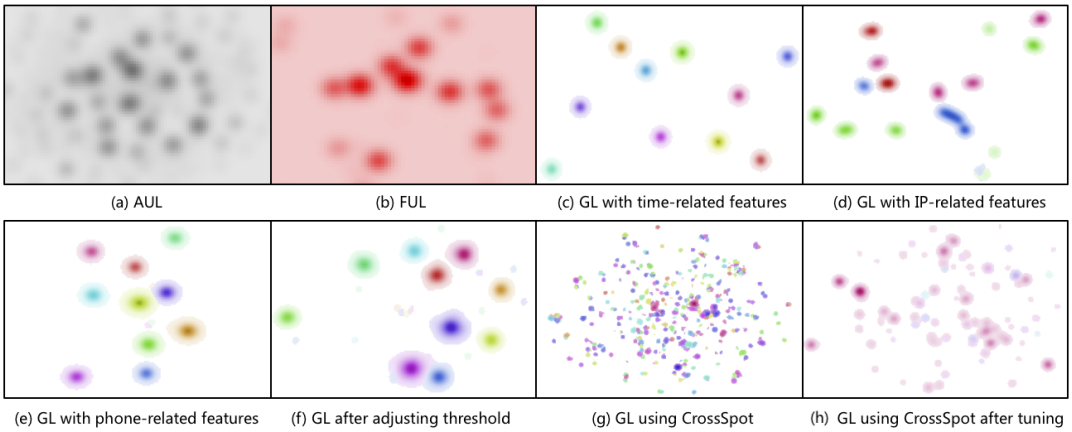


Figure 4: Screenshots for tuning algorithms on the e-commercial scenario.

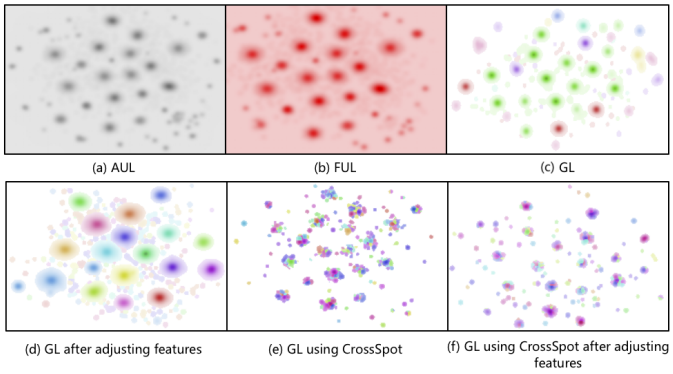


Figure 5: Screenshots for tuning algorithms on the video case.

have labels in this case to evaluate our result, we showed the performance with a precision of 1.0 and a recall of 0.6 in the performance panel. We gave feedback to the domain experts, and they consider this result promising given the small false positive, which is described in Section 4.

Using CrossSpot. Additionally, we also apply CrossSpot to FDHelper. Ban et al. [29] claim that D-Spot outperforms CrossSpot in detecting frauds effectively, which is also applicable in our case. Normally, we have the performance of CrossSpot as Figure 4 (g), which shows a poor performance with chaotic colors mixed in each clique. Following the instruction we proposed, we can surprisingly get a better performance shown as Figure 4 (h) with a precision of 0.81 and a recall of 0.21. Even our algorithm experts never expect such a good performance from CrossSpot. After the discussion, we reach a consensus that CrossSpot only perform well when we omit IP-related features. It is probably the noise in IP-related features destroys the locality for CrossSpot's search.

Case Study on Social-video Website

Dataset. In order to detect frauds in social networks, we get a sample of clickstream records from an online social video website. We filtered out less important non-interactive records, such as registers, logins, and logouts, and focus only on user interaction records, such as follow, like and send gifts. Different from the last e-commerce case, here we do not have any labels to evaluate the performance.

Tuning process. Like what we did before in the e-commercial case in Section 7.1, applying all the features results in poor performance, while also provides insight on distributions of all features on the feature statistical panel. After sorting these features in the descending order of their entropies, we get potentially useful features – timestamp, the target user, the source user, the IP address and the type of an event. With the procedure, we can get AUL and FUL views as (a) and (b) in Figure 5, which means we can move forward on the GL exploration. After getting the view shown as (c), we think the feature sets are applicable since they can distinguish the frauds from the normal users, and the color is not very mixed except for only one large group. Therefore, we need to adjust the weights of these values. Finally, we choose the timestamp with the lowest entropy value as the most important feature and adjust the weights of others proportionally. We get the improved result as (d) with a less chaotic mixture of colors in big cliques. It also detects more purely colored groups, which indicates a better result.

Using CrossSpot. We also test CrossSpot in our system. Changing from the original five features to four hand-picked features derived from the experiment of D-Spot, it also gets an obvious improvement from (e) to (f). Checking the mixture of colors, drilling down the details of groups and the feature statistic panel can all prove this. However, it still fails to match D-Spot's performance even with our best efforts, limited by the inferior algorithm design.

EXPERT INTERVIEW

We held regular discussions with two fraud detection algorithm experts, E1 and E2, throughout FDHelper development that lasted six months. We have weekly discussions to collect feedbacks and refine FDHelper's design iteratively. For both registration and video cases, experts were actively involved in choosing the subset of datasets and fraud detection algorithms to deploy, testing the algorithm performance, designing interfaces for connecting algorithms and evaluating FDHelper. We also held discussions with two domain experts, D1 and D2.

Feature Selection in FDHelper. FDHelper can help algorithm experts better understand the contribution of different features to the detection result as we discussed in Section 6.2.

E1 commented that being able to select important features timely helps her a lot to eliminate her pain in selecting proper features. She also commented on the statistical information *entropy* and *KL-divergence* calculated from the current detected groups. She always looks at the two metrics, and FDHelper makes things a lot easier. E2 commented switching feature labels is also useful and has a strong visual impact. E2 said that "*Just by switching those labels, I know which features are more synchronized. I normally only look at this and use those two statistical metrics for verification.*"

Human-Computer Interaction in Fraud Detection. All experts appreciate the interactions introduced in FDHelper. They believed the interaction can simplify and improve a fraud detection process for analysts to find the best algorithm. They commented that "*people with little algorithm knowledge can now easily play with the system visualization panel and feature selection panel, and identify some surprisingly useful features, and even get their own fraud detection result!*"

Diagnose the Detection Result. Last but not least, it is a surprise that algorithm experts and domain experts love FDHelper when it comes to diagnosing the result. D1 commented that "*With brushing the users in one group, I can use my domain knowledge to exclude some less suspicious users. We can't stand the outcome of losing good users.* "E1 commented that "*I can clearly know which group needs a future investigation from the group layer. I have more confidence about algorithm performance after using FDHelper.*" E2 also commented that "*Although D-Spot outperforms CrossSpot, I still find some outliers in good groups in FDHelper, maybe we will add some rules to D-Spot to fix this problem.*"

USER STUDY

After knowing how experts like FDHelper, we also want to know *how can FDHelper help analysts new to the field to get started on fraud detection? Can they start fine-tune the model after a brief introduction and exploration?*

Participants and Apparatus. We recruited five Ph.D. students and three master students (six females and two males, age 23-29, $\mu = 25$, $\theta^2 = 3.5$) with diverse backgrounds including computer science, statistics, biology and etc, denoted as P1-P8. All of them have experience in data science but do not have experience of fraud detection. We conduct the experiments with eight 13-inch MacBook Pros. We use the `registration_dataset` in Section 7.1 for evaluation.

Tasks and Procedure. The participants are asked to perform two tasks and fill out a 17-question survey. Among them, twelve questions Q1-Q12 are objective, while Q13-Q16 are subjective questions. The questions are designed to investigate how FDHelper helps from the *visualization wise, feature wise, hyper-parameter wise, and algorithm wise*.

The procedure consists of three major steps: 1) We give a quick tutorial about FDHelper to all of our volunteers. Volunteers know more about each component of FDHelper and how to use in this process. 2) We showed both a live demo and a recorded demo for volunteers. 3) Then we let volunteers explore FDHelper on their wills, and we record their clickstreams during the process.

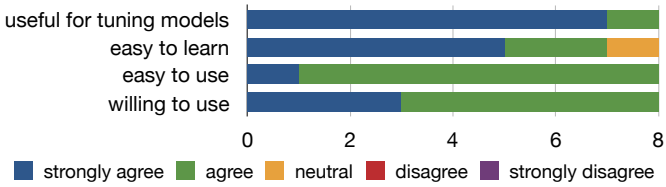


Figure 6: Results of usability questions

Our anonymous questionnaires are accessible online, and every volunteer can only submit once. From the clickstream records, we find that volunteers usually spend 13-30 minutes on exploring FDHelper with 84.8% clicks for the multi-layer visual map, 5.5% for the feature panel and 9.7% for the detail inspection. They run 2-4 rounds of FDHelper to finish the survey. It indicates that the feature panel and the multi-layer map provides more information for analysts.

We collected all eight surveys we sent out, and seven of volunteers correctly answered all twelve objective questions. Figure 6 shows the result of usability questions. Although one volunteer thinks FDHelper is not easy to learn, all of them agree that FDHelper is useful for tuning models and easy to learn. The majority of them strongly believe they will use it for future fraud detection tasks. In future work, we will find more volunteers to use and provide feedback about FDHelper.

CONCLUSION

In this work, we presented FDHelper, an interactive visualization tool that supports fraud detection algorithm experts to fine-tune the weights and features timely. The workflow of using FDHelper was proposed based on the deep understanding of the requirements of both algorithm experts and domain experts. The three key designs in the workflow - choosing the algorithm and dataset, refining the feature selection and algorithm setting, and evaluating the detection result - are identified to guide the implementation of FDHelper. It brings the controllability, readability, and dependability to visual fraud detection. We next propose a multi-granularity three-layer visualization map with in-situ configuration to enable users to refine and check a fraud detection progress in time from the algorithm level, feature level, and the hyperparameter level. FDHelper works for fine-tuning both supervised algorithms and unsupervised algorithms. We also prove the efficiency through two real-world datasets and two state-of-the-art algorithms respectively. To the best of our knowledge, our work is the first interactive visual fraud detection system based on the grouping behavior of online fraud users. We hope this work can inspire both algorithm experts and visualization researchers on the subject of interactive fraud detection.

ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China (NSFC) Grant 61532001 and 61772504, Tsinghua Initiative Research Program Grant 20151080475, Fundamental Research Funds for the Central Universities and gift funds from Huawei, Ant Financial and Nanjing Turing AI Institute.

REFERENCES

[1] Evmorfia N Argyriou, Aikaterini A Sotiraki, and Antonios Symvonis. 2013. Occupational fraud detection through visualization. In *IEEE International Conference on Intelligence and Security Informatics*. 4–6.

[2] Richard J. Bolton and David J. Hand. 2002. Statistical Fraud Detection: A Review. *Statist. Sci.* 17, 3 (2002), 235–249.

[3] Nan Cao, Conglei Shi, Sabrina Lin, Jie Lu, Yu-Ru Lin, and Ching-Yung Lin. 2016. Targetvue: Visual analysis of anomalous user behaviors in online communication systems. *IEEE TVCG* 22, 1 (2016), 280–289.

[4] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 477–488.

[5] Emilio Corchado and Álvaro Herrero. 2011. Neural visualization of network traffic data for intrusion detection. *Applied Soft Computing* 11, 2 (2011), 2042–2056.

[6] Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Pietro Palladino. 2010. Visual analysis of financial crimes. In *Proceedings of the International Conference on Advanced Visual Interfaces*. 393–394.

[7] Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Pietro Palladino. 2011. An advanced network visualization system for financial crime detection. In *IEEE Pacific Visualization Symposium*. 203–210.

[8] Tom Fawcett and Foster Provost. 1997. Adaptive fraud detection. *Data mining and knowledge discovery* 1, 3 (1997), 291–316.

[9] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1487–1495. DOI:**http://dx.doi.org/10.1145/3097983.3098043**

[10] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Günnemann, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. *BIRDNEST: Bayesian Inference for Ratings-Fraud Detection*. 495–503.

[11] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. 2016. Spotting Suspicious Behaviors in Multimodal Data: A General Metric and Algorithms. *IEEE TKDE* 28, 8 (2016), 2187–2200.

[12] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. CatchSync: catching synchronized behavior in large directed graphs. (2014), 941–950.

[13] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and V. S. Subrahmanian. 2017. FairJudge: Trustworthy User Prediction in Rating Platforms. *CoRR* (2017).

[14] Roger A Leite, Theresia Gschwandtner, Silvia Miksch, Simone Kriglstein, Margit Pohl, Erich Gstrein, and Johannes Kuntner. 2018. EVA: Visual Analytics to Identify Fraudulent Events. *IEEE transactions on visualization and computer graphics* 24, 1 (2018), 330–339.

[15] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. 2018. xStream: Outlier Detection in Feature-Evolving Data Streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, UK*. 19–23.

[16] Joe Marks, Paul Beardsley, Brad Andalman, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, and others. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH*. Association for Computing Machinery.

[17] Sean McGregor, Hailey Buckingham, Thomas G Dietterich, Rachel Houtman, Claire Montgomery, and Ronald Metoyer. 2015. Facilitating testing and debugging of Markov Decision Processes with interactive visualization. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 53–61.

[18] Animesh Patcha and Jung-Min Park. 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks* 51, 12 (2007), 3448–3470.

[19] Neil Shah, Alex Beutel, Bryan Hooi, Leman Akoglu, Stephan Gunnemann, Disha Makhija, Mohit Kumar, and Christos Faloutsos. 2017. EdgeCentric: Anomaly Detection in Edge-Attributed Networks. In *IEEE ICDM*. 327–334.

[20] Jiao Sun, Qixin Zhu, Zhifei Liu, Xin Liu, Jihae Lee, Zhigang Su, Lei Shi, Ling Huang, and Wei Xu. 2018. FraudVis: Understanding Unsupervised Fraud Detection Algorithms. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 170–174.

[21] Kurt Thomas, Danny Yuxing, Huang David, Wang Elie, Bursztein Chris Grier, Thomas J Holt, Christopher Kruegel, Damon McCoy, Stefan Savage, and Giovanni Vigna. 2015. Framing dependencies introduced by underground commoditization. In *In Proceedings (online) WEIS*. Citeseer.

[22] Tian Tian, Tong Zhang, Tong Zhang, Tong Zhang, and Tong Zhang. 2015. Crowd Fraud Detection in Internet Advertising. In *International Conference on World Wide Web*. 1100–1110.

[23] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9 (2008), 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html

[24] Gang Wang, Xinyi Zhang, Shiliang Tang, Haitao Zheng, and Ben Y Zhao. 2016. Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 225–236.

[25] Qianwen Wang, Yao Ming, Zhihua Jin, Qiaomu Shen, Dongyu Liu, Micah J. Smith, Kalyan Veeramachaneni, and Huamin Qu. 2019. ATMSeer: Increasing Transparency and Controllability in Automated Machine Learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 681, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300911

[26] Daniel Karl I. Weidele, Justin D. Weisz, Eno Oduor, Michael Muller, Josh Andres, Alexander G. Gray, and Dakuo Wang. 2019. AutoAIViz: Opening the Blackbox of Automated Artificial Intelligence with Conditional Parallel Coordinates.

[27] Liang Xiong, Barnabás Póczos, and Jeff Schneider. 2012. Group Anomaly Detection using Flexible Genre Models. *Advances in Neural Information Processing Systems* (2012), 1071–1079.

[28] Liang Xiong, Barnabás Póczos, Jeff G. Schneider, Andrew Connolly, and Jake Vanderplas. 2011. Hierarchical Probabilistic Models for Group Anomaly Detection. *Journal of Machine Learning Research* 15 (2011), 789–797.

[29] Ban Yikun, Liu Xin, Huang Ling, Duan Yitao, Liu Xue, and Xu Wei. 2019. No Place to Hide: Catching Fraudulent Entities in Tensors. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 83–93. DOI: http://dx.doi.org/10.1145/3308558.3313403

[30] Rose Yu, Xinran He, and Yan Liu. 2015. GLAD:Group Anomaly Detection in Social Media Analysis. *Acm Transactions on Knowledge Discovery from Data* 10, 2 (2015), 1–22.