Do Not Pull My Data for Resale: Protecting Data Providers Using Data Retrieval Pattern Analysis

Guosai Wang[†], Shiyang Xiang[*], Yitao Duan[‡], Ling Huang[*], Wei Xu[†]

[†]Tsinghua University, Beijing, China
[*]AHI Fin-tech Inc., Beijing, China
[‡]Netease Youdao Inc., Beijing, China
wgs14@mails.tsinghua.edu.cn,shiyang@fintec.ai,duan@rd.netease.com
linghuang@fintec.ai,weixu@tsinghua.edu.cn

ABSTRACT

Data providers have a profound contribution to many fields such as finance, economy, and academia by serving people with both web-based and API-based query service of specialized data. Among the data users, there are data resellers who abuse the query APIs to retrieve and resell the data to make a profit, which harms the data provider's interests and causes copyright infringement. In this work, we define the "anti-data-reselling" problem and propose a new systematic method that combines feature engineering and machine learning models to provide a solution. We apply our method to a real query log of over 9,000 users with limited labels provided by a large financial data provider and get reasonable results, insightful observations, and real deployments.

ACM Reference format:
Guosai Wang, Shiyang Xiang, Yitao Duan, Ling Huang, Wei Xu. 2018. Do Not Pull My Data for Resale: Protecting Data Providers Using Data Retrieval Pattern Analysis. In *Proceedings of The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, Ann Arbor, MI USA, July 2018 (SIGIR'18),* 4 pages.
https://doi.org/https://doi.org/10.1145/3209978.3210158

1 INTRODUCTION

Data providers (aka. data vendors) collect and index both public and proprietary data source into a searchable database and serve people in various areas such as finance [2, 7, 16] and academia [1, 5]. The key value-add for data providers is data integration, cleaning, updating and offering the structured query interface. For example, Bloomberg [2] users can query real-time financial and economic data by using manual commands or scripting-friendly APIs.

It is important for the data providers to protect their data by only allowing users who commit to the *fair use* of the data. While providers have different definitions of fair use, almost everyone agrees that *data reselling* is not acceptable use. In general, *data resellers* (DRs) pull loads of data through the query API from data providers, and resell them, or offer their own data service. Such data reselling activities can cause loss of revenue to the data provider and potential copyright infringement to the data source. Thus it is necessary for the data providers to identify DRs and take countermeasures against them.

Identifying data resellers, which we call the *anti-data-reselling* (ADR) problem, is different from the anti-web-crawling (AWC) problem. Existing AWC work analyzes the web access patterns to distinguish automatic web crawlers from humans [4, 6, 9, 13, 15, 17, 18] or to detect malicious crawlers for security issues (e.g., defending against DDoS attacks) [14, 19].

However, these techniques do not help solve ADR problem. A key goal for AWC is to distinguish bots from human, while many data providers offer API-based queries to support scripting as a feature, allowing different kinds of bots. Thus, the goal is to distinguish a specific type of abnormal data retrieval behavior, rather than whether the task is done programmatically. In some sense, for ADR, we need to reason about "why the data is taken" rather than "how the data is taken", making it a hard problem. Technically, AWC often relies on analyzing very short-term behaviors to identify a bot, while ADR focuses on much longer-term patterns.

It is nontrivial to precisely identify data resellers. People often use simple rules, such as limiting the query volume. However, modern applications may need lots of data. For example, some automated trading applications use lots of queries to track the price of a set of stocks. If the user paid for the query volume, setting a quota hurts the user experience. Thus, we need more sophisticated features and model to distinguish DRs from regular heavy data users. As an additional challenge, there are not many labeled DRs for model training, and thus we have to leverage unsupervised learning techniques as much as possible.

While people can write rules to capture DR's patterns, DRs can change their patterns to avoid the detection. We design our systems based on a set of fundamental behavior patterns that the DRs cannot change easily, and we have the following three key insights about DRs' behavior patterns:

1) Volume. While heavy data users are not necessarily DRs, the other way around is usually true (they may not be the heaviest users, but still heavy). This is because someone retrieving only a small data volume cannot cause much damage anyways.

SIGIR'18, July 2018, Ann Arbor, MI USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/https://doi.org/10.1145/3209978.3210158

2) Spread. DRs need to retrieve a large variety of data items, instead of getting a small set over and over, as legitimate users do. This is because normal users often focus more on certain keys (e.g., a subset of stocks, or a specific academic discipline), while data resellers need more data variety for reselling. Note that spread by itself does not necessarily lead to DR suspicion. For example, legitimate users may conduct a survey that requires retrieving a variety of keys.

3) Periodicity. DRs need to query data periodically to update the database they are reselling, and thus they will send queries periodically over time.

Based on these three insights, we propose a systematic method combining feature engineering and supervised/unsupervised learning techniques to solve the ADR problem. Our basic idea includes: 1) *Feature creation.* Based on the three characteristics above, we create some features covering user behavior patterns including query volume, distribution, time, periodicity and burstiness. The goal of this step is to cover the three characteristics from as many aspects as possible and as redundant as possible, to make it difficult for DRs to avoid; 2) *feature selection and reweighting.* The key problem is that there is no precise definition of the DR behavior, even from the data providers. We take a learn-by-example approach and automatically select a subset of features based on the few observed DR samples. The goal here is to "learn" the data provider's definitions of DR behavior. 3) *DR identification.* Using the selected features above, we use unsupervised outlier detection algorithms (and supervised classification, if we have some labels) to identify DRs. We report the detected DRs to the data provider, and it decides what measures to take against each of them.

Without ground truth labels, we cannot quantitatively evaluate the detection result. Instead, we focus on getting *interpretable* results and insights and getting feedbacks from data security experts at the data provider.

We perform experiments on a real-world query log containing 9,000+ users' queries from a large-scale financial data provider. There are very limited data labels. Our method can identify a number of DRs, achieving much better accuracy than the naive methods. Security experts at the data provider have confirmed our detection results and our system has been deployed at the data provider.

We define ADR problem and focus on solving a general framework. In summary, our major contributions are:

- We define the anti-data-reselling (ADR) problem, identify the three key characteristics behaviors of DRs and propose expressive features and a systematic method to identify them;
- We apply our methods to a real query log and provide insightful detection results.

2 DATASET AND METHODS

Dataset. In this study, we use query log **D** from a large financial data provider with tens of thousands of paying subscribers. This data provider maintains information on stocks, bonds, foreign exchange, economic indices and so on. The data are organized into five databases ($\mathbf{D}_A$ to $\mathbf{D}_E$), and each DB contains many data items retrievable by keys. For brevity, we use *keys* to refer to these data items. The log **D** contains one-month of query history. Table 1 shows basic statistics of **D**. For confidentiality, we normalize

Table 1: Dataset overview. The numbers are normalized to the number of users of $\mathbf{D}_E$.

DB id	# queried indices	# users	Total query volume
$\mathbf{D}_A$	1×10^3	14	6×10^3
$\mathbf{D}_B$	2×10^2	17	4×10^3
$\mathbf{D}_C$	9×10^2	16	1×10^4
$\mathbf{D}_D$	6×10^2	2	4×10^4
$\mathbf{D}_E$	5×10^2	1	2×10^4

Table 2: Features of Each User's Profile. X is the time granularity that can be second, minute, hour, or day. A user is active on a day if she has at least one query.

Feature	Definition
index_day_entropy	See text
index_num_entropy	See text
index_avg_entropy	See text
total_num	Total query volume
total_indices	Total number of unique queried data indices
sum_day_indices	Sum of each day's number of unique queried data indices
avg_day_indices	sum_day_indices/number of active days
day_set_ratio	avg_day_indices/total_num
avg_7nday_indices	Total number of unique queried data indices for every consecutive 7 days
avg_7day_indices	Total number of unique queried data indices for every consecutive 7 active days
X_num	Total number of active Xs
X_num_rank	Percentile of X_num
X_rate	total_num / X_num
X_rate_rank	Percentile of X_rate
normal_num	Total query volume occurred during 8 am-10 pm
abnormal_num	total_num - normal_num
abnormal_ratio	abnormal_num / total_num

all numbers. While the log records contain many fields, in this work, we only use the following self-explanatory fields: account_id, query_time, queried_DB and queried_indices.

Security experts at the data provider have labeled about 1/6 of the users in $\mathbf{D}_A$, and we call the labeled subset $\mathbf{D}_{A(L)}$. Within $\mathbf{D}_{A(L)}$, 3% are labeled as DRs, and the rest 97% are legitimate. All other users are unlabeled.

We analyze each DB separately, as both legitimate users and labeled DRs access each DB independently. As we have introduced in Section 1, we follow a three-step method: *feature creation*, *feature selection and reweighting*, and *DR identification*.

2.1 Step 1: Feature creation

Based on the three insights in Section 1, we design 29 features to capture each user's query history on each DB. Table 2 provides an overview. There are three categories of features:

1) Entropy features to capture the uniformity of access patterns. We use entropy to capture the "uniformity" of a certain distribution, leading to the first three features in Table 2. Formally, given a set X with n discrete probabilities p_i such that $\sum_i p_i = 1$, its *entropy* is $H(X) = -\sum_i p_i \log p_i$. Given a user u, DB T with m keys, and T_i as the i-th key in T, we count number of days on which u retrieves T_i at least once, as well as the total number of u's retrieval on T_i for the month, and we denote them as q_i and w_i, respectively. Then we define the entropy of these two counts:

$$\text{index\_day\_entropy} = H(\{q_i / \sum_i q_i\})(i = 1, 2, \cdots, m) \quad (1)$$

$$\text{index\_num\_entropy} = H(\{w_i / \sum_i w_i\})(i = 1, 2, \cdots, m) \quad (2)$$

Additionally, we vary index_day_entropy to create index_avg _entropy as the entropy of the seven-active-day-moving-average of the number of unique keys retrieved.

Intuitively, the entropy features capture the uniformity of a user's queries over different data indices and a user's query periodicity. Legitimate users have a bias on the keys they retrieve, and usually access a different number of keys in different time periods. However, DRs usually retrieve many more different keys and exhibit stronger periodicity, resulting much higher entropy than legitimate users.

2) Retrieval volume features to capture the activity. The second group of 23 features in Table 2 are designed to capture access-count-related features, such as volume, diversity, activeness and temporal density of a user's queries.

3) Features about the time-of-the-day for queries. The last three features (from normal_num, abnormal_num and abnormal_ratio) are designed to capture whether a user queries at night, which makes him more suspicious.

Finally, we assemble all features into a vector $v_i^T = \langle f_0(h_i), f_1(h_i), \cdots, f_{28}(h_i) \rangle$. where f_k is the k-th feature, and h_i is the query history of u_i on DB $\mathbf{D}_T$ (where $\mathbf{D}_T \in \mathbf{D}_A, \ldots, \mathbf{D}_E$). We create one such vector for each user on each DB, and we call v_i^T the *profile* of u_i on $\mathbf{D}_T$.

2.2 Step 2: Feature selection and reweighting

Redundancy in the designed features makes it harder for DRs to avoid a feature but affects detection by introducing noisy ones. We leverage the labeled subset $\mathbf{D}_{A(L)}$ to select useful features. In other words, we try to learn which features are important for the security experts.

We use L1-penalized logistic regression (LR), a common feature selection technique to learn a weight for each feature, and we only use the features with nonzero weights. Without losing generality, we denote the selected n' features by $f_0, \cdots f_{n'-1} (n' \leq 29)$.

While LR helps us to pick some features initially, we further adjust the weight of each feature using the procedures discussed in [8]: We run the *Random Forest* (RF) algorithm [3] on all labeled users in $\mathbf{D}_{A(L)}$ to train a classifier G. During the training process, we also learn relative importance of the features according to their depth in the learned decision trees. Assuming the importance of the n' features are $c_0, \cdots, c_{n'-1}$, we rescale each user profile vector v_i^T to $\overline{v}_i^T = \langle c_0^k \cdot f_0(h_i), \cdots, c_{n'-1}^k \cdot f_{n'-1}(h_i) \rangle$, where k is a scaling parameter with a default value of 0.5.

We use labeled data to train the label weights instead of using the trained classifier directly on other DBs. This is because the feature distributions for different DBs vary, and we believe DRs' key behavior patterns are more stable. In other words, from the labels, we learn the "rules" about which features are important to look at rather than a data-dependent detection model.

Table 3: Feature importance estimation.

Selected feature	Importance
index_day_entropy	**0.375**
total_indices	**0.245**
index_avg_entropy	**0.114**
minutes_rate_rank	0.069
sum_day_indices	0.058
seconds_rate_rank	0.052
days_num	0.035
abnormal_ratio	0.032
minutes_num_rank	0.021

2.3 Step 3: Data reseller identification

We assume that the vast majority of the users are legitimate and behave differently from DRs in our feature space, and thus we can model DRs as outliers in the data. We evaluated different outlier detection algorithms, and find density-based algorithms and those insensitive to dimension rescaling (e.g., Isolation Forest [10]) both perform poorly on our dataset. Thus, we adopt *one-class SVM* [11], a nearest-neighbor-based algorithm that takes advantage of dimension rescaling.

For $\mathbf{D}_A$, we do not want to waste its precious labels, and thus we compliment the outlier detection results with the classifier G's prediction result on the unlabeled data by taking the union of both the results.

Many users are detected as outliers because they do not have many activities. We take a simple post-processing step to filter out these low-activity outliers using threshold-filter on the three features days_num, total_num and index_day_entropy. We set the threshold at the median of all users. The accuracy of the threshold does not affect results much, as the activity level for DRs is much higher than the median anyway.

3 RESULTS

We implement our method using `scikit-learn` [12]. Due to limited space, we only report the best set of hyperparameters we find. We set $C = 15$ for L1-penalized LR, *n_estimators* $= 64$ for the random forest, *kernel* $=$ rbf, *gamma* $= 0.1$, and *nu* $= 0.004$ for the one-class SVM. The outlier fraction parameter *nu* is an important tradeoff between precision and recall, and we find 0.004 a good setting for precision.

Feature weightings. L1-penalized LR selects 9 out of the 29 features, and Table 3 lists them with their importance estimation from RF. We can see that two of the entropy-related features and the total_indices feature are significantly more important than others. This finding matches experts' intuitions that DRs tend to retrieve relatively more keys and in a relatively more uniform way.

Detection results on $\mathbf{D}_A$. On $\mathbf{D}_A$, we detect 24 DRs and Table 4 shows 6 samples. The classifier G finds 21 DRs and outlier detector finds 9 (both detect 6).

All the 24 detected DRs have very high values in the top-3 features in Table 3, with a percentile rank of at least 91%, and almost 70% of these features are above the 99-th percentile.

Different types of data resellers. To see whether all DRs have the same pattern, we run k-means clustering algorithm with $k = 2$

Table 4: **Example of detected DRs on $\mathbf{D}_A$. Numbers are the percentile rank. Algo indicates the detection method (C=classifier, OD=outlier detector).**

uid	algo	index_day _entropy	total_ indices	index_avg _entropy	minutes_ rate_rank	sum_day _indices
0	both	99.93%	99.93%	99.51%	63.46%	96.17%
1	both	99.98%	99.98%	96.24%	57.36%	98.69%
2	C	99.59%	99.48%	99.38%	67.55%	92.87%
3	C	99.56%	99.66%	99.93%	57.48%	96.75%
4	OD	93.20%	91.11%	96.31%	**99.23%**	99.95%
5	OD	94.95%	94.15%	95.39%	**98.74%**	99.97%

on the 24 detected DRs, and exactly three DRs stand out as a separate cluster, two of which are shown in the last two rows in Table 4.

Interestingly, the classifier detects none of the three outliers. We believe it is not coincidental. Given the limited training set, the classifier tends to overfit to a single DR behavior pattern, but there are more new patterns. For example, the three outliers have all the features high as the other DRs but have additional features like minutes_rate_rank, sum_day_indices and days_num also abnormally high. Intuitively, they are not only retrieving data uniformly over keys but also at an extreme frequency and intensity. Thus, they are highly suspicious DRs, if not the worst ones[1].

Detection results on other DBs. We manually examine the detection results from the other four DBs and confirm that the top-3 features are also significantly high for DRs, and thus showing that the features have some transferability to other data. We show an example detected DR in $\mathbf{D}_B$, who never uses $\mathbf{D}_A$. All of his top-5 features are over the 99-th percentile. We believe that he is a DR because 1) he retrieved nearly 9,000 distinct keys within two hours, 130× higher than average; and 2) he repeatedly and only retrieved a set of around 500 keys in about 20 days, all around the same time of the day. Thus, he has all three characteristics of *large volume*, *spread*, and *strong periodicity*, making him highly suspicious.

Result validation from the data provider. We reported our detected DRs to the security experts at the data provider. They manually examined these results and confirmed most of them. They took actions on the confirmed ones and marked others as suspicious for a longer-term tracking. The method is now deployed in their systems.

Comparison with a naive method. The most popular method for data providers to defend themselves is limiting the query volume (our total_num feature). We compare our detection result with the naive method. Surprisingly on $\mathbf{D}_A$, the top 35 heavy users do not overlap at all with our detected 24 (the heaviest user in our results ranks 36th in utilization).

Security experts do not believe the heavy users are DRs. Comparing to manual labels in $\mathbf{D}_{A(L)}$, no labeled DRs are among the top-10% heaviest users, while 12% of the labeled legitimate users ranks among the top 10%. We show the top-3 heavy users in Table 5 as an example. We can see that their entropy features and total_indices are not very unusual. In fact, they use a few keys repeatedly, but not many keys. This is typical behavior of many automated stock trading scripts. Our method permits these heavy users, unlike the naive method.

[1]Their the entropy is a little lower, but they have so many retrievals and thus can query many keys over and over as cover traffic for their retrieval pattern.

Table 5: Example heavy users. Number are the percentile rank on the features. None are DRs.

uid	rank of total_num	index_day _entropy	total_ indices	index_avg _entropy	minutes _rate_rank
6	1	86.56%	86.73%	89.75%	99.84%
7	2	65.65%	70.10%	56.89%	32.25%
8	3	81.98%	81.30%	84.76%	99.87%

4 CONCLUSION AND FUTURE WORK

Anti-data-reselling (ADR) is key to protect data providers, but many people still use naive query-volume limiting, which we show is not an effective method, while hurting the user experience. In this paper, we identify the most fundamental behavior characteristics of DRs and build a set of sophisticated features and models to capture the query volume, the spread over the dataset, as well as the periodicity of the data retrieval patterns. Experiments on a real-world dataset with 9, 000+ users on five independent databases demonstrated the effectiveness of our method. We also discover interesting divergent behavior patterns of different DRs.

As future work, we will build a visualization system to help experts to examine the detection result. We will also explore ways of leveraging semi-supervised learning to adopt the user-confirmed outlier detection results to fine tune the classifier. Last but not least, we will integrate DR detection results with other log-based fraud detections (e.g., fake account detection) and cross-check the results.

REFERENCES

[1] ACM Digital library. https://dl.acm.org/.
[2] Bloomberg Indices. https://www.bloombergindices.com/.
[3] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
[4] K. Brown and D. Doran. Contrasting web robot and human behaviors with network models. *arXiv preprint arXiv:1801.09715*, 2018.
[5] CNKI. http://oversea.cnki.net/.
[6] D. Doran and S. S. Gokhale. Web robot detection techniques: overview and limitations. *Data Mining and Knowledge Discovery*, 22(1-2), 2011.
[7] FactSet. https://www.factset.com/.
[8] Feature Importance Evaluation. http://scikit-learn.org/stable/modules/ensemble.html.
[9] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna. Pubcrawl: Protecting users and businesses from crawlers. In *USENIX Security Symposium*, 2012.
[10] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008.
[11] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *NIPS*, 2000.
[12] scikit-learn. http://scikit-learn.org/.
[13] D. Stevanovic, A. An, and N. Vlajic. Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications*, 39(10), 2012.
[14] D. Stevanovic, N. Vlajic, and A. An. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing*, 13(1), 2013.
[15] P.-N. Tan and V. Kumar. Discovery of web robot sessions based on their navigational patterns. In *Intelligent Technologies for Information Analysis*. Springer, 2004.
[16] Thomson Reuters. https://www.thomsonreuters.com/en.html.
[17] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Eurocrypt*, 2003.
[18] M. Zabihi, M. V. Jahan, and J. Hamidzadeh. A density based clustering approach for web robot detection. In *ICCKE*. IEEE, 2014.
[19] M. Zabihimayvan, R. Sadeghi, H. N. Rude, and D. Doran. A soft computing approach for benign and malicious web robot detection. *Expert Systems with Applications*, 87, 2017.