Usage Patterns Identification Using Graphs and Machine Learning

Ovidiu-Dan Sonea(✉)

Babes-Bolyai University, Cluj-Napoca, Romania
ovidiu.dan.sonea@gmail.com

Abstract. During the past years, the number of platforms that are introducing a subscription plan is steadily increasing. This phenomenon helps support the developers as well as continuing to provide quality content. Since not so many individuals are willing to spend money or some simply do not have the means, they resort to sharing an account that has a subscription plan. This behavior can, in some instances, be harmful for the developers and, even if it is not, any provider can benefit from knowing what type of clients they have. The solution depicted and explored in this article will focus on using data that is easily available and structuring it in a way that can provide insight into each account activity.

1 Introduction

Since sharing credentials is very easy and many people don't see it as a problem, this practice continues to expand. This phenomenon leads many content creators to be interested in developing a way of identifying shared accounts but often it is not enough just to know if an account is shared or not; content creators want to know how an account is shared. This means that the algorithm must also classify users into patterns that are predefined by the provider to suite their needs. In the end, based on the constrains of each pattern and other metrics calculated, mostly using graph theory, the algorithm provides a sharing probability for each account. Most providers have access to massive amounts of data which, most likely, means that they have the necessary tools to identify password sharing, they just have not found an efficient way processing at the data in order to solve this problem. Given that the algorithm, which will be detailed in the next pages, uses only information that most content providers already have access to, it can be easily implemented successfully on a large number of platforms.

Although there are several solutions that are implemented, these approaches cannot provide a definitive answer for the problem previously described. A few examples are:

1. Fraud Detection attempts to identify individual events/transactions as "fraudulent" do not work in our case, while we need to label the entire subscriber activity as "shared" or not.
2. "Anomalies" spotted within a subscriber activity are not necessarily an indication for account sharing, while sharing can happen with no visible anomalies if the account is shared from the beginning.

It is necessary to detect usage patterns. Having only the label "shared" or "not shared" is insufficient, because the content creator may want to allow certain types of sharing that

© The Author(s) 2021
M. Cochez et al. (Eds.): GKR 2020, LNAI 12640, pp. 84–92, 2021.
https://doi.org/10.1007/978-3-030-72308-8_6

do not harm their business. An example from the streaming industry can be a teenager that went to college and is sharing an account with his/her parents.

2 The Problem

The problems that demand a solution are identifying accounts that are shared and classifying users into usage patterns. The end goal is to give providers insights on their subscribers so they can take action on the users from a certain usage pattern. The solution should also be implemented in a reliable and testable way.

3 Approach

In solving this problem we used several graph theory algorithms to structure the data in a way that ensures the validity of our assumptions and assures that the data was not corrupted in a prior step.

The proposed solution will analyse the subscriber's activity during a given time period, classify the subscriber into a known usage pattern and it will provide a password sharing probability. Moving forward, we will describe the capabilities and functions of the proposed solution on an example from the TV industry.

The raw data, that will be inserted in the algorithm, observes the activity of real users during a month. This data was collected and provided by a client from the industry and contains the following fields: user id (as defined in the clients database), the coordinates at which an event took place, device type, device id and the time at which an event occured. All the data gathered in a time interval will be processed and the end result for each user will be:

- Number of devices – the number of unique devices per subscriber per analysed time interval;
- Location clusters – that are determined by finding common locations between subscriber's devices. The devices that have been seen in a common location will be considered to belong to the same cluster. But at the same time, two devices can be in the same cluster even though they do not have common locations, but they are connected through the locations of other devices from that cluster;
- Type of location clusters – mobile or static cluster;
- The minimum distance between location clusters – the sum of distances between the closest points of different clusters;
- The minimum number of persons behind an account – determined by looking at the subscriber's activity in a chronological order and analysing successive events which are produced by different devices. Here we are determining the cases where the usage cannot be made by a single person.

Based on the detailed criteria, each subscriber will be labeled into a single usage pattern and after that, they will be given an account sharing probability. The table that follows shows how this particular client has chosen to define the patters in order to extract information they considered valuable. The algorithm allows for the patterns to be defined in

Table 1. Detailed view of Usage patterns defined criteria

Usage pattern label	Devices	Cluster configuration	Min. distance between all clusters
1. Single location usage	Less than 5	Only one location	N/A
2. Traveling User	Less than 5	1 static cluster or/and 1 mobile cluster, or 2 mobile clusters	Unspecified
3. Large Family	More than 4 but less than 21	Less than 3 and at most 1 static	Less than 100 Km
4. Multiple houses at close distance	Unspecified	2 or 3 static clusters and the total number less than 5	Less than 100 Km
5. Heavy usage across multiple locations	Unspecified	3 or more static and the total number more than 5	More than 100 km
6. Multiple distant locations	Unspecified	2 or more static but the total number less than 5	Unspecified
7. Concurrent usage from different locations	Less than 5	1 static cluster or/and 1 mobile cluster, or 2 mobile clusters	Unspecified
8. Secondary location	Less than 8	2 clusters and at most one static	Unspecified
9. Few clusters close to each other	Unspecified	Less than 4 clusters but at most one static	Less than 100 km
10. Multiple clusters close to each other	Unspecified	More than 3 but at most one static	Less than 100 km
11. Few clusters distant from each other	Unspecified	Less than 4 clusters but at most one static	More than 100 km
12. Multiple clusters on multiple distant location	Unspecified	More than 3 but at most 1 static	More than 100 km

many ways without them affecting its performance. There are, however, a few limitations when it comes to defining these usage patterns. The limitations are: the defined patterns must be mutually exclusive, meaning that a subscriber must fit into only one pattern and that the entire pool of subscribers must be fitted into the defined patterns (there can't be subscribers that don't have a pattern assigned).

4 Implementation

The algorithm is structured in way in which it achieves the intended goal by following nine steps (Table 2).

Step 1. This step mainly deals with the input processing by reading the raw data from the specified time period and filtering out inputs that might not be relevant for the algorithm.
Step 2. After the input is validated, the data must be arranged in a meaningful way for it to provide the desired results. This means grouping entries by users and sorting them in chronological order. For the algorithm to be more efficient, this step also deals with data compression. Meaning that, if there are multiple consecutive entries from the same device, they will be considered as being a single event with the starting date of the most recent and the end date of the last entry from the consecutive sequence.
Step 3. A square matrix is created with the size being the number of devices squared. This matrix represents a way to track which devices are being used by different persons. If such a case is found, the values in the matrix corresponding to the found devices will be marked with "1". Additionally, we create a buffer that contains events which span at most 48 h (we assumed that in this time period you can physically get to any two points in the United States). In this buffer, we recreate the activity of the subscriber by adding each event from the chronological event array, one by one, and check if it is physically possible. We have two ways of analysing if the activity is done by one person or more. The first one is by looking at two consecutive events and the second one is by checking three or more events (maximum is determined by the number of events in buffer) and analysing them with a machine learning algorithm (we used XGBoost Classifier with the objective of logistic regression). Choosing which consecutive events are analysed (and how) is a challenge by itself, since there can be multiple occurrences of the same device in buffer. To solve this, we created an occurrence array in which we store the last occurrence of each device. If a device that is already in the buffer is added again, then we analyse with a XGBoost algorithm the loop created by the two devices, as well as the whole buffer. For an example please look at Fig. 1.
Step 4. At this point, we can start calculating the minimum number of persons that is needed for the subscribers activity to be physically possible. Having the matrix from Step 3, we can consider it to be a graph represented as a matrix where we know that the values of one indicate that the devices corresponding to the line and the column

Table 2. List of used terms.

Term	Description
Static device	A device that has very low mobility (e.g.: GameConsole, SetTopBox, TV, etc.)
Mobile device	A device that has high mobility (e.g.: MobilePhone, Tablet, Laptop, etc.)
Cluster	A cluster is created by finding common locations between devices. The devices that have been seen in a common location will be considered to belong in the same cluster. Two devices can be in the same cluster even though they do not have common locations but can be connected through the locations of other devices
Mobile cluster	A cluster that has no static devices
Static cluster	A cluster that has at least one static device

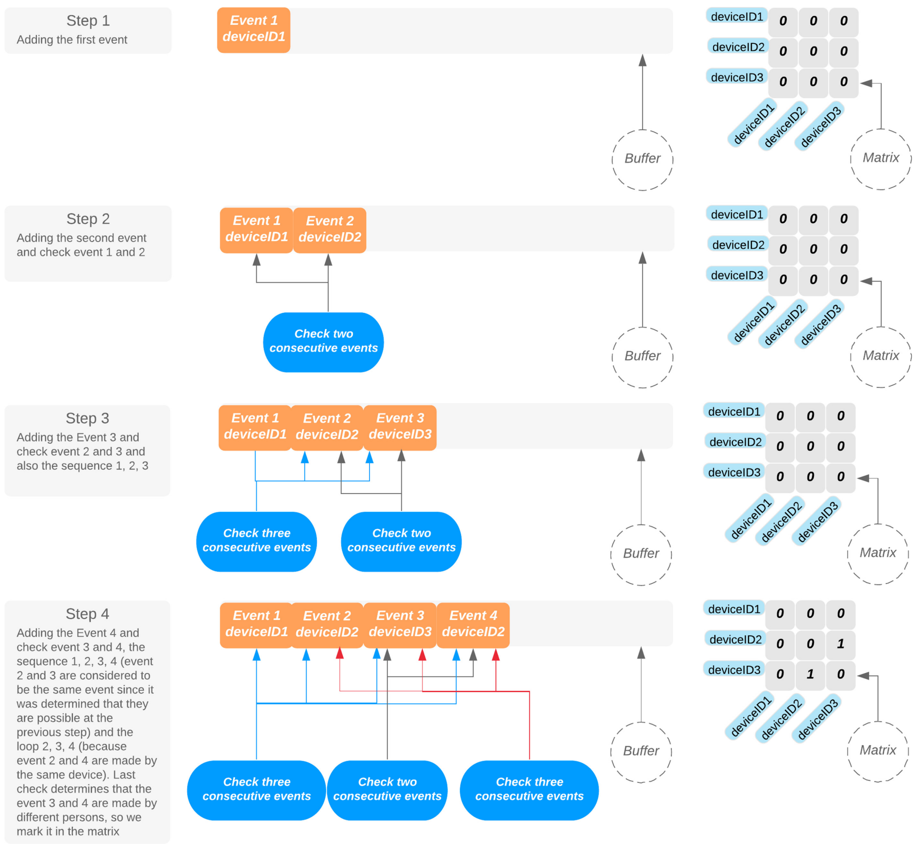
Step 1
Adding the first event
Event 1 deviceID1
Buffer
Matrix
deviceID1
deviceID2
deviceID3
0 0 0
0 0 0
0 0 0
Step 2
Adding the second event and check event 1 and 2
Event 1 deviceID1
Event 2 deviceID2
Check two consecutive events
Step 3
Adding the Event 3 and check event 2 and 3 and also the sequence 1, 2, 3
Event 3 deviceID3
Check three consecutive events
Step 4
Adding the Event 4 and check event 3 and 4, the sequence 1, 2, 3, 4 (event 2 and 3 are considered to be the same event since it was determined that they are possible at the previous step) and the loop 2, 3, 4 (because event 2 and 4 are made by the same device). Last check determines that the event 3 and 4 are made by different persons, so we mark it in the matrix
Event 4 deviceID2
0 0 0
0 0 1
0 1 0

Fig. 1. Graphical representation of Step 3

are used by different persons. At the same time, in graph theory, we can say that these two vertices are adjacent. At this point, the problem can be solved by a simple Graph Coloring algorithm [1]. To ensure that an optimal solution is found, we need to apply the algorithm from each vertex since this problem is NP-complete. From an efficiency point of view, this would seem extreme and inefficient, but we deal with small graphs, and in our experiments we had no issues. On average, in the data we had, the number of devices per account was around four, and only in extreme cases the count exceeded thirty. The end result will contain all the optimal solutions of coloring the graph, since, in most cases, it is not just one. Translating from graph theory, this means we found the minimum number of persons and all the possible ways of pairing a person with one or more devices.

Step 5. In this part of the algorithm, we implemented a method that is able to quantify how connected is the activity of a user. This quantifier is represented by the number of clusters, a term which was previously explained. Since we know the locations visited by a device, we can consider each device as being a graph and the visited locations as the vertices of this graph. Now, we have multiple graphs with common vertices but we don't know which of these vertices are mutual. If two graphs have a common vertex, that means they can be considered as one big graph. In the end, each remaining graph translates to a cluster. To solve this problem in an efficient way, we devised an algorithm based on a balanced binary search tree [2]. The information, contained in the nodes that create the tree, represents the location (which serves as a search key) and the device id (which is unique for an account). We add, one by one, all the locations that were visited by a user and if that location already exists in the tree, we know which device was already seen there. By having an array where we keep track of such cases, in the end, we can determine all the clusters. The previously mentioned array has the size equal to the number of devices. Each value in the array represents the index of a cluster in which a device is positioned.

Step 6. Each device has a degree of mobility, these degrees being "mobile" or "static", based on the type of the device. Using the result from Step 5 we can determine which cluster is mobile and which one is static. A static cluster has at least one static device and a mobile cluster does not have any static devices. If a subscriber has two or more distinct static clusters, we can safely label this account as being shared. Having the processed clusters at this step, we can also calculate the minimum distance between all clusters. To do this, we have to find the closest locations between each two clusters and after that, apply the Dijkstra algorithm [5] to create a minimum spanning tree. The sum of all remaining edges represents the minimum distance between all clusters.

Step 7. By using the results from Step 4 and 5, we can find distinct persons belonging to one or more clusters that have not visited other clusters and have never been in contact with the persons belonging to those clusters. In this instance, we can safely assume that we detected account sharing but, because Step 4 does not always return a single solution, we must check that the number of cases where we identified account sharing, divided to the total number of cases, is 1, before labeling an account as shared. The result of the division will be taken into account when calculating the sharing probability.

Step 8. Using the results from the steps above, we determined some thresholds that create a pattern and fit each subscriber in the corresponding usage pattern.

Step 9. In the end, using a machine learning algorithm (XGBoost Regressor with the objective of linear regression) a sharing probability is calculated. The algorithm takes into account the number of clusters, the number of devices, minimum number of persons, the usage pattern and the number obtained from Step 7.

5 Technologies

We decided to put XGBoost [4] at the core of this algorithm since it is very efficient, flexible, and it can learn really quickly. This was highly important because we didn't had any pre-labeled data and creating multiple thousands of repetitive entries in order to train a neural network would have been really difficult and time consuming. Using this approach we only had to label about one thousand for each model. All mentioned factors make the implementation of these types of gradient boosted decision trees to be the perfect solution for this problem.

The model used at Step 3 has an XGBClassifier with a structure as displayed in Table 3(a). The end result for this model was achieving an accuracy of 91.87% for the

Table 3. XGBClassifier Structures

(a) Step 3		(b) Step 9	
Parameter name	Value	Parameter name	Value
max_depth	1	max_depth	1
min_child_weight	3	min_child_weight	1
learning_rate	0.065	learning_rate	0.05
n_estimators	500	n_estimators	700
Objective	binary:logistic	Objective	reg:linear
Gamma	0	Gamma	0
Subsample	0.9	Subsample	0.4
colsample_bytree	0.7	colsample_bytree	0.5
colsample_bylevel	1	colsample_bylevel	1
scale_pos_weight	1	scale_pos_weight	1

training data and an accuracy of 93.24% for the validation data, which means that there was no over fitting.

For the model at Step 9 we used XGBRegressor with the specifications shown in Table 3(b). The accuracy for the training data was 89.22% and for the validation data 93.11%. For both models, the evaluation metric used was area under the curve(auc) [3]. During the tests made to find an optimal model for these tasks, we obtained an accuracy close to 100% for the training data but, for the validation, the accuracy was much lower, meaning that the model just learned the results.

6 Results and Analysis

We had access to a large data set. The Figs. 2, 3 and 4 are created from a data set with more than 13 million subscribers. Each subscriber had one or more events, meaning that, at least for the situations it was tested for, the algorithm produces results that can be considered to reflect reality.

Looking at Fig. 2, we notice that most users are classified as having either less than 20%, either 100% sharing probability, meaning that the algorithm is fairly certain of it's prediction. This is very important, since it would be troublesome to predict a high probability to a user that is not sharing the account.

Observing Fig. 3, it is obvious how unbalanced the distribution of patterns is. However, looking at Fig. 4, this represents good news for the provider of this data since the patterns that have the highest number of users represent a low risk of sharing.

Overall, 10% of shared accounts may not seem as a large number. However, taking into consideration that these users share their account with at least another person, it means that, if all those who benefit from sharing would get a subscription, the number of subscribers would increase with at least 10%. From a marketing point of view, this is a considerable and very favorable percentage for providers.

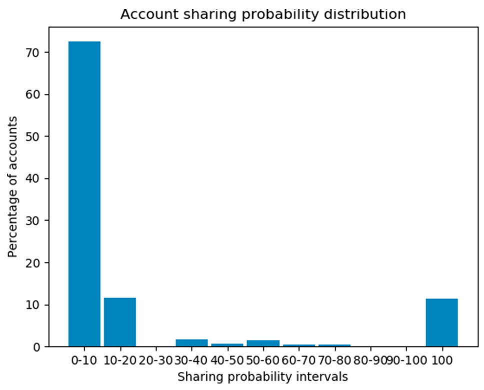

Fig. 2. Sharing probability distribution

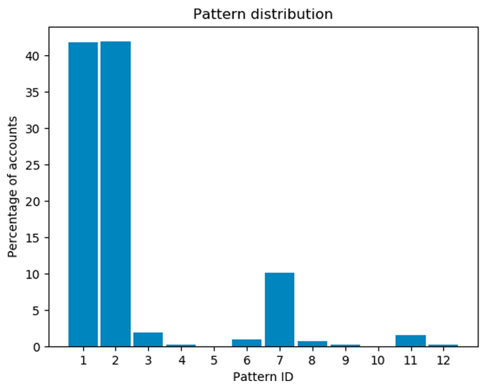

Fig. 3. Pattern distribution where the bar index represents the pattern from Table 1

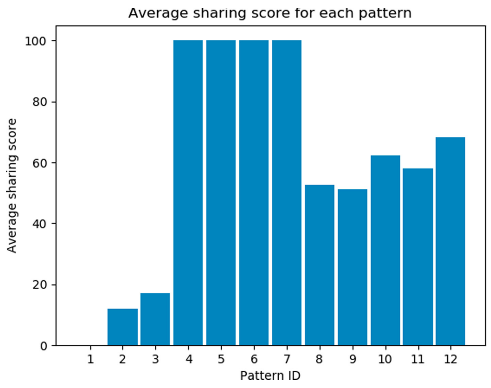

Fig. 4. Average sharing score for each pattern where the bar index represents the pattern from Table 1

7 Conclusion

Looking at the results, we are satisfied with the overall performance since we found a way to identify account sharing in a reliable way. Not only this, but we can actually determine multiple types of sharing. Moreover, the implementation of this algorithm is simple and can be done by other providers since this type of data is easily available. Even though the presented solution does not identify all accounts which are being shared, we consider this to be a step in the right direction. With further research, we are confident that more usage patterns will emerge and as a consequence the number of shared accounts might increase.

Acknowledgements. I would like to thank Dr. Christian Săcărea for his kindness, help and time invested in making this article possible.

References

1. Aslan, M., Baykan, N.: A performance comparison of graph coloring algorithms. Int. J. Intell. Syst. Appl. Eng. **4**, 1–1 (2016)
2. Austern, M., Stroustrup, B., Thorup, M., Wilkinson, J.: Untangling the balancing and searching of balanced binary search trees. Softw. Pract. Exper. **33**, 1273–1298 (2003)
3. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Patt. Recogn. **30**(7), 1145–1159 (1997)
4. Chen, T., Guestrin, C: Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. ACM, New York (2016)
5. Javaid, A.: Understanding Dijkstra algorithm. SSRN Electron. J. (2013). https://doi.org/10.2139/ssrn.2340905