Detecting Influencers in Very Large Social Networks of Games

Leonardo Mauro Pereira Moraes and Robson Leonardo Ferreira Cordeiro
Institute of Mathematics and Computer Sciences, University of São Paulo,

Keywords: Data Mining, Social Networks of Games, Player Modeling, Classification, Feature Extraction, Data Streams.

Abstract: Online games have become a popular form of entertainment, reaching millions of players. Among these players are the game influencers, that is, players with high influence in creating new trends by publishing online content (*e.g.*, videos, blogs, forums). Other players follow the influencers to appreciate their game contents. In this sense, game companies invest in influencers to perform marketing for their products. However, how to identify the game influencers among millions of players of an online game? This paper proposes a framework to extract temporal aspects of the players' actions, and then detect the game influencers by performing a classification analysis. Experiments with the well-known Super Mario Maker game, from Nintendo Inc., Kyoto, Japan, show that our approach is able to detect game influencers of different nations with high accuracy.

1 INTRODUCTION

The digital game market is in constant ascendancy, both in the production and in the consumption. This industry moves billions of dollars per year (Lucas, 2009). According to Newzoo[1], a specialist company of game marketing, the estimated value for 2018 is US$ 134.9 billion. Only in the first quarter of 2018, players produced more than 260 million hours of online videos related to game contents on Twitch and YouTube platforms.

Online games are dynamic environments, in which the players interact with the game and with other players all over the world (Yannakakis and Togelius, 2015). The interactions performed by each player represent relationships of several types; they can relate to other players (*e.g.*, friend, sharing games) or to the game itself (*e.g.*, play, like) (Lee et al., 2012). These relationships form a social network, *i.e.*, in this context a *Social Network of Games*.

For example, the well-known Super Mario Maker game (SMM), from Nintendo Inc., Kyoto, Japan, forms a very large social network. According to Nintendo[2], it has more than 3.5 million players, and 7.2 million courses (or maps) that were already played more than 600 million times. In this game, a player can play, give a star, break a record or comment on online courses created by other players, besides creating his/her own courses to share with the world.

Influencer users in the digital field exist since the popularization of social media (Pei et al., 2018). They produce digital contents (*e.g.*, online videos) that arouse the attention of other users with similar tastes. Among them, there are the game influencers that produce contents related to digital games (Gros et al., 2018; Hilvert-Bruce et al., 2018). Players follow the game influencers looking for entertainment and credible information on the universe of the games (Sjöblom and Hamari, 2017; Gros et al., 2018).

Game influencers get recognition from the players who follow them. Consequently, game companies invest in influencers to endorse and perform marketing for their products (Sjöblom and Hamari, 2017). Hence, an influencer has direct relevance in such phenomena as viral marketing, innovation diffusion and behavior adoption (Pei et al., 2018). To detect game influencers is therefore almost mandatory when looking for popular preferences, new trends, and the like (Sjöblom and Hamari, 2017; Hilvert-Bruce et al., 2018).

This paper proposes a novel framework to detect game influencers in Social Networks of Games. Given the actions of millions of players in an online game, *how to detect the game influencers?* It is obvi-

[1]Newzoo. "Insights". https://newzoo.com/insights/ (accessed December 18, 2018).

[2]NintendoAmerica. "If you played every level in #SuperMarioMaker for 1 minute each, it would take you nearly 14 years to play them all!". https://twitter.com/NintendoAmerica/status/732624228428750848/ (accessed November 10, 2018).

Moraes, L. and Cordeiro, R.
Detecting Influencers in Very Large Social Networks of Games.
DOI: 10.5220/0007728200930103
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 93-103
ISBN: 978-989-758-372-8

ously necessary to model the players' characteristics. We express the problem as a classification task, and propose a player modeling heuristic based on temporal aspects of their courses analyzing the history of "likes" over time. To validate our proposal, we studied the famous Super Mario Maker game, from Nintendo Inc., Kyoto, Japan, and report experimental results to show that our framework detects game influencers of different nations with high accuracy.

The rest of this paper is organized as follows: it starts with background concepts (Section 2), followed by the related work (Section 3), the proposed methodology (Section 4), the experiments (Section 5) and the conclusion (Section 6). Table 1 lists the main symbols of our notation.

Table 1: Definitions.

	Symbols	Definitions
Graph	$G_{dev} = \{V, E_{dev}\}$	Developer network
	$G_{star} = \{V, E_{star}\}$	Stars network
	$g(n) \mid n \in V$	Degree centrality
Modeling	$T = [t_1, t_2, \ldots]$	Timestamps
	$c = [\varepsilon_1, \varepsilon_2, \ldots]$	Stars stream
	$C_p = \{c_1, c_2, \ldots\}$	Player's courses
Extractor	$LR(p, L)$	Algorithm 1
	$DR(p, L)$	Algorithm 2
	$CA(p, L)$	Algorithm 3
	$F_{ALL}(p, L)$	Ensemble method

2 BACKGROUND CONCEPTS

This section discusses Social Networks of Games and presents the problem definition.

2.1 Social Networks of Games (SNG)

Social networks describe interactions and relations among users (Barabási and Pósfai, 2016). In a social network, the users represent their relations by links. There are many types of links, *e.g.*, links determined by individual opinions on other individuals, links denoting collaboration, links resulting from behavioral interactions, and so on (Savić et al., 2019).

Social links may also be present in digital games, where the players interact, compete and relate among each other (Lee et al., 2012). A real-world example is the Super Mario Maker game (SMM). In this game, players can elaborate their own Super Mario courses and share their courses with the world. A Super Mario course is a map of the game that players play. Besides playing and developing courses, a player can also give a star to (*i.e.*, similar to "like") a course elaborated by another player.

Figure 1 illustrates two players p_1 and p_2 interacting (develop, star, etc.) with three courses c_1, c_2 and c_3. Such relationships form a social network. In this sense, online games have their intrinsic social networks; we coin the term Social Networks of Games (or simply SNG) to refer to them in this paper.

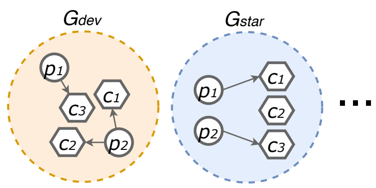


Figure 1: Graphs of different types of interactions (develop, star, etc.) in a Social Networks of Games.

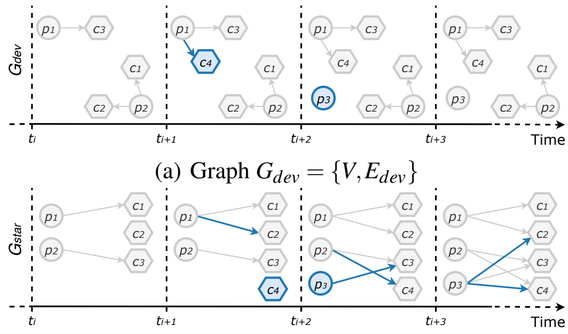
2.2 Network and Problem

In the SMM game, players can elaborate and share their courses. In this sense, the SMM network includes a directed bipartite graph $G_{dev} = \{V, E_{dev}\}$ with edges $e \in E_{dev}$ of players $p \in V$ sharing courses $c \in V$. Additionally, graph G_{dev} changes over time. Thus, in time t_i, graph $G_{dev,i}$ presents a determined topology, while new nodes (*i.e.*, new players and/or courses) and new edges can arrive in graph $G_{dev,i+1}$, in time t_{i+1}, where $t_i < t_{i+1}$. Thus, since there exists a stream of graphs $[G_{dev,1}, G_{dev,2}, \ldots]$ as a function of time, it is a dynamic network (Westaby, 2012).

Figure 2(a) illustrates this network. The initial graph $G_{dev,i}$ has nodes $V = \{p_1, p_2, c_1, c_2, c_3\}$, *i.e.*, two players and three courses. The corresponding edges are $E_{dev} = \{(p_1, c_3), (p_2, c_1), (p_2, c_2)\}$, defining the courses shared by each player. In $G_{dev,i+1}$, player p_1 shares a new course c_4. In $G_{dev,i+2}$, there appears a new player p_3 without sharing any course. At last, in $G_{dev,i+3}$, nothing changes. Thus, the network can only grow over time.

Besides creating new courses, a player can also star courses of others players in SMM. In this sense, the SMM network has also another directed bipartite graph $G_{star} = \{V, E_{star}\}$ with edges $e \in E_{star}$ of players $p \in V$ that give stars to courses $c \in V$. Note that the nodes in V are the same for both graphs $G_{dev,i}$ and $G_{star,i}$, considering a time instant t_i. Similarly to graph G_{dev}, graph G_{star} is also a dynamic network.

Figure 2(b) illustrates this network. In time t_i, graph $G_{star,i}$ has nodes $V = \{p_1, p_2, c_1, c_2, c_3\}$ and edges $E_{star} = \{(p_1, c_1), (p_2, c_3)\}$ indicating players that starred courses. In $G_{star,i+1}$, player p_1 stars



(a) Graph $G_{dev} = \{V, E_{dev}\}$

(b) Graph $G_{star} = \{V, E_{star}\}$

Figure 2: Graphs G_{dev} and G_{star} changing over time.

course c_2, and there appears a new course c_4 with no star. In $G_{star,i+2}$, there appears a new player p_3 that stars course c_3; also, player p_2 stars course c_4. At last, in $G_{star,i+3}$, player p_3 stars courses c_2 and c_4.

According to this discussion, the SMM game can be represented by two dynamic networks, G_{dev} and G_{star}, whose topologies are correlated and evolve as a function of time. To analyze such correlated evolution is essential when spotting game influencers, since the courses that they develop become popular right after they advertise them in other media, like Twitch and YouTube, while the other players' courses gain popularity gradually. The correlated evolution is probably even more important than looking for static graph topologies in individual time stamps. With that in mind, the main problem that we tackle in this paper is: *how to spot game influencers by evaluating the correlated evolution of graphs G_{dev} and G_{star}?*

Obviously, there exist other types of interactions that lead to additional dynamic networks describing the SMM game, such as, play, break time record, etc. Nevertheless, without loss of generality, we only consider G_{star} and G_{dev} in this paper, and show that they are enough to accurately identify game influencers.

The next section describes existing works focused on identifying influencer users in social networks.

3 INFLUENCER DETECTION

To the best of our knowledge, there is no research on detecting influencers in Social Networks of Games; the existing approaches to detect influencers in other types of social networks are thus the closest related works. In this section, we outline the state-of-the-art.

The naïve approaches to detect digital influencers are based on selecting the top-ranked nodes identified by some centrality indices; more advanced strategies are greedy methods and heuristic methods (Wang et al., 2017; Al-Garadi et al., 2018). The centrality-based methods find a set of the most influential nodes based on the network topology (Barabási and Pósfai, 2016). In (Morone et al., 2016), it is presented an algorithm that ranks nodes based on collective influence propagation, which quantifies the nodes' "influences" in the network. Meanwhile, another research infers the influential nodes based on spreading paths (*i.e.*, the links) (Wang et al., 2017).

Unfortunately, these algorithms work only on homogeneous networks (*i.e.*, a single type of node), while our problem refers to heterogeneous/bipartite networks (*i.e.*, nodes of two types, players and courses). Also, we need to correlate two dynamic networks, and analyze their evolution over time. In this sense, it is difficult to use centrality indices because they infer knowledge only based on the topology of a single static network. To tackle our problem, one must observe the nodes' features, so to model game influencers' characteristics instead of ranking them.

In (Chino et al., 2017), a dynamic graph of comments and online reviews was analyzed to detect suspicious users, like spammers. The authors model the temporal aspect of the volume of activity (*e.g.*, number of characters in a review) to detect an anomaly in a social media. The proposed temporal aspect modeling is interesting and inspiring, but this research works with only one dynamic network and specific application-related features of the contents, besides detecting suspicious users; not, influential users.

Other approaches extract features to describe social media users; then, they use the features to train Machine Learning (ML) algorithms to identify the most influential ones. ML-based approaches have great potential to identify influential users in large networks and complex networks (Al-Garadi et al., 2018). In (Liu et al., 2014), temporal features are extracted based on microblog posts (post-features) to train an ML method to identify influential users in a temporal microblog network. The authors infer features for a user according to his/her number of followers, the number of microblog posts, the number of responses received and the number of comments over time.

In (Qi et al., 2015), the aforementioned work is extended by identifying domain-topics of influential users so to improve the microblog's search engines. The improved work uses temporal post-features combined with microblog text to fit the domain-topics of each influential user. Unfortunately, both works (Liu et al., 2014) and (Qi et al., 2015) use a single dynamic graph and heavily depend on the post-features, so it is not clear how to use them to tackle our problem.

Finally, let us summarize the state-of-the-art on influencer detection. Some techniques use centrality metrics to rank influential users based on graph topology (Morone et al., 2016; Wang et al., 2017). Other methods are designed to work with dynamic networks; they commonly perform feature extraction from the network and from the users' external contents to train ML algorithms (Liu et al., 2014; Qi et al., 2015). Also, user modeling based on temporal aspects has been applied (Chino et al., 2017). However, these works cannot tackle our problem mainly because they do not correlate more than one dynamic network and/or fail to quantify the changes/evolution over time. In the next section, we present a novel framework to tackle the problem.

4 PROPOSED METHODOLOGY

This section presents a new framework to detect influencers in very large Social Networks of Games. The main question here is: *how to model the correlated evolution of players' actions on the dynamic networks G_{dev} of developers and G_{star} of stars?* To answer it, we propose a new player modeling heuristic based on temporal aspects of their courses by extracting features from the history of players' actions over time.

4.1 Main Idea

Individually, neither network G_{dev} nor G_{star} has intrinsic characteristics that are truly useful to spot game influencers. For example, by using an outdegree centrality ranking algorithm on G_{dev} we simply rank the players by the number of shared courses; note that it does not help much. Similarly, network G_{star} would lead to a ranking of players by the number of stars given, which does not help either. Both rankings fail to present relevant characteristics of the players that would help identifying the influencers. Unfortunately, the same problem happens when using more elaborated analytical techniques, simply because there is not enough information in any of the networks alone; they must be analyzed combined.

As it was described before, we need to capture relevant information from the correlated evolution of both G_{dev} and G_{star} over time. To make it possible, we propose to model this correlated evolution using data streams, and to extract relevant features from the streams to be used by a classification ML algorithm.

4.2 Stream Modeling

In this sense, we obtain from G_{star} the quantity ε_i of stars received by each course c by applying degree centrality $g(c) \mid c \in V$ in each instant of time t_i. Thus, a course is represented here by a stream of star counting events $c = [\varepsilon_1, \varepsilon_2, \ldots]$, where each event ε_i refers to a pair $(star_i, t_i)$ with a number of stars $star_i$ and a time stamp t_i. Figure 3 illustrates this idea for the toy graph G_{star} shown in Figure 2(b). The number of stars is constant over time for course c_1, as it is shown in Figure 3(a). Course c_2 received one star from p_1 in time t_{i+1}, and another one from p_3 in time t_{i+3} (Figure 3(b)). Course c_3 received one star from p_3 in time t_{i+2} (Figure 3(c)). At last, course c_4 received a new star in each time instant after it arises in time t_{i+1} (Figure 3(d)).

In this way, each course has a star counting stream through time instants $T = [t_1, t_2, \ldots]$. In addition, by observing G_{dev}, for each player p its elaborated courses are extracted, so $C_p = \{c_1, c_2, \ldots\}$. Therefore, we have for each player p a set C_p of elaborated courses and their stars counting stream over time.

Now, we need to extract temporal features from each player. Note that each player p develops courses $c \in C_p$, and each course c has a star counting stream. Thus, *how to model the players based on their courses?* To accomplish this goal, we extract temporal features of the courses and ensemble them to model the players, as it is detailed in the next section.

4.3 Temporal Feature Extraction

This section describes our temporal aspects models. We developed three temporal features extractors. Each extractor captures a specific characteristic of the star counting stream. They use the players' courses and their stars counting streams to infer the features.

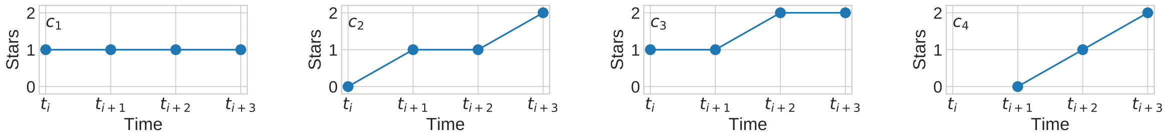
4.3.1 Linear Regression (LR)

The first feature extractor, called Linear Regression (LR) extractor, infers features related to the stars ascension of the courses, *i.e.*, the growth of star counting over time. The linear regression model is widely used in social sciences; it can model the statistical relation between the variables.

We used the well-known Least Squares method for estimating the unknown parameters in the linear model. Since each event ε_i in the stream has only two attributes (*i.e.*, $stars_i$ and t_i), the linear function can be simplified to a vector (α, β) (Equation 1).

$$f(t_i) = \alpha + \beta \cdot t_i = (\alpha, \beta) \tag{1}$$

In this sense, $f(t_i)$ represents an estimated star counting for time t_i. The parameters (α, β) are inferred by the stars stream $c = [\varepsilon_1, \varepsilon_2, \ldots]$. Considering that $\overline{stars}$ is the average number of stars and $\bar{t}$ is the average time, the parameters are defined as follows:



(a) c_1 stream (b) c_2 stream (c) c_3 stream (d) c_4 stream

Figure 3: Stream Modeling on the toy graph G_{star} shown in Figure 2(b).

$$\beta = \frac{\sum t_i stars_i - \frac{1}{|c|}\sum t_i \sum stars_i}{\sum t_i^2 - \frac{1}{|c|}(\sum t_i)^2} \qquad (2)$$
$$\alpha = \overline{stars} - \beta \cdot \bar{t}$$

Figure 4 illustrates an example of linear regression for a star counting stream $c = [0,1,4,6,6,7,7]$ between time instants t_1 and t_7. The LR extractor finds the linear function and extracts the angle $\measuredangle_c$ and the coefficient of determination R_c^2 of the course c. In this example, $\measuredangle_c = 51.34^o$ and $R_c^2 = 0.88$.

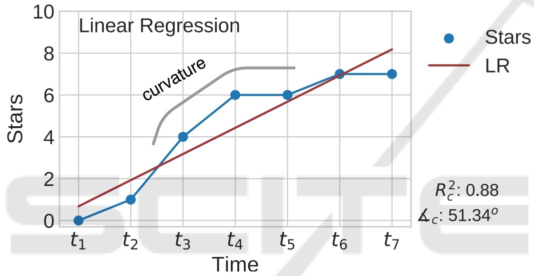


Figure 4: Linear Regression for a star counting stream.

The angle $\measuredangle_c$ represents how ascending are the events of a star counting stream c. Meanwhile, the coefficient of determination R_c^2 is the proportion of the variance from the linear regression prediction, ranging from 0 to 1. In this range, $R_c^2 = 1$ is the best prediction for a stream c. The extractor uses coefficient R_c^2 to estimate a "curvature" on the stream; low R_c^2 represents how large the "curvature" is.

The LR extractor is in Algorithm 1. It infers for each player's course $c \in C_p$ (Line 8) the linear function $model_c$, angle $\measuredangle_c$ and coefficient of determination R_c^2 (Line 9). In addition, the average angle coefficient $\overline{\measuredangle}$ (Line 15) and the corresponding standard deviation $\sigma_\measuredangle$ (Line 16) are extracted from the player's courses.

How ascending is each player p? We measure the angle $\measuredangle_p$ (Line 14) of a player p by the average vector $model_p$ (Line 13) of the linear function of the courses (Equation 3):

$$model_p = \frac{1}{|C_p|}\sum_{c \in C_p} model_c \qquad (3)$$
$$\measuredangle_p = angle(model_p)$$

Additionally, it measures the entropy $S_\measuredangle$ of courses' angles (Line 17). Entropy is a measure of the randomness in the information, ranging from 0 to ∞ as the randomness increases. Thus, entropy verifies the tendency of player's courses to have similar ascension patterns, *i.e.*, similar angles.

In Line 18, it is obtained the bottom-L lower coefficients of determination R_c^2 of the courses, represented by $R_{rank}^2 = [min_1(R^2), \ldots, min_L(R^2)]$. Also, the top-$L$ great angle $\measuredangle_c$ of courses is computed in Line 19, represented by $\measuredangle_{rank} = [max_1(\measuredangle), \ldots, max_L(\measuredangle)]$. Thus, the extractor uses a parameter $L \in \mathbb{N}$ to determine the top-L and bottom-L.

Note the algorithm captures the bottom-L lower coefficients of determination R_{rank}^2 to identify "curvatures", whereas top-L angles $\measuredangle_{rank}$ to represent the greatest star counting stream of a player.

Algorithm 1: LR Extractor.

1: **procedure** LR-MAP(c) ▷ Course Modeling
2: $model_c \leftarrow$ LinearRegression(c)
3: $\measuredangle_c \leftarrow angle(model_c)$
4: $R_c^2 \leftarrow R^2(model_c, c)$
5: **return** $model_c$, $\measuredangle_c$, R_c^2

6: **procedure** LR(p, L) ▷ Player Modeling
7: Initialize $models$, $\measuredangle$, R^2 with $\emptyset$
8: **for** $c \in C_p$ **do**
9: $model_c$, $\measuredangle_c$, $R_c^2 \leftarrow$ LR-MAP(c)
10: $models \leftarrow models \cup \{model_c\}$
11: $\measuredangle \leftarrow \measuredangle \cup \{\measuredangle_c\}$
12: $R^2 \leftarrow R^2 \cup \{R_c^2\}$
13: $model_p \leftarrow mean(models)$
14: $\measuredangle_p \leftarrow angle(model_p)$
15: $\overline{\measuredangle} \leftarrow mean(\measuredangle)$
16: $\sigma_\measuredangle \leftarrow std(\measuredangle)$
17: $S_\measuredangle \leftarrow entropy(\measuredangle)$
18: $R_{rank}^2 \leftarrow bottom(R^2, L)$
19: $\measuredangle_{rank} \leftarrow top(\measuredangle, L)$
20: **return** $\measuredangle_p$, $\overline{\measuredangle}$, $\sigma_\measuredangle, S_\measuredangle, R_{rank}^2, \measuredangle_{rank}$

4.3.2 Delta Rank (DR)

The Delta Rank (DR) extractor infers features related to star counting differences in time. The difference $\Delta \in \mathbb{N}$ between two sequential events $\varepsilon_i, \varepsilon_{i+1}$ is:

$$\Delta = stars_{i+1} - stars_i \tag{4}$$

Δ means how many new stars a course c received in a short temporal interval $t_{i+1} - t_i$. In this way, a set D of star counting differences is extracted for a course c, so $D = \{\Delta_1, \Delta_2, \ldots\}$. Figure 5 illustrates this idea. Course c has six pairs of sequential events, so $D = \{\Delta_1, \ldots, \Delta_6\}$ with the values $1, 3, 2, 0, 1$ and 0, respectively. By ranking the values in D it is possible to identify the greatest Δ; it is $\Delta_2 = 3$ in this example. Also, we measure the entropy S_D of D values. In this example, it is $S_D = 1.33$.

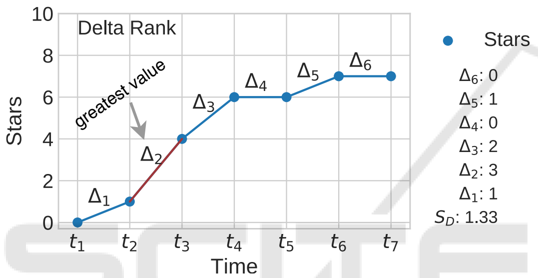


Figure 5: Delta Rank for a star counting stream.

The DR extractor is in Algorithm 2. It represents a player p as the union of D for all its courses (Line 12), so $D_p = D_1 \cup \ldots \cup D_{|C_p|}$. A set S_p of entropy values S_D for the courses (Line 13) is also collected. Then we use D_p and S_p to infer features for player p.

In this analysis, we compute: $\overline{\Delta}$ that is the average of D_p values (Line 14); σ_Δ that is the corresponding standard deviation (Line 15), and; S_Δ that is the entropy of D_p values (Line 16). Also, the extractor obtains the top-L great star counting differences Δ of the player, represented by $\Delta_{rank} = [max_1(\Delta), \ldots, max_L(\Delta)]$ (Line 17).

Additionally, the extractor analyzes the entropy values S_D of the player's courses. It calculates the average entropy $\overline{S}$ of the courses (Line 18), and the corresponding standard deviation σ_S (Line 19). Also, the extractor obtains the top-L great entropy values S_D of the player's courses, represented by $S_{rank} = [max_1(S_D), \ldots, max_L(S_D)]$ (Line 20).

4.3.3 Coefficient of Angle (CA)

The Coefficient of Angle (CA) extractor infers features related to the angles between events. The angle θ between two sequential events $\varepsilon_i, \varepsilon_{i+1}$ is:

Algorithm 2: DR Extractor.

1: **procedure** DR-MAP(c) ▷ Course Modeling
2: Initialize D with $\emptyset$
3: **for** $i \leftarrow 1$ **to** $|c| - 1$ **do**
4: $\Delta \leftarrow stars_{i+1} - stars_i$
5: $D \leftarrow D \cup \{\Delta\}$
6: $S_D \leftarrow entropy(D)$
7: **return** D, S_D

8: **procedure** DR(p, L) ▷ Player Modeling
9: Initialize D_p, S_p with $\emptyset$
10: **for** $c \in C_p$ **do**
11: D, $S_D \leftarrow$ DR-MAP(c)
12: $D_p \leftarrow D_p \cup D$
13: $S_p \leftarrow S_p \cup \{S_D\}$
14: $\overline{\Delta} \leftarrow mean(D_p)$
15: $\sigma_\Delta \leftarrow std(D_p)$
16: $S_\Delta \leftarrow entropy(D_p)$
17: $\Delta_{rank} \leftarrow top(D_p, L)$
18: $\overline{S} \leftarrow mean(S_p)$
19: $\sigma_S \leftarrow std(S_p)$
20: $S_{rank} \leftarrow top(S_p, L)$
21: **return** $\overline{\Delta}$, σ_Δ, S_Δ, Δ_{rank}, $\overline{S}$, σ_S, S_{rank}

$$\theta = tan^{-1} \frac{\Delta}{t_{i+1} - t_i} \tag{5}$$

θ means the star counting ascension between an interval $t_{i+1} - t_i$. For each course, it is extracted a set $T = \{\theta_1, \theta_2, \ldots\}$ of coefficients of angle. Note that θ varies within interval $[0^o, 90^o)$; high values indicate high star counting growth in two sequential events. Figure 6 illustrates this idea. Course c has six pairs of sequential events, so $T = \{\theta_1, \ldots, \theta_6\}$ with values $45.0^o, 71.6^o, 63.4^o, 0.0^o, 45.0^o$ and 0.0^o, respectively.

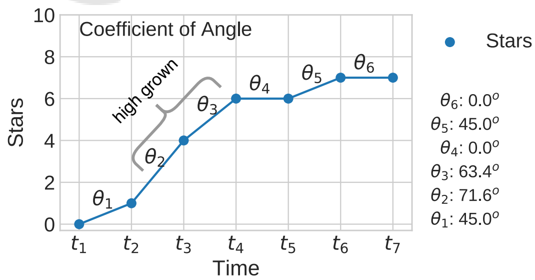


Figure 6: Coefficient of Angle for a star counting stream.

The CA extractor is in Algorithm 3. It represents a player p as the union of T for all its courses (Line 12), so, $T_p = T_1 \cup \ldots \cup T_{|C_p|}$. Then it analyzes T_p to infer features for player p.

In the analysis, we obtain: $\overline{\theta}$ that is the average of values in T_p (Line 13), and; σ_θ that is the corresponding standard deviation (Line 14). We also calculate the player's top-L high growth θ, represented as $\theta_{rank} = [max_1(\theta), \ldots, max_L(\theta)]$ (Line 15).

Algorithm 3: CA Extractor.

1: **procedure** CA-MAP(c) $\triangleright$ Course Modeling
2: Initialize T with $\emptyset$
3: **for** $i \leftarrow 1$ **to** $|c| - 1$ **do**
4: $\Delta \leftarrow stars_{i+1} - stars_i$
5: $\theta \leftarrow tan^{-1}(\Delta)/(t_{i+1} - t_i)$
6: $T \leftarrow T \cup \{\theta\}$
7: **return** T

8: **procedure** CA(p, L) $\triangleright$ Player Modeling
9: Initialize T_p with $\emptyset$
10: **for** $c \in C_p$ **do**
11: $T \leftarrow$ CA-MAP(c)
12: $T_p \leftarrow T_p \cup T$
13: $\overline{\theta} \leftarrow mean(T_p)$
14: $\sigma_\theta \leftarrow std(T_p)$
15: $\theta_{rank} \leftarrow top(T_p, L)$
16: **return** $\overline{\theta}$, σ_θ, θ_{rank}

4.3.4 The F_{ALL} Feature Extractor

Each feature extractor, *i.e.*, LR, DR, and CA, infers a series of features for a player p. They are then combined into a single feature extractor F_{ALL}:

$$F_{ALL}(p,L) = LR(p,L) + DR(p,L) + CA(p,L) \quad (6)$$

The three feature extractors have an input parameter $L \in \mathbb{N}$ that determines the quantity of ranked values to be extracted, each one with its particularities. Note that, whenever L exceeds the number of values available to rank some characteristic of a player, the missing values are assumed to be zeros.

4.4 Framework

This section summarizes our proposed strategy to detect game influencers in Social Networks of Games. *Given the actions of millions of players in an online game, how to detect the game influencers?* It is obviously necessary to model the players' characteristics, *but how can it be done?* So far we have noticed that neither graph G_{dev} nor G_{star} has intrinsic characteristics that are truly useful to spot game influencers, since we need to capture relevant information from the correlated evolution of both graphs over time. Then, we described how to model this correlated evolution using data streams, from which we extract relevant features that properly represent the players' characteristics. Now, we propose to simply express the game influencer detection problem as a classification ML task, using the features of our extractors as input. In the next section, we validate our proposal with the famous SMM game; specifically, we report experimental results to show that our framework detects game influencers of different nations with high accuracy.

5 EXPERIMENTS

We performed a series of experiments to validate our proposal. They are described in the following:

1. In the first experiment, we analyzed each feature extractor individually, considering players from one country only, *i.e.*, Canada;
2. In the second experiment, we analyzed the ensemble of the three feature extractors, also considering Canadian players.
3. We then performed input parameter tuning in the classifiers to improve their results.
4. At last, we validated the best classifier on data from another country, *i.e.*, France, so to demonstrate the generality of our framework.

In the first two experiments, we applied a set of 28 classification algorithms. They are: AdaBoostClassifier, BaggingClassifier, BernoulliNB, CalibratedClassifierCV, ComplementNB, DecisionTreeClassifier, ExtraTreeClassifier, ExtraTreesClassifier, GaussianNB, GaussianProcessClassifier, GradientBoostingClassifier, KNeighborsClassifier, LinearDiscriminantAnalysis, LinearSVC, LogisticRegression, LogisticRegressionCV, MLPClassifier, MultinomialNB, NearestCentroid, NuSVC, PassiveAggressiveClassifier, Perceptron, QuadraticDiscriminantAnalysis, RandomForestClassifier, RidgeClassifier, RidgeClassifierCV, SGDClassifier, and SVC.

The algorithms are implemented in Python 3 with the *sklearn* package. We used the standard input parameter values of each algorithm. For *reproducibility*, our code is open-sourced at GitHub[3], We present how to use it and the performed experiments.

5.1 Dataset Collection

We crawled data from the Super Mario Maker game on the SMM Bookmark website[4] continuously over

[3] GitHub. "Detecting Influencers in Very Large SNG" https://github.com/leomaurodesenv/paper-2019-iceis

[4] SMM Bookmark. https://supermariomakerbookmark.nintendo.net (accessed December 03, 2018).

Table 2: Experiments: results of the top-3 best classifiers in each experiment.

	Algorithm	Accuracy	Precision	Recall	F1-score
LR	DecisionTreeClassifier	0.670 (±0.079)	0.621 (±0.109)	0.645 (±0.100)	0.595 (±0.100)
	BernoulliNB	0.665 (±0.106)	0.432 (±0.145)	0.600 (±0.100)	0.489 (±0.131)
	QuadraticDiscriminantAnalysis	0.665 (±0.106)	0.432 (±0.145)	0.600 (±0.100)	0.489 (±0.131)
DR	DecisionTreeClassifier	0.690 (±0.065)	0.680 (±0.089)	0.658 (±0.070)	0.632 (±0.073)
	ExtraTreeClassifier	0.670 (±0.067)	0.616 (±0.101)	0.653 (±0.075)	0.607 (±0.087)
	GradientBoostingClassifier	0.670 (±0.055)	0.593 (±0.102)	0.613 (±0.058)	0.579 (±0.077)
CA	GradientBoostingClassifier	0.740 (±0.069)	0.745 (±0.068)	**0.766** (±0.076)	0.722 (±0.072)
	BaggingClassifier	0.696 (±0.090)	0.648 (±0.104)	0.675 (±0.106)	0.640 (±0.096)
	ExtraTreeClassifier	0.680 (±0.071)	0.650 (±0.086)	0.687 (±0.091)	0.644 (±0.079)
F_{ALL}	LogisticRegression	**0.808** (±0.108)	**0.808** (±0.108)	0.733 (±0.157)	**0.745** (±0.142)
	RidgeClassifierCV	0.775 (±0.105)	0.733 (±0.131)	0.675 (±0.155)	0.685 (±0.144)
	LinearSVC	0.750 (±0.113)	0.750 (±0.111)	0.683 (±0.154)	0.688 (±0.139)
Tuning	LogisticRegression	**0.8709** (±0.066)	**0.9029** (±0.053)	**0.8588** (±0.080)	**0.8573** (±0.076)
	RidgeClassifierCV	0.8064 (±0.220)	0.7903 (±0.270)	0.8225 (±0.212)	0.7849 (±0.255)
	LinearSVC	0.8387 (±0.094)	0.8596 (±0.097)	0.8306 (±0.103)	0.8266 (±0.099)

a three-months period, collecting information about 74,915 courses, and 884,302 players in which 32,055 are makers (*i.e.*, players who elaborated courses). These courses were developed by Canadian and French makers and received more than 380,000 stars.

The data was split into Canadian makers (CAN dataset) that elaborated 34,479 courses, and French makers (FRA dataset) that elaborated 40,436 courses. The idea is to use the CAN dataset in a train/test step, and the FRA dataset in a validation step.

To make it possible, we had to create ground truth for the CAN dataset. Specifically, we first ranked the top-100 makers from CAN dataset based on the total count of stars received by all of their courses. In this sense, we selected graph $G_{star,f}$ of the last timestamp t_f to get the degree centrality (Barabási and Pósfai, 2016, ch.2) of each course , so it is the stars counting of course c in $G_{star,f}$. Also, graph $G_{dev,f}$ was used to identify the maker's courses.

Then, we manually labeled the top-100 makers into two classes, non-influencer and influential player, by checking their social activities searching the Web with Google and also sites of the communities of games. In total, 41% (41) influential and 59% (59) non-influencer players were identified.

Our consensus to determine whether a player is an influencer is if the player markets/publish its courses in popular sites of the communities of SMM, they are: Reddit, GameFAQs, Twitch, YouTube, Facebook, NintendoLife, and Makers of Mario.

5.2 Feature Extractors Experiments

In the first experiment, we evaluated each feature extractor individually, *i.e.*, LR, DR and CA. Here, the CAN dataset was used with a 5-fold cross-validation strategy. We evaluated the LR extractor with L ranging from 1 to 10. Figure 7 reports accuracy results for the top-6 classifiers. In this interval, the LR ($L = 8$) extracting 20 attributes presented the best average accuracy. The results of the best classifiers for the LR extractor ($L = 8$) are shown in Table 2. The DecisionTreeClassifier algorithm had the best performance with accuracy of 67.0% and f1-score of 59.5%.

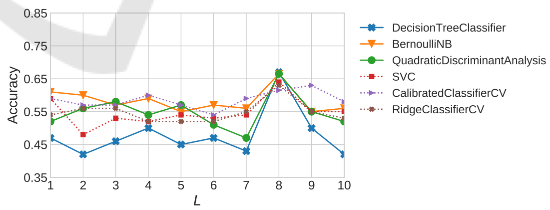


Figure 7: Accuracy of the top-6 classifiers using the LR feature extractor with parameter L ranging from 1 to 10.

Figure 8 reports the accuracy of the top-6 classifiers using the DR extractor with L ranging from 1 to 10. First, we observe that the average accuracy of classifiers grows with L. However, if L becomes higher than the number of values ranked, nonexistent attributes will be filled with zeros. In this sense, L cannot be so great. Second, the experiment shows that $L = 10$ present the best average accuracy with 25 attributes. Table 2 reports the evaluation of the 3 best classifiers for the DR extractor ($L = 10$). The DecisionTreeClassifier had the best performance again, with 69.0% accuracy and 63.2% f1-score.

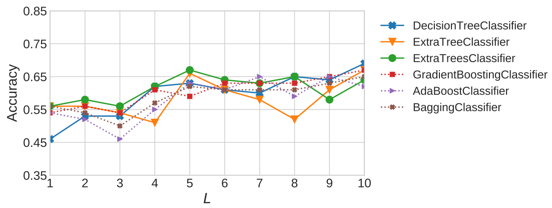


Figure 8: Accuracy of the top-6 classifiers using the DR feature extractor with parameter L ranging from 1 to 10.

At last, we evaluated the CA extractor with L ranging from 1 to 10. Figure 9 reports results from the top-6 classifiers with the highest accuracy. The best average accuracy was obtained with $L = 9$, using 11 attributes only. The results of the 3 best classifiers for the CA extractor ($L = 9$) are shown in the Table 2. Here, we highlight the GradientBoostingClassifier with 74.0% accuracy and 72.2% f1-score.

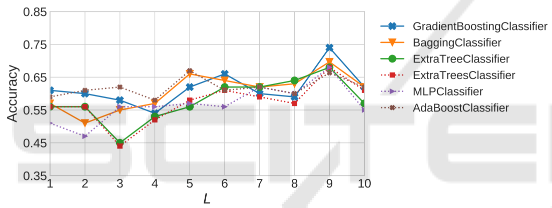


Figure 9: Accuracy of the top-6 classifiers using the CA feature extractor with parameter L ranging from 1 to 10.

In summary, the three feature extractors presented average results in a classification task, with approximately 70% accuracy.

5.3 Framework Experiments

In this section, we analyze the F_{ALL} feature extractor, *i.e.*, the ensemble of the three feature extractors that we proposed. Once more, we used the CAN dataset with a 5-fold cross-validation strategy.

Figure 10 reports the accuracy of the top-6 classifiers using F_{ALL} with L ranging from 1 to 10. In this experiment, $L = 9$ clearly leads to the best average accuracy, using 56 attributes. Table 2 reports results for the 3 best classifiers with F_{ALL} extractor ($L = 9$). We highlight the LogisticRegression with the 80.8% accuracy and 74.5% f1-score.

Note that the framework F_{ALL} presented better results than the feature extractors individually. In addition, the top algorithms that were similar to the feature extractors also changed, as well as the performance of the LogisticRegression. F_{ALL} ($L = 9$) presented great

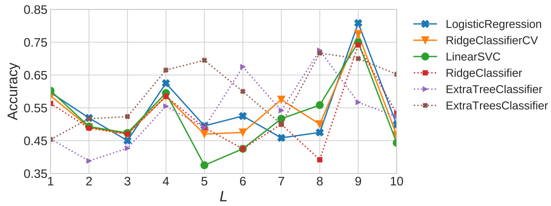


Figure 10: Accuracy of the top-6 classifiers using the F_{ALL} feature extractor with the L ranging from 1 to 10.

results using 56 attributes, meanwhile the LR ($L = 8$), DR ($L = 10$) and CA ($L = 9$) used 20, 25 and 11 features, respectively. However, LogisticRegression using F_{ALL} ($L = 9$) obtained the best accuracy (80.8%), precision (80.8%), and f1-score (74.5%).

5.4 Parameter Tuning

In this step, we focused on improving the top-3 classifiers of the framework experiments ($F_{ALL}(L = 9)$) using a grid search, that is, an exhaustive search through the hyperparameter space of the classifiers.

The grid search on each classifier's parameters, *i.e.*, LogisticRegression (C, $dual$, $fit\_intercept$, max_iter, $penalty$, $solver$), RidgeClassifierCV ($alphas$, cv, $fit\_intercept$, $store\_cv\_values$) and LinearSVC (C, $dual$, $fit\_intercept$, $loss$, max_iter, $penalty$), also used the CAN dataset with a 5-fold cross-validation strategy.

Table 2 reports the best results of each classifier. The LogisticRegression ($C = 0.9$, $dual = True$, $fit\_intercept = True$, $max\_iter = 100$, $penalty = l2$, $solver = warn$) using F_{ALL} ($L = 9$) obtained the best accuracy (**87.1%**), precision (**90.3%**), recall (**85.9%**), and f1-score (**85.7%**). The performance of RidgeClassifierCV and LinearSVC also improved.

5.5 Validation

The validation experiment evaluated the generality of our framework. Here, we used the best configuration, *i.e.*, F_{ALL} ($L = 9$) plus LogisticRegression with the fine tuned parameter values of Section 5.4, taking the whole CAN dataset for training and the FRA dataset for testing. At first, we ranked the top-100 makers from the FRA dataset based on their total number of stars received. Then, we used F_{ALL} ($L = 9$) to infer the players' features and the LogisticRegression classifier to label the game influencers. Thanks to our proposed framework, the algorithm labeled 27 players as game influencers, automatically, in which 21 were manually confirmed as true influencers. Thus, our proposed model presented 77.8% precision to find influencers with a classifier trained in data from another country. These results indicate that our proposal is generic enough to accurately model the behaviour of game influencers from different nationalities.

6 CONCLUSION

This paper presented a novel framework to detect game influencers in Social Networks of Games. *Given the actions of millions of players in an online game, how to detect the most influential users?* It was obviously necessary to model the players' characteristics, but *how can it be done?* To tackle the problem, we needed to capture relevant information from the correlated evolution of more than one dynamic network over time, which could not be performed with the existing works. Then, we described how to model this correlated evolution using data streams, from which we extracted relevant features to properly represent the players' characteristics, and mapped the game influencer detection problem into a classification ML task that uses our features as input. Finally, we validated our proposal by studying the famous Super Mario Maker game, from Nintendo Inc., Japan.

The novel framework includes three feature extractors, *i.e.*, Linear Regression, Delta Rank and Coefficient of Angle. They are unsupervised and based on the temporal aspects of the players' actions on the social network. In the experimental evaluation, 28 classification algorithms were studied. Using our features as input, the LogisticRegression classifier obtained the bests results with accuracy (**87.1%**), precision (**90.3%**), recall (**85.9%**) and f1-score (**85.7%**). We also demonstrated that the proposed framework automatically detects game influencers with high accuracy even when using data from distinct nations for testing and training.

Analyzing Other Types of Social Networks: in theory, our methodology can also be used in non-game-related applications, such as to spot influencers in other types of social networks by analyzing the "likes" received by posts over time. Minor adaptations may be needed, due to domain specificities. Basically, two dynamic networks are the only requirement; one to store the creator of digital content and another to represent positive reactions (*e.g.*, "like") of users to the content, as it is formalized in the paper.

Further Research: (1) analyze influencers' games to discover popular games' characteristics, *e.g.*, platform games' characteristics, such as general monsters, course size, challenges, traps, and so on; (2) apply this framework in other domains to spot influencers; (3) also, use regression modeling to study different degrees of influence over the players.

ACKNOWLEDGEMENTS

This work was supported by the Brazilian National Council for Scientific and Technological Development (CNPq); Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES) [grant 001]; Sao Paulo Research Foundation (FAPESP) [grant 2018/05714-5]; and AWS Cloud Credits for Research.

REFERENCES

Al-Garadi, M. A., Varathan, K. D., Ravana, S. D., Ahmed, E., Mujtaba, G., Khan, M. U. S., and Khan, S. U. (2018). Analysis of online social network connections for identification of influential users: Survey and open research issues. *ACM Comput. Surv.*, 51(1):16:1–16:37.

Barabási, A.-L. and Pósfai, M. (2016). *Network science*. Cambridge university press, Cambridge, USA.

Chino, D. Y. T., Costa, A. F., Traina, A. J. M., and Faloutsos, C. (2017). *VolTime: Unsupervised Anomaly Detection on Users' Online Activity Volume*, pages 108–116. SIAM International Conference on Data Mining.

Gros, D., Hackenholt, A., Zawadzki, P., and Wanner, B. (2018). Interactions of twitch users and their usage behavior. In Meiselwitz, G., editor, *Social Computing and Social Media. Technologies and Analytics*, pages 201–213, Cham. Springer International Publishing.

Hilvert-Bruce, Z., Neill, J. T., Sjöblom, M., and Hamari, J. (2018). Social motivations of live-streaming viewer engagement on twitch. *Computers in Human Behavior*, 84:58 – 67.

Kamber, M., Han, J., and Pei, J. (2012). *Data mining: Concepts and techniques*. Elsevier, Amsterdam.

Lee, J., Lee, M., and Choi, I. H. (2012). Social network games uncovered: Motivations and their attitudinal and behavioral outcomes. *Cyberpsychology, Behavior, and Social Networking*, 15(12):643–648. PMID: 23020746.

Liu, N., Li, L., Xu, G., and Yang, Z. (2014). Identifying domain-dependent influential microblog users: A post-feature based approach. In *AAAI*, pages 3122–3123.

Lucas, S. M. (2009). Computational intelligence and ai in games: A new ieee transactions. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):1–3.

Morone, F., Min, B., Bo, L., Mari, R., and Makse, H. A. (2016). Collective influence algorithm to find influencers via optimal percolation in massively large social media. *Scientific reports*, 6:30062.

Pei, S., Morone, F., and Makse, H. A. (2018). *Theories for Influencer Identification in Complex Networks*, pages 125–148. Springer International Publishing, Cham.

Qi, L., Huang, Y., Li, L., and Xu, G. (2015). Learning to rank domain experts in microblogging by combining text and non-text features. In *2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC)*, pages 28–31.

Savić, M., Ivanović, M., and Jain, L. C. (2019). *Introduction to Complex Networks*, pages 3–16. Springer International Publishing, Cham.

Sjöblom, M. and Hamari, J. (2017). Why do people watch others play video games? an empirical study on the motivations of twitch users. *Computers in Human Behavior*, 75:985 – 996.

Strang, G., Strang, G., Strang, G., and Strang, G. (2016). *Introduction to Linear Algebra*, volume 5. Wellesley-Cambridge Press Wellesley, MA.

Wang, X., Zhang, X., Yi, D., and Zhao, C. (2017). Identifying influential spreaders in complex networks through local effective spreading paths. *Journal of Statistical Mechanics: Theory and Experiment*, 2017(5):053402.

Westaby, J. D. (2012). *Dynamic network theory: How social networks influence goal pursuit.* American Psychological Association.

Yannakakis, G. N. and Togelius, J. (2015). A panorama of artificial and computational intelligence in games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):317–335.

Yannakakis, G. N. and Togelius, J. (2018). *Artificial Intelligence and Games*. Springer International Publishing, Cham.