A Behavioral Pattern Mining Approach to Model Player Skills in Rocket League

Romain Mathonat, Jean-François Boulicaut, Mehdi Kaytoue

A Behavioral Pattern Mining Approach to Model Player Skills in Rocket League

Romain Mathonat
ATOS - Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621, France
romain.mathonat@insa-lyon.fr

Jean-Francois Boulicaut
Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621, France
jean-francois.boulicaut@insa-lyon.fr

Mehdi Kaytoue
Infologic - Université de Lyon
CNRS, INSA-Lyon, LIRIS
UMR5205, F-69621, France
mka@infologic.fr

***Abstract*—Competitive gaming, or esports, is now well-established and brought the game industry in a novel era. It comes with many challenges among which evaluating the level of a player, given the strategies and skills she masters. We are interested in automatically identifying the so called *skillshots* from game traces of *Rocket League*, a "soccer with rocket-powered cars" game. From a pure data point of view, each skill execution is unique and standard pattern matching may be insufficient. We propose a non trivial data-centric approach based on pattern mining and supervised learning techniques. We show through an extensive set of experiments that most of *Rocket League* skillshots can be efficiently detected and used for player modelling. It unveils applications for match making, supporting game commentators and learning systems among others.**

***Index Terms*—Rocket League; Esports; Analytics and Player Modelling; Pattern mining; Supervised Classification**

I. Introduction

Competitive gaming, or esports, is now a well-established phenomena [1]. Online and offline tournaments flourish for hundreds of video games, at any level of player expertise. The most prestigious offer cash prizes up to several US$ millions, and are widely followed on video game live streaming platforms [2], [3]. For the game industry, designing games that can be played as an esport comes with a difficult trade-off between game difficulty and reward/fun. Indeed, a game shall be difficult enough so that professional gamers exhibit extraordinary skills that casual players will enjoy to watch on live streaming platforms. However, the game should also provide an attractive learning curve and clear segments of skills. One should play with/against players of approximately the same level of skills. They finally should also feel a progress in the learning of the game. As popularized by the famous board game Othello's slogan, many esports take "A minute to learn, a lifetime to master". When the trade-off between game difficulty and reward is well handled, a game has better chances to see its life time extended, a major goal for the industry, especially for games with staggering budgets.

Consequently, a major challenge is to understand the level of a player and to evaluate its progression. Match making systems rank players, generally, following an ELO-like scoring [4], and depending only on previous victories and losses of the player. It suffers from the cold-start problem, but most importantly, it does not bring any factual elements such as a player profile of skills and mastered strategies. Furthermore, skills and strategies are not given beforehand - and this is probably one of the reasons why such games are so attractive - but are discovered with time. Fortunately, several games provide access to game logs storing enough information to replay the game. Analyzing these logs enables one to exhibit behavioral patterns, that is, discovering patterns that correspond to strategies and skills. Logs are generated not only for competitive match, but also training sessions, so a profile can be updated with any match.

In this article, we focus on the game Rocket League played as an esport, which can be described as "soccer, but with rocket-powered cars". Released in 2015, it now counts more than 10 millions sales and tens of thousands active players [5]. The game should be played competitively at an international event for the next Olympic Games in Tokyo [6]. Each player controls a car on a soccer field, and has to score just like for classical soccer. The rules are simple, however the control of the car is very precise and requires thousands of hours of play to master. Actually, even years after the release of the game, the community discovers new ways of playing, and new skills that impact the way professionals play. Those skills are recognizable by commentators when games are streamed. Adding a system that can detect them automatically could enhance players ranking, help commentators recognize skills among the spectrum of available ones, or create new game modes based on skills execution (rewarding players given difficulty of skills they executed during the game).

Identifying skillshots from game traces is challenging as each occurrence is unique from a pure data point of view. Therefore, advanced data analysis techniques must be involved. Our contributions are as follows: (i) we provide an original dataset, composed of sequences of *Rocket League* game states (replay), manually labeled with skillshots, and augmented with player actions thanks to a self-made capture program, (ii) we design a non trivial data-centric approach for the automatic detection of skillshots involving discriminant pattern mining and supervised classification, (iii) we support our claims with an extensive set of experiments.

The paper is organised as follows. Section II describes our data and methodology. Section III provides related works. Section IV presents experimental results supporting our claims,

978-1-7281-4533-4/20/$31.00 @2020 IEEE

and Section V discusses them.

II. Data and Methodology

We focus here on building an interpretable prediction model taking a game log in real-time and outputting each time a skillshot is detected. To the best of our knowledge, there exists no dataset of *Rocket League* game logs labeled with skillshots. Thus, we first build a dataset, perform a number of transformations and manually annotate it with skillshots. Then, the resulting dataset is mined and behavioral patterns that strongly characterize skillshots are extracted. Finally, such patterns are used to re-encode the initial dataset, and a model is trained to predict skillshots given unseen player's trace segments. The fact that patterns are used in training enable to produce an interpretable model (in contrast to black box models) which is required to build an interpretable player skill profile. The general workflow is given in Fig. 1.

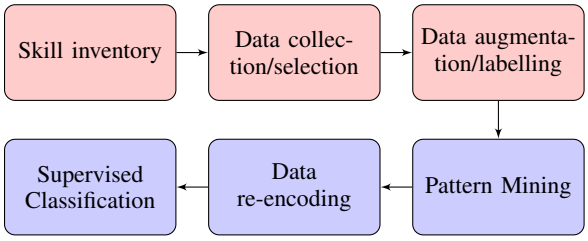
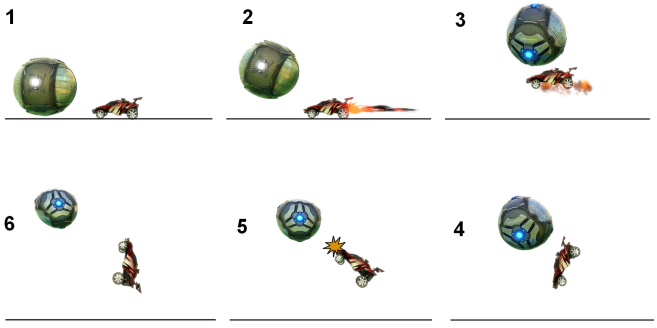


Fig. 1: Methodology (domain expert interventions in red).

A. Skill inventory

This expert knowledge can be found on community websites, e.g., [7]. We focus on the most popular skillshots, namely *Ceiling Shot*, *Power Shot*, *Waving Dash*, *Air Dribbling*, *Front Flick*, and *Musty Flick*. Fig. 2 illustrates the *Musty Flick*: first, the player accelerates and boosts to make the ball roll on top of the car (1, 2), then she jumps making both the car and the ball go up (3). Next, she orientates the car towards the floor (4), jumps again, orientating the car backward (5), resulting in lobbing the ball (6).

Fig. 2: Decomposition of the "Musty Flick"



B. Data collection and Feature Selection

After a match of *Rocket League*, the game client stores a replay file. It is composed of contextual information about the cars and the ball for every frame of the game. It allows to replay the game with the game engine at any moment. Replays can be parsed[1] to extract dozens of variables on the ball and players such as a speed vector, positions vector, rotations vector, rotation speed, each composed of 3 dimensions, and this for each actor on the field (players and balls), several times per second. It results in a large sequence of game states, each containing values for a set of variables, e.g., $\langle\{time : 1.256, P_x : 578, P_y : 5768, P_z : 2245, P_{vx} : 22425, P_{vy} : 15848, P_{vz} : 354, P_{rx} : 0, P_{ry} : 589, P_{rz} : 23, Ball_x : 5588, Ball_y : 789, Ball_z : 22, ...\}, \{time : 1.298, P_x : 7578, P_y : 254, P_z : 4678, P_{vx} : 511, P_{vy} : 555, P_{vz} : 7863, P_{rx} : 6365, P_{ry} : 5665, P_{rz} : 6, Ball_x : 568, Ball_y : 8663, Ball_z : 665, ...\}, \{...\}, ...\rangle$, with P_i being a position of the car in dimension i, P_{si} its velocity, P_{ri} its rotation, etc.

In order to deal with such a number of features, we can reduce it with feature selection/engineering, using expert knowledge. As an example, using the information of the position of a static boost pad (an item on the field giving boost to players) will not help to classify the action the player is performing. The list of relevant information is given in Table I. The wall distance, ceiling distance and ball distance can easily be computed using positions of the ball and players by retro-engineering the positions of the walls and ceiling in replays of games already played.

Contextual information	Data type
Wall distance (not backboard)	Numeric
Ceiling distance	Numeric
Ball distance	Numeric
Ball speed	Numeric
Ball acceleration	Numeric
Car speed	Numeric
Goal Scored	Boolean

TABLE I: Contextual information selected by the expert

C. Data augmentation

Unfortunately, *Rocket League* replay files contain only contextual information on the cars and the ball (position, speed, ...), but do not contain player inputs (*turn left*, *accelerate*, ...). To overcome this limitation, during each game, we also gathered player inputs. To do so, we executed a program listening to the game controller (joystick) to detect sequences of buttons, say inputs, a player presses. It then generates a sequence of inputs, for example: $\langle\{accelerate, boost, time : 1.2\}, \{accelerate, right, boost, time : 1.25\}, \{jump, time : 1.34\}, \{accelerate, up, left, time : 1.6\}, \{accelerate, jump, down, boost, time : 1.7\}, ...\rangle$.

This sequence is then merged with the contextual sequence. Indeed, the sequence of player inputs alone is not enough

[1] https://github.com/jjbott/RocketLeagueReplayParser

to tell if a skill has been executed. For example, entering the sequence of inputs for a "Power shot" when the ball is far away will just result in flipping the player. On the other hand, contextual information alone does not provide player inputs, which are required to produce interpretable behavioral patterns.

The workflow of the data augmentation process is given in Fig. 3. The sequence of contextual information (obtained from the game replay on which feature selection is operated) and the sequence of player inputs are merged. Then, for improving the computational efficiency of the next steps, we filter out a sequence state X_i (built in the previous subsection) if its previous state X_{i-1} has the same inputs. Indeed, behavioral patterns will be built only on states resulting from player's actions.

An expert then labels sequences with skillshots performed using the in-game replay viewer: she defines the beginning and the end of sequences corresponding to a particular skill, leading to the creation of our labelled dataset. Sequences are then split according to the beginning and the end chosen by the expert (the end is often set when a goal is scored). A simplified example of a labeled sequence is the following (a for *accelerate*, DW for *DistanceWall* etc.): $\langle(\{a,b\},\{Time:1.2,DW:2131,...\}),(\{\{a,r,b\},\{Time:1.25,DW:1801,...\}),(\{j\},\{Time:1.34,DW:1325,...\}),(\{a,u,l\},\{Time:1.6,DW:600,...\}),(\{a,j,d,b\},\{Time:1.7,DW:233,...\})\rangle, figure:MustyFlick$.

Note that those steps that may appear straightforward as described here are in fact technically complex. As replays are meant to be replayed and not to extract data, a lot of retro engineering is necessary to adjust x, y, z orientations, and to synchronize sequences from replays and players actions inputs. Note also that when visualizing the replay for labelling, the time is a bit faster than the real play, leading to difficulties of synchronization with timing of inputs that we corrected thanks to a linear regression. Details can be found in the supplementary materials, see Section V.

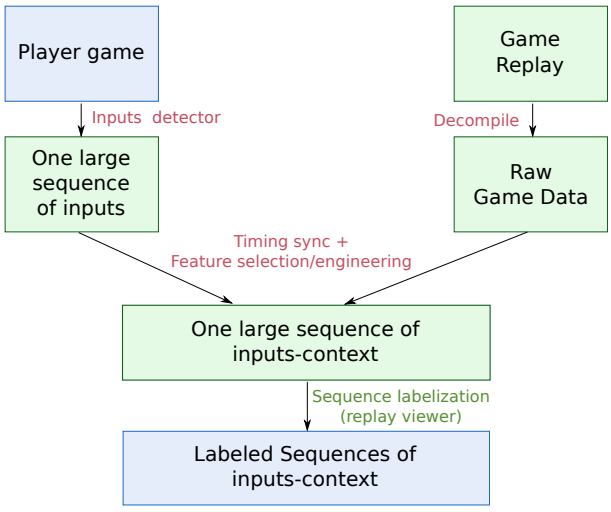


Fig. 3: Data augmentation (expert intervention in blue)

D. Discriminative pattern mining

Using discriminative pattern mining [8] for sequences has shown to give relevant patterns in the case of game analytics (e.g., [9]). In this subsection, we first formally introduce an original behavioral pattern mining technique, put then in practice with an adaptation of an existing algorithm.

Definition 1 (Complex Event Sequence): Let $\mathcal{I}$ be *a set of items*. Each subset $I \subseteq \mathcal{I}$ is called an *itemset*. A complex event sequence is an ordered list of states $s = \langle X_1 ... X_n \rangle$ where each state $X_i = (t_i, I, N)$ is composed of a timestamp t_i, an itemset I and a list of numerical valued variables $N \times \mathbb{R}$. Note that each state is composed of a list of the same numerical variables.

Each labeled sequence produced in the previous step can be represented as a complex event sequence (Table II).

Definition 2 (Behavioral pattern): A *behavioral pattern* is a complex event sequence generalization and can be written as an ordered list of states $s = \langle X_1 ... X_m \rangle$ where each state $X_i = (I, N)$ is composed of an itemset I of player actions and a list of numerical interval valued variables $N \times [a, b]$ $with$ $a, b \in \mathbb{R}$ denoting the contextual information ranges of the event.

It is precisely the ranges of contextual variables (intervals) that enable behavioral patterns to grasp slight variations of the different executions of a skillshot, along with common subsets of player actions. Patterns thus represent skillshot generalizations, resistant to noise and variations of execution.

Definition 3 (Behavioral pattern extent and support): Given a behavioral pattern $p = \langle X_1 ... X_m \rangle$, and a database $\mathcal{D}$ of complex events sequences, the *extent* of a pattern p in $\mathcal{D}$ is $ext(p) = \{s \in \mathcal{D} \mid p \sqsubseteq s\}$ that is, the set of sequences that p generalizes. The *support* of a pattern p is $supp(p) = |ext(p)|$.

Definition 4 (Subsequence): A behavioral pattern $p = \langle X_1 ... X_{m_p} \rangle$ is a *subsequence*, or a generalization, of complex event sequence $s = \langle X'_1 ... X'_{m_s} \rangle$, denoted $p \sqsubseteq s$, if and only if there exists $1 \leq j_1 < ... < j_{m_p} \leq m_s$ such that $X_1 \subseteq X'_{j_1}, ..., X_{m_p} \subseteq X'_{j_{m_p}}$. Here, $X_i \subseteq X_j$ means that $A_i \subseteq A_j$ and that $\forall [a_i, b_i]_k \in N_i, n_{jk} \in N_j, a_{ik} \leq n_{jk} \leq b_{ik}$

The pattern $\langle(\{a\},\langle[1,3],[2,5]\rangle),(\{b\},\langle[2,3],[4,5]\rangle)\rangle$ is a generalization of the complex event sequence $\langle(\{a,b\},\langle 2,2\rangle),(\{c\},\langle 7,0\rangle),(\{b\},\langle 3,5\rangle)\rangle$.

As the number of patterns grows exponentially w.r.t. the number of complex event sequences, we focus on patterns that discriminate skillshots, that is, whose support sequence are strongly correlated to a skillshot. This is done thanks to a popular score for discriminative pattern mining [8].

Definition 5 (Weighted Relative Accuracy (WRAcc)): Given a pattern p, we have:

$$WRAcc(p, c) = \frac{supp(p, \mathcal{D})}{|\mathcal{D}|} \times \left(\frac{supp(p, \mathcal{D}_c)}{supp(p, \mathcal{D})} - \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \right)$$

with $\mathcal{D}$ being a dataset of complex event sequences labeled with a class in $\{+, -\}$ and $\mathcal{D}_c$ being the subset of $\mathcal{D}$ composed of complex event sequences labeled by $+$. The higher the WRAcc, the better.

Given the arbitrary pattern, using only player inputs for simplification, $p = \langle(\{a,b\}),(\{j,d\})\rangle$ and the data $\mathcal{D}$ given in

Table II, we have $extent(p) = \{1, 3\}$, $support(p) = 2$, and $WRAcc(p, +) = \frac{2}{4} \times (\frac{2}{2} - \frac{2}{4}) = 0.25$. Roughly speaking, the $WRAcc$ favors patterns that mostly cover the positive class and not the negative one. Notice that we have more than two labels for our skillshot classification task. We consider a one versus all scheme, i.e., we will focus on a target class, say the positive class while all the others will be merged to build the negative one.

TABLE II: Toy dataset. "+" means the sequence is a "Musty Flick". We only included player inputs for readability.

id	Sequences	class
1	{a,b}, {a,r,b},{j}, {a,u,l}, {a,j,d,b}	+
2	{a,r}, {j}, {j}, {l}	-
3	{a,b}, {a,b},{a,j}, {a,u}, {r}, {j,d}	+
4	{a,b}, {a,b},{a,j}, {j,u}	-

There exists no mining algorithm for behavioral patterns as defined in this article. However, we recently introduced the `SeqScout` algorithm [10] that can mine discriminative patterns in sequences of itemsets. In the following, we first present the original version of `SeqScout` and then its slight adaptation to mine behavioral patterns.

a) SeqScout: The principle of `SeqScout` is explained on Fig. 4. The root of the search space, at the top, is the most general pattern, meaning that it is the pattern that covers, i.e. is a subsequence of, all sequences of the dataset. The more we go down in the search space, the more specific patterns are, covering less elements. When reaching the bottom, patterns are in fact so specific that they are, for most of them, directly a sequence of the dataset. The idea of `SeqScout` is to iteratively select a sequence of the dataset following a trade-off between exploration and exploitation, using UCB [11], and then to generalize this element ("going up" in the search space) creating a new pattern. The quality of this pattern, i.e., its discriminating power, is then computed with the chosen quality measure, the $WRAcc$ in our case. Once the time budget has been reached, patterns are filtered to make sure they are non-redundant following Jaccard index, using a parameter θ (see [10], and the top-k are returned. To adapt this algorithm to our problem, we need to reconsider the generalisation step as complex event sequences also contain vectors of intervals.

b) Sequence generalisation: The sequence generalisation consists of two steps. In the first, each itemset of inputs I is considered, for the selected sequence. Each item in I is removed following the rule:

$$\begin{cases} remain, & \text{if } z < 0.5 \\ remove, & \text{if } z \geq 0.5 \end{cases}, \text{ where } z \sim \mathcal{U}(0, 1).$$

If I is empty, the entire corresponding state X is removed. Then in the second step, each numerical variable is considered. Each $n \in N$ is mutated following the rule, given $n_{left} \in Dom(n)$ s.t. $n_{left} \leq n$, $n_{right} \in Dom(n)$ s.t $n_{right} \geq n$ and $\alpha \in [0, 1]$, where $Dom(v)$ represents the set of values taken by variable v in the dataset:

$$\begin{cases} [-\infty, \infty], & \text{if } z < \alpha \\ [n_{left}, n_{rigth}], & \text{if } z \geq \alpha \end{cases} where\ z\ \sim \mathcal{U}(0, 1)$$

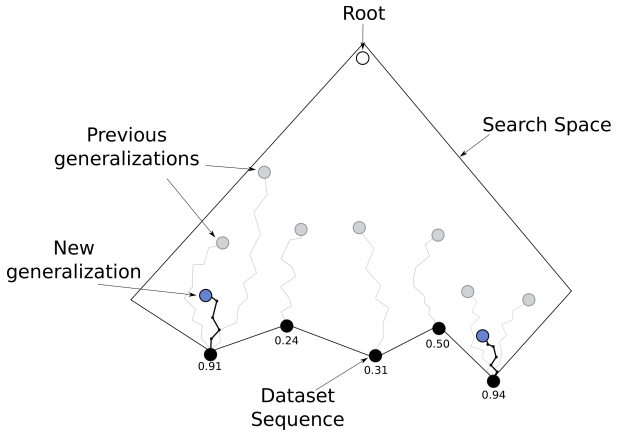


Fig. 4: `Seqscout` principle

For example, considering the sequence:

$$\langle (1, \{jump\}, \{speed = 158, DistanceBall = 10\}),$$
$$(2, \{right, slide\}, \{speed = 102, DistanceBall = 29\}) \rangle$$

One possible generalisation is:

$$\langle (2, \{slide\}, \{speed = [88, 107], DistanceBall = [-\infty, \infty]\}) \rangle$$

Removing constraints of random variables leads to the creation of patterns with restrictions only on a subset of variables from the state. It helps to find more interesting patterns faster.

E. Dataset re-encoding

Once a set of patterns has been extracted, we re-encode the dataset, in the same way as authors in [12], [13]. We create a feature for each pattern, putting a Boolean "1" value if the pattern appears in the sequence, "0" otherwise, as illustrated in Table III. As explained in [13], "The binary feature construction process is certainly the most straightforward but has also shown good predictive performance". As we know that those features are discriminative, they will give good insights about the class we need to predict. Note that we need to mine discriminative patterns of each class to classify all of them.

TABLE III: Toy dataset re-encoded with 3 patterns

id	Pattern 1	Pattern 2	Pattern 3	class
1	1	1	0	+
2	0	0	1	-
3	1	1	0	+
4	1	0	1	-

F. Classification

Once the dataset in transformed into a binary transaction labeled dataset, classical machine learning algorithms can be used to predict the skillshot the player is performing.

III. Related Work

Understanding and analyzing players behavior is an important subject. For unlabeled data, time series clustering approaches were applied to different games, on free-to-play

game data to find relevant patterns in [14], or to discover seasonal patterns in [15].

Concerning labeled data, we take interest in the identification of "skills", or "moves". In games, it is most of the time done with expert knowledge and ad-hoc techniques. For example, in fighting games, multiple precise combinations of controller inputs lead to the execution of particular attacks or combos. To do so, exact pattern matching is used [16]. However in a game like *Rocket League*, such an approach cannot be used. Indeed, each player has a full and precise control of her car, in a 3D space, including all possible rotations, speed, acceleration, even the reaction of bumping other cars. Having such a huge space of possible configurations leads to the fact that each performance of a skill is unique: positions, speeds, rotations of the car and the ball will not be the same, and even inputs pressed by the player can vary a lot (many micro-adjustments of trajectory). This makes the problem of automatically classifying (or detecting) the action of the player difficult. *Rocket League* seems to implement a system that uses ad-hoc rules such as "if the ball touched the back of the car and then scored, it is a backward goal". However, this approach is limited to simple rules, based purely on expert knowledge, that would not work on more complex skills.

In our context, we need to take into account inputs of the player, to know what she wanted to perform, but we also need the contextual information of the game. For instance, performing a "front flip" in its own goal without touching the ball and doing it to hit the ball will impact the game differently. We are then in the presence of the so called complex event sequences [17]. Here is an example of the kind of sequence we are talking about: $S = \langle X_1 ... X_m \rangle$, $X_1 = (t_1, \{a, b\}, \langle 1000, 22, 58 \rangle)$, $X_2 = (t_2, \{c\}, \langle 125, 27, 101 \rangle)$.

In fact, we can transform the event part of the sequence in booleans taking 0 values by default, and 1 when the event fires. It reduces the problem to a multivariate timeseries classification problem [18]. Most of the recommended methods require timeseries to have the same length, whether it is BOSS [19], COTE [20], Shapelet Transform [21], or Time Series Forest [22]. In our case here, timeseries have different lengths. The 1-Nearest Neighbor Dynamic Time Warping (1-NN DTW) [23] is said to be a good baseline [18]. However, the prediction step in KNN can take a long time, even more because the DTW distance has a complexity of $O(n^2)$ for sequences of size n. It would make a good candidate to compare to in our experiments though.

Finally, it should be highlighted that we were not able to find any data analytics article on *Rocket League* in the literature. It strengthens our first contribution: it appears useful to provide a dataset mixing contextual information and player actions, along with the needed tools to create new datasets.

IV. Experiments

The whole methodology has been fully implemented and a thorough experimental study has been carried out. All materials are publicly available as detailed in Section V, including our original dataset, scripts, algorithms, and our self-made program for capturing inputs from joysticks. In this section, we discuss a number of experiments that indicate to which extent our methodology can automatically detect skillshot from previously unseen games.

A. Dataset

Following the presented workflow, we generated data with the help of experienced players of the game. We describe the resulting dataset in Table IV. The *max. size* corresponds to the maximum number of states in a sequence. We recall that each state is composed of 7 variables and between 1 and 11 events, creating long sequences nearly impossible to read for a human. Note that during labelling step we added failed skills, i.e., missing goals, or not hitting the ball while performing the skill, and random non-skill sequences in games as "noise". Indeed, to train our classifier to perform in real conditions, we need to be able do detect when player fails. We also added classes distribution in Table V. The min and max values of the variables are given in Fig. 10.

B. Experimental setup

We ran the first series of experiments not reported here to determine the best default parameters for building a reasonable classifier: we will study how each parameter affects the results when other parameter are fixed to their default value. As such, when not specified, the default parameters in experiments are the following. We used 5-fold stratified cross validation to assess the robustness of our classifier. We took the top-5 best patterns given by `SeqScout`, on each class of the dataset, as features for the classifier. We gave 1,000 iterations for `SeqScout`, which seemed a reasonable trade-off between pattern quality and time taken by the algorithm. The non-redundancy parameter θ is set to 0.8. The parameter α, controlling the sequence generalisation, is set to 0.9, and the Decision Tree (from [24]) is chosen for the final classification step, for its simplicity and interpretability. Section V gives details on access to supplementary materials.

With this experimental setup, we provide answer elements to the following questions.

- How many patterns are required to maximize the accuracy of our classifier?
- How does the α parameter affect patterns quality?
- How does the qualities of computed patterns affect the quality of the classification and how many iterations does `SeqScout` need to perform well?
- How does the non-redundancy of patterns affect the classification?

TABLE IV: Dataset

# Sequences	# Inputs	# Variables	Max. size	# Classes
298	11	7	64	7

TABLE V: Class Distribution

Noise	Ceiling	Power	Waving	Air	Flick	Musty
43	30	60	38	45	46	36

- What is the best classification method to use?
- What is the performance of our method compared to a state-of-the-art algorithm like 1-NN DTW?
- Using only sequences of variables (and not player inputs), can we still accurately predict performed skills?
- How can we use our classifier in real-time to detect performed skills?

C. Influence of the number of mined patterns

We evaluated how the number of mined patterns influences the mean accuracy of our method. Note that the pattern number corresponds to the number of mined patterns for each class with `SeqScout`. For example, this means that for 20 mined patterns, having 8 classes in our dataset, we re-encode our dataset with 160 features. Results are given in Fig. 5. Here we can see that the predictive power of our method quickly increases with the number of patterns, and then tends to decrease a bit. Setting this parameter between 10 and 30 seems to be a reasonable choice.

D. Influence of α parameter

In Fig. 6, we tested the influence of the parameter α from the generalisation step on the quality of patterns. Note that the $WRAcc$ takes its values in $[-0.25, 0.25]$ (see [10] for more information). Here we can see that we have an optimum for $\alpha = 0.8$. This means that the method gives better results when we remove restrictions on a numeric with the probability 0.8.

E. Pattern quality w.r.t. accuracy

Increasing the time budget leads to an increase of the mean pattern quality [10]. We then propose to evaluate the impact of the time we give to `SeqScout`. As we can see in Fig. 7, the more iterations we give (and equivalently, time), the better are the predictions. Finding good discrimative patterns leads to the design of good discriminative features.

F. Impact of diversity on accuracy

In Fig. 8, we show the impact of the θ parameter. We recall that θ is the threshold above which patterns are considered similar when filtering resulting pattern. Having a low θ means we want `SeqScout` to give us patterns which have few or no sequences in common, and a high θ that we accept similar patterns as the output. Interestingly, this parameter does not seem to have an impact on the accuracy. It is important if we want to filter patterns to an end-user, to show non-redundant results, but in the case of classification, we can choose to remove this costly step without impacting the prediction quality.

G. Predictive performance of the method

We evaluated the performance of different state-of-the-art classification methods on the re-encoded dataset. In light of preceding results, we chose here the best found parameters: $\alpha = 0.8$, $\theta = 1$, 20 extracted patterns, and a number of iteration of 10,000. We tested Decision Tree (DT), Random Forest (RF), SVM, Naive Bayes from sk-learn [24] and XGBoost (XGB) from [25], all with default parameters values.

TABLE VI: Comparaison of 1-NN DTW vs our method

	Acc Train	Acc Test	Time Train	Time Test
Best Approach	94.1	84.9	14 min	-
1-NN DTW	-	71.5	-	7 min

As we can see in Fig. 9, RF, XGboost and SVM seem to give the best results.

H. Comparison to 1-NN DTW

If we transform events to booleans taking 1 values when event fires, and 0 otherwise, we can reduce the problem to a multivariate timeseries classification. We then compare ourselves to 1-NN DTW, without forgetting to z-normalize timeseries as specified in [18]. We used a DTW implementation[2]. Results are shown in Table VI. Our method improves significantly the accuracy. Moreover, we have the strong advantage of being able to classify in nearly-real time, contrary to 1-NN DTW. This is important to create a system that could classify directly in game what players are doing. Note also that there is a trade-off between the duration of the training and the accuracy of our method, as we can tune the number of iterations we give to `SeqScout`.

I. Using numerical variables only

We tried the approach on sequences of purely numerical variables, meaning that we removed the player inputs information. In a 5-fold stratified cross-validation, we found a mean accuracy of 73.9%. We can then deduce that the information of players inputs leads to a clear increase of accuracy of the system. It supports the need for the proposed data augmentation step.

J. Classify goals

An example of confusion matrix given in the stratified 5-fold cross-validation is given in Table VII. True classes are on the left, predicted are on the top. As we can see, the classifier

TABLE VII: Confusion matrix of our classifier

		Predicted						
		Noise	Ceiling shot	Power shot	Waving Dash	Air Dribble	Flick	Musty Flick
Actual	Noise	12	38	0	0	0	38	12
	Ceiling shot	0	100	0	0	0	0	0
	Power shot	0	0	100	0	0	0	0
	Waving Dash	0	0	0	100	0	0	0
	Air Dribble	0	0	0	0	100	0	0
	Flick	10	0	0	0	0	90	0
	Musty Flick	0	0	12	0	0	0	88

has difficulties to classify the "noise" class, composed of failed goals, and random part of games. Using sliding windows to detect skills in real time in a game would then probably give poor results. However, in real setting, the vast majority of

[2]https://github.com/pierre-rouanet/dtw

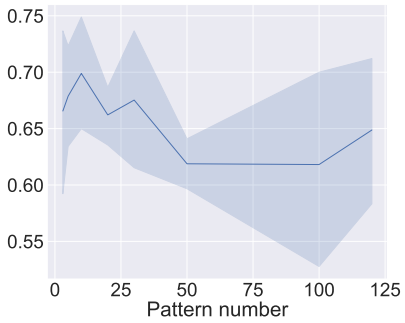


Fig. 5: Mean accuracy w.r.t. the number of patterns

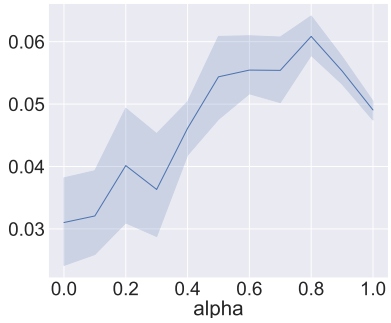


Fig. 6: Mean WRAcc vs α

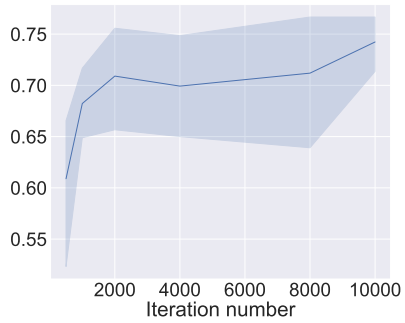


Fig. 7: Accuracy w.r.t. the number of iterations

Fig. 8: Accuracy w.r.t. θ

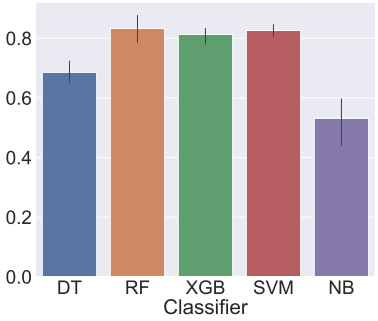


Fig. 9: Comparison of accuracy of different classifiers

Numeric	Min	Max
BallAcceleration	-319123	294344
Time	0	13.4
DistanceWall	0	4043
DistanceCeil	0.07	2020
DistanceBall	115	9509
PlayerSpeed	27	229999
BallSpeed	0	329814

Fig. 10: variables min and max values

figures are ways of scoring goals. To deal with the previous issue and to increase the accuracy of our system, we can introduce a bit more of expert knowledge: we classify the sequence only if a goal has been scored. This way, our system could be directly used in real settings, extracting the sequence of actions before a goal, and feeding it to our classifier. After filtering sequences of our dataset by keeping only those having a "goal" in it, and using our method, we have a mean accuracy of **87.6%**.

K. Pattern interpretability

One of the interest of using a pattern mining approach is its interpretability. The two following patterns are the top-1 patterns extracted respectively for the "Musty Flick" and the "Ceiling Shot" by `SeqScout`. Note that here for simplicity of notation we removed variables whose intervals were $[-\infty, \infty]$.

$$\langle \{accelerate\}, \{down\ jump\}, \{goal\} \rangle \ (WRAcc = 0.0863)$$
$$\langle \{accelerate, jump, DistanceCeil = [1.52, 1233.51]\} \rangle$$
$$(WRAcc = 0.0755)$$

Interestingly here, the pattern corresponding to the "Musty Flick" only takes into account the inputs of the user, with no restrictions on variables. The sequence of inputs is indeed a part of the required inputs to perform the "Musty Flick". The pattern is only a "part of" the required inputs because in fact the beginning of the sequence of inputs for the "Musty Flick" is common to the "Front Flick": `SeqScout` returns only the discriminative part, i.e., the part that is present in "Musty Flick" figures, and not in others. That is why this method gives good results: the more discriminative patterns are, the better are the feature for the classification step.

Moreover, the best found pattern discriminative of the "Ceiling Shot" is in fact composed of only one state, which is enough to discriminate the skill here: if the player jumps when she is near the ceiling (1,200 corresponds approximately to the medium position between ceiling and floor), it often leads to a goal, in our dataset.

V. Discussion

We introduced an original dataset collected by our means, and a method to classify skillshots of players in *Rocket League*. This methodology could be applied to other games, as the type of required data is generic enough. We obtained good results with a mean accuracy of 87.6%, a high score for a multi-class classification problem.

The classifier training can take some time, depending on the time budget the user want to allocate, but the classification can be performed in real-time, contrary to 1-NN DTW.

However, there also exists some drawbacks that could be addressed in future works. The biggest one is the noise problem, as it is difficult to discern failed skills from well

executed ones. We dealt with this issue by filtering only sequences finishing with a goal. This approach would not work in the case of classifying skills that do not finish with a goal. Moreover, all existing skills are not present in this dataset, as it would require a lot of additional work with several different experts to increase the diversity of performed and labeled skills.

One may notice when looking at the confusion matrix that many examples of noise are considered as other figures by the classifier. This is due to the fact that noise is composed of different "failed" figures. As they are similar to correctly performed figures, they are difficult to discern. To deal with that, a possibility would be to add a new label for each figure corresponding to its failed or succeeded execution, and to reserve the "noise" class to random moves. Our classifier would then give the intention of the player, and another classifier would give the success or not of this intention.

Note also that this project is a proof-of-concept, that used some "hacks" in the data collection process. A production-ready system would require inputs data directly recorded by the game. In the case of new skill appearing in-game, the system would need to be trained again with new data, to keep being up-to-date with the new meta game.

Some skillshots are also not present in the dataset because our expert players cannot perform them consistently. Indeed, the better the player, the more consistent she is at performing skills, as already showed in [9], and so the easier it is to classify (professional players have less variability when repeating strategies). Moreover, at very high level, it is not only one skill at a time that is performed, but more a sequence of skills. Using a system like presented in this paper would be beneficial to several actors. For players, a histogram of skills detected during games could be used for player profiling. For game editor, such a system may help to better rank them, or to detect "smurfing", i.e., playing with another account to improve her ranking [26]. It could also improve analytics for esport structures to better know their future opponents by better understanding their play-style. It would also be interesting to create new game modes where the goal is to perform skills on increasing difficulty (like the game of "horse" in basketball). Finally, this type of analytics could help to study the evolution of the meta-game during seasons, as *Rocket League* is already a game with a history, and is probably going to keep growing over the next years.

Supplementary materials

All our experiments, data collection workflow and data are provided with the code of our solution online: https://github.com/Romathonat/RocketLeagueSkillsDetection.

References

[1] T. L. Taylor, *Raising the Stakes:E-Sports and the Professionalization of Computer Gaming*. MIT Press, 2012.

[2] M. Kaytoue, A. Silva, L. Cerf, W. Meira, and C. Raïssi, "Watch me playing, i am a professional: A first study on video game live streaming," in *WWW (companion volume)*, 2012, p. 1181–1188.

[3] T. L. Taylor, *Watch Me Play: Twitch and the Rise of Game Live Streaming*. Princeton University Press, 2018.

[4] R. Herbrich, T. Minka, and T. Graepel, "Trueskill(tm): A bayesian skill rating system," in *Advances in Neural Information Processing Systems 20*. MIT Press, January 2007, pp. 569–576.

[5] Wikipedia, *Rocket League*, Accessed March 28, 2020, https://en.wikipedia.org/wiki/Rocket_League.

[6] Intel, *Intel World Open*, Accessed March 28, 2020, https://www.intelworldopen.gg/rocket-league/.

[7] Reddit, *All Rocket League moves / skills with descriptions*, Accessed March 28, 2020, https://www.reddit.com/r/RocketLeague/comments/adiu96/all_rocket_league_moves_skills_with_descriptions/.

[8] P. K. Novak, N. Lavrač, and G. I. Webb, "Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining," *J. Mach. Learn. Res.*, vol. 10, p. 377–403, Jun. 2009.

[9] G. Bosc, P. Tan, J.-F. Boulicaut, C. Raïssi, and M. Kaytoue, "A pattern mining approach to study strategy balance in RTS games," *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 9, no. 2, pp. 123–132, 2017.

[10] R. Mathonat, D. Nurbakova, J.-F. Boulicaut, and M. Kaytoue, "Seqscout: Using a bandit model to discover interesting subgroups in labeled sequences," in *Proceedings IEEE International Conference on Data Science and Advanced Analytics DSAA*, Oct 2019, pp. 81–90.

[11] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, May 2002.

[12] L. Diop, C. Diop, A. Giacometti, D. Li, and A. Soulet, "Sequential pattern sampling with norm-based utility," *Knowl. Inf. Syst.*, october 2019, in Press.

[13] E. Egho, D. Gay, M. Boullé, N. Voisine, and F. Clérot, "A user parameter-free approach for mining robust sequential classification rules," *Knowl. Inf. Syst.*, vol. 52, no. 1, p. 53–81, Jul. 2017.

[14] A. Saas, A. Guitart, and A. Periáñez, "Discovering playing patterns: Time series clustering of free-to-play game data," in *Proceedings IEEE Conference on Computational Intelligence and Games*, Sep. 2016, pp. 1–8.

[15] D. Vihanga, M. Barlow, E. Lakshika, and K. Kasmarik, "Weekly seasonal player population patterns in online games: A time series clustering approach," in *IEEE Conference on Games*, 2019, pp. 1–8.

[16] G. L. Zuin, Y. P. Macedo, L. Chaimowicz, and G. L. Pappa, "Discovering combos in fighting games with evolutionary algorithms," in *Proceedings ACM Genetic and Evolutionary Computation Conference GECCO*, 2016, p. 277–284.

[17] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explor. Newsl.*, vol. 12, no. 1, p. 40–48, 2010.

[18] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Min. Knowl. Discov.*, vol. 31, no. 3, pp. 606–660, 2017.

[19] P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," *Data Min. Knowl. Discov.*, vol. 29, no. 6, p. 1505–1530, 2015.

[20] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with cote: The collective of transformation-based ensembles," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2522–2535, Sep. 2015.

[21] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Min. Knowl. Discov.*, vol. 28, no. 4, pp. 851–881, July 2014.

[22] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Information Sciences*, vol. 239, pp. 142 – 153, 2013.

[23] A. Mueen and E. Keogh, "Extracting optimal performance from dynamic time warping," in *Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, p. 2129–2130.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[25] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[26] O. Cavadenti, V. Codocedo, J.-F. Boulicaut, and M. Kaytoue, "When cyberathletes conceal their game: Clustering confusion matrices to identify avatar aliases," in *Proceedings IEEE International Conference on Data Science and Advanced Analytics DSAA*, 2015, pp. 1–10.