Machine Learning Techniques for Mobile Application Event Analysis

Ben Falchuk, Chris Mesterharm, Euthimios Panagos
Applied Communication Sciences (Vencore Labs)
Basking Ridge, NJ, USA
e-mail: {bfalchuk,jmesterharm,epanagos}@appcomsci.com

Shoshana Loeb
InterDigital Inc.
Wilmington, DE, USA
e-mail: Shoshana.Loeb@InterDigital.com

***Abstract*—As increasing amounts of economic, entertainment and social activities are occurring using native and web applications, it has become essential for developers to analyze user interactions in order to better understand their behavior and increase engagement and monetization. In this paper, we describe how JumpStart, a real-time event analytics service, utilizes machine learning techniques for empowering developers and businesses to both identify users exhibiting similar behavior and discover user interaction patterns that are strongly correlated with specific activities (e.g., purchases). Discovered interaction patterns can be used for enabling contextual real-time feedback via JumpStart's complex event pattern matching.**

Keywords - Machine Learning; Analytics; Big Data; Mobile Apps; Data Clustering;

I. INTRODUCTION

Organizations offering mobile applications of all kinds (e.g., games, social media, etc.) are presently interested in learning about how users interact with these applications for reasons that range from marketing and advertising to application improvements and user retention. The most common way to acquire these insights is through the collection and analysis of events corresponding to specific actions users take while using an application. Event collection is typically achieved by using a home-grown or third-party library (e.g., software development kit – SDK – such as those offered by Google, Facebook, and Yahoo) for instrumenting the application code to record information related to specific user actions (e.g., kill an opponent in a first-person shooter game). Recorded information is then periodically uploaded to a back-end system for processing and analysis.

Many of today's mobile event processing/analytics offerings expose computed statistical information relating to key performance indicators (KPI) (e.g., daily active and new users, day 1 retention, and total sessions) in the form of reports and visualizations using a small number of pre-defined events or event attributes. These same offerings may also expose limited querying interfaces for users to filter the collected event data or offer the ability to download the collected information for further analysis by business intelligence and data mining tools.

A major limitation of such offerings is that they lack automated techniques for discovering interesting event patterns in collected events. In addition, they are often limited in their ability to match and react to such event patterns in real time. (e.g., notify users about a YouTube video that offers hints on how to complete the current game level after they fail three times).

The JumpStart [1] real-time context-aware analytics service overcomes these shortcomings by offering an end-to-end solution that enables application developers to capture events (via a simple SDK), specify event patterns (via a web portal) that will be matched in real-time as events are streamed to the service, and associate specific information to be sent back to the application on event pattern matches. JumpStart employs machine learning techniques to automatically learn user behavior in the context of specific applications. Such deep analysis allows, for example, for the discovery of otherwise hard-to-detect issues that may be limiting user experience (e.g., specific levels in a gaming application may be far too hard for many users).

In this paper, we focus on JumpStart's machine learning component. In particular, we discuss how JumpStart applies supervised and unsupervised machine learning technique to application event streams for: (1) identifying users exhibiting similar application behavior and (2) discovering event patterns that are strongly correlated with specific user activities within an application. These techniques enable JumpStart customers (e.g., application developers and businesses) to understand and control many aspects of their application. For example, customers can specify event patterns that, when matched against received event streams, provide contextual information back to the application to: help users that are stuck at a particular place in a given game, select the next level in a game with multiple levels based on the difficulty level of the level and user skills, offer incentives for increasing monetization opportunities, etc.. Our mechanisms and syntax for the aforementioned returned contextual information (also referred to as triggered alerts) has previously been described in [2].

The remainder of the paper is structured as follows. In Section II, we briefly cover related work in the area of mobile application analytics. In Section III, we offer an overview of the JumpStart service architecture. In Section IV, we provide some details about the two example applications used in this paper during the discussion of the JumpStart machine learning techniques. In Section V, we discuss the JumpStart machine learning techniques and present some results for the two example applications. Finally, in Section VI, we conclude our work.

II. Related Work

The past several years have seen a dramatic rise in big data, application (app) analytics, and ever-increasing sophistication in advertising and monetization [4]. Both startups and well established technology companies are in the game, including (but not limited to): Facebook, Flurry, Google, Amazon, Adobe, and Riot. Smaller startups for analytics include Localytics, Swrve, Countly, MixPanel, and Apptimize, to name only a few. Without this increasingly necessary new breed of tools, developers would have very limited insight into how users interact with their apps, especially standalone apps that do not depend on a backend server component (in the cloud or private data center).

Fundamental features shared by many of today's mobile analytics offerings include: SDKs for various platforms (Android, iOS, etc.), in-app event collection and warehousing, creation and analysis of app "funnels" consisting of sequences of key in-app events, event charting dashboards for various metrics (including KPI, retention, number of active users), analysis of user base demographics and personas, and push notifications. Furthermore, the very desirable ability to manage campaigns, such as A-B variants, is also rapidly becoming fundamental.

Although existing mobile analytics solutions can compute a large number of metrics, developers still bear the burden of understanding how specific application events (e.g., purchases) are correlated with user interactions prior to these events. Our work lifts this burden by employing machine learning techniques that learn event patterns that are: (1) common across many users, and (2) correlated with specific user actions. Developers can then register discovered patterns with JumpStart and associate specific information (i.e., actionable alerts) that is sent back to the app when these patterns are matched in near real time [2].

The two main machine learning techniques utilized by JumpStart are: Bayesian Rule Lists [6] and Falling Rule Lists [7]. These are machine learning algorithm for returning a classifier that can be easily interpreted by humans. This can be compared to techniques that learn a complex rule, such as [5]. Interpretability is an important constraint as we want a user of JumpStart to be able understand and modify these rules. We build on these algorithms in novel ways. The most interesting is a new clustering technique that uses Falling Rule Lists to do dynamic hierarchical clustering.

III. JumpStart Overview

JumpStart aims to be the first cross-platform solution to offer near real-time detection of user behavior patterns associated with past and current user interactions with mobile applications. User behavior pattern matches can trigger actionable alerts aimed to achieve specific goals, such as increased user retention and monetization. Our implementation uses the Esper pattern matching engine [3] and is cloud based to allow scalability for multiple applications and users. This allows devices to respond to user events that match patterns stored on the server with latencies based primarily on the quality of the network connection.

Fig. 1 shows the high-level architecture of JumpStart. Input and output adaptors are responsible for connecting to data sources or sinks and retrieving or storing/sending events. The analytics component is responsible for on-line and batch processing of events and computation of various metrics (e.g., daily active and new users, user retention, session and users per hour, event funnels of various lengths). The machine learning component, the main focus of this paper, is responsible for discovering correlations between events that reflect specific user interactions with an application utilizing supervised and unsupervised learning. The profile manager component is responsible for managing user profiles in terms of both statistical properties (e.g., session duration and event distributions) and user-specific event patterns discovered by machine learning. Finally, the real-time rules processor is carrying out real-time complex event processing and matching of event patterns, referred to as triggers in the remainder of the document.

As a simple example of JumpStart event collection, consider a game that has multiple levels (the basketball game described in Section IV). By recording events that correspond to level completion and failure, JumpStart can gauge the difficult of a level. A level that is easy to complete will require few attempts before success while a more difficult level might require many attempts.

Fig. 2 shows the ratio of the number of failures to the number of successes. While there are a range of opinions on how the game level difficulty should progress, it is without controversy that we do not want to advance to a new level that is considerably easier than the last level. At a minimum, the JumpStart framework gives us the analytics to determine if the order of levels is appropriate. If it is discovered that certain levels are too easy or hard, they can be removed, redesigned, or moved to a more appropriate location in the game, as the case may be.

Things get more interesting when we use the real-time nature of the JumpStart rules processor. With JumpStart we can detect how well a user is doing on a particular level by

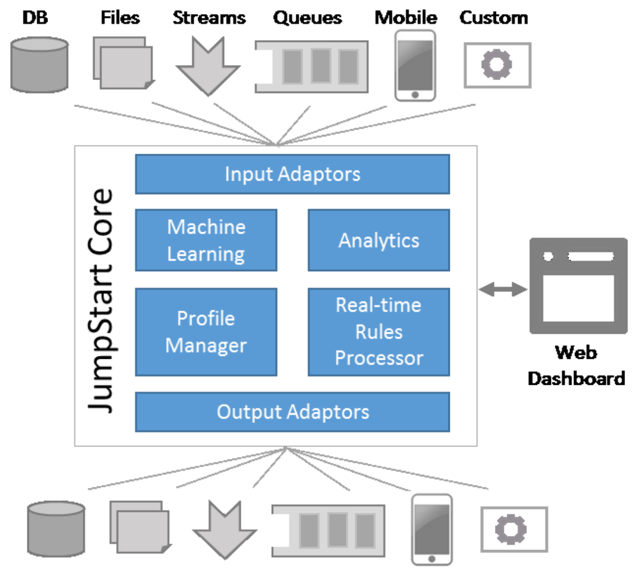

Figure 1. JumpStart high-level architecture

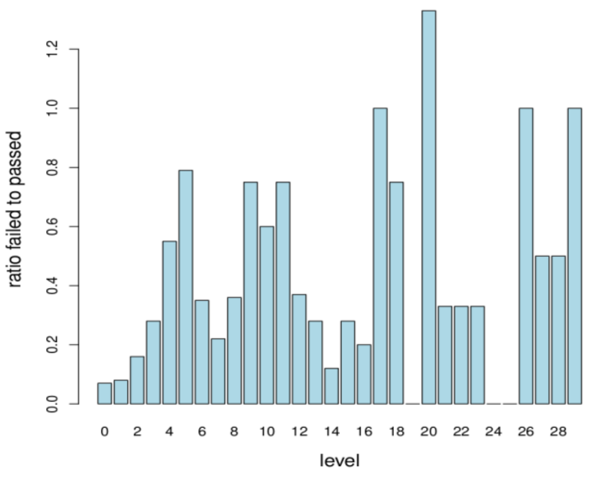

Figure 2. Basketball game level difficulty

comparing the level's outcome and time taken to those of the previous level via the appropriate event pattern. The output of the event pattern match can trigger a modification to the levels and even the creation of a non-linear path through the levels. This can reduce the churn of users who are bored by simple levels or frustrated by difficult levels. While some of this logic could be built into the application, using JumpStart is advantageous because:

1. Since it analyzes events from the entire user-base (not just the user in question), it can anticipate churn before it happens by, for example, clustering the user in with a "high churn" group.
2. It allows us to make modifications based on real user data as opposed to doing expensive customer trials that delay the release of the application and might be dated by the time the application is released.

IV. EVENT GENERATING APPLICATIONS

In this paper, we give experimental results for two applications. The first application is a game where either a single player or two players can compete in a basketball shooting contest. The game is composed of thirty levels of varying difficulty. For our experiments, we have 7,395 users, 122 unique types of events, and more than 630,000 collected events. Example events include taking a shot, missing a shot, completing a level, failing a level, and completing a section of the tutorial. In this paper, we call this game **basketball**.

The second application sends printed letters to members of the military. Many members of the military are not allowed access to the internet and need to receive physical letters. This application allows users to easily and simply compose these letters. For our experiments, we have 3,400 users, 55 unique events, and more than 137,000 collected events. Example events include buying tokens to send letters, starting a new letter, typing a letter using the keyboard, and taking a picture to send with the letter. In this paper, we call this application **post office**.

We have also performed experiments on three other applications: a music application, an event scheduling application, and a novelty humor application (with more that 11 million events). We obtained similar results but do not have the space to include these additional experiments.

V. MACHINE LEARNING WITH EVENTS

While analytics based on event data provides a useful tool for learning user behavior, machine learning techniques can often find non-obvious correlations that a human might miss. In this section, we give several techniques that find patterns that, while interesting on their own, can be used as a source of inspiration for creating event pattern expressions to be matched by JumpStart.

For the learning algorithms we are going to use, we need to represent a sequence of events as a fixed length $\{0, 1\}^n$ vector. We use the following mix of standard features.

- We take the last three events in the sequence and create a feature for every possible event type value (referred simply as event in the remainder of this paper). For example, if the last event is A, we create a binary feature A=1 to represent that the last feature is event A.
- We use a bag-of-words on the events in the sequence. We include single words and pairs. For example, if the events A and B occur consecutively in the event stream we create three binary features, A and B for the single events and A+B for the pair of consecutive events.
- We remove repeated consecutive events and represent them with special features that take into account the count. For example, if the event stream has 8 consecutive A events, we create features A, A>1, A>3, A>7. Notice that we create the features based on powers of 2.

In all cases, if a feature occurs more than once in an event sequence, we remove the repeats. In the following sections, we will occasionally add or remove features for specific learning problems.

A. Algorithms

We consider two algorithms for building understandable rules: the Bayesian Rule List (BRL) algorithm [6] and the Falling Rule List (FRL) algorithm [7]. Both algorithms return rules in the form of a decision list. A decision list is an ordered sequence of rules such that the first rule that is true makes the prediction. A rule is a conjunction of predicates that predicts a probability that the label is true. Fig. 3 gives an example of our notation.

```
rule 1 (90%)
        A+B
        C
rule 2 (5%)
        D=3
default (10%)
```

Figure 3. Decision list notation example

In this example, rule 1 has two predicates. First, the consecutive events A followed by B must occur somewhere in the sequence of events. Second the event C must occur somewhere in the sequence. If these two predicates are true, then the decision list gives 90% probability that the label is 1. If rule 1 fails, we check rule 2. In this example, rule 2 is true if event D is the third to last event in the sequence. If rule 2 is true, then the decision list predicts label 1 with 5% probability. The final rule is the default rule. If all the previous rules fail, we predict label 1 with 10% probability.

Both rule list algorithms are Bayesian algorithms that try to learn accurate - yet short - decision lists. The FRL algorithm has the extra constraint that a rule that is earlier in the list should have a higher probability of predicting the label than a rule that occurs later in the list. While this extra constraint can return a less accurate decision list, it makes the rules easier to interpret because many people will only consider the first few rules. A major competitor to these algorithms is decision trees. Experiments have found that the accuracy of the rule list algorithms is comparable to decision trees [8]. Decision lists also tend to be easier to understand because they are just a one sided subset of decision trees.

All experiments were performed using a single core of a 2011 vintage 2.4 GHz processor. The computational time for both algorithms, on the problems we studied, was on the order of a minute. While BRL does not scale as well on larger problems there is active research in improving its computational performance [9].

B. *Supervised Learning*

The first technique we cover is based on supervised learning. In supervised learning we have a set of objects, and we want to predict labels for these objects. For example, an object might be all the events in a session of a game and we want to predict if a user will make a purchase during the session. In order to learn a prediction model, we take a set of training data that consists of already labeled objects and use them to induce a classifier.

Typically, this classifier is used to predict labels for new objects. Instead we are focused on creating rules that describe the classifier and can be used to understand the data and generate triggers (e.g., register event patterns with the JumpStart rules processor). These rules can often reveal correlations that while reasonable in hindsight are difficult for humans to create by inspecting the data.

We perform three types of experiments based on events from the previously described post office application. The label for these experiments is based on a purchase event enabling the sending of future letters. For each experiment, we have a baseline level of performance based on this label. For example, out of the set of users we are considering say 25% make a purchase. Therefore, a rule that predicts a purchase with 25% probability is making a safe baseline prediction. Rules that have higher probability point to an informative situation where a purchase is more likely than the baseline. Rules that have a lower probability point to situations where a user is less likely to make a purchase. Note that cases at the baseline probability suggest that the user is “on the fence” with regard to a purchase.

A typical use of these algorithms is to run both the BRL and FRL multiple times to find interesting decision lists. In our experiments, we ran each algorithm twice (reporting here on only a subset of results; for the non-reported experiments, we get qualitatively similar results).

The first experiments are based on the initial 30 events generated by a user. Our goal is to predict if the user will eventually make a purchase. The baseline probability that a user makes a purchase is 31%. Fig. 4 gives the results of using FRL to generate a decision list.

```
rule 1 (60%)
        user_logged_in+compose_new_letter_screen
        launch_camera
rule 2 (37%)
        compose_letter_kb
default (7%)
```

Figure 4. FRL applied to **post office** for first 30 events

In this case, based on the first rule, if right after a user logs in to the application the user starts to compose a letter and at some point in these initial 30 events the user takes a picture then there is a 60% chance that the user will eventually make a purchase. If the first rule fails, then rule 2 is about equal to the baseline. However, if rule 2 also fails, then there is a good chance the user will never make a purchase. This suggests it is important to get a user actively creating a letter as soon as they start using the application.

The second experiments predict whether a user will make a purchase during a session based on the initial 15 events of the session. The baseline probability that a user makes a purchase is 25%. Fig. 5 gives the results of using BRL to generate a decision list.

```
rule 1 (2%)
        1=menu_opened
rule 2 (68%)
        letters_billing_appear
        user_info-user_logged_in
rule 3 (47%)
        launch_camera
rule 4 (11%)
        kin_pressed_from_menu+menu_closed
rule 5 (27%)
        compose_letter_kb
default (3%)
```

Figure 5. BRL applied to **post office** for first 15 events of session

Here we see that if a user opens a menu on the last event in our window of 15 events, he is unlikely to make a purchase. This rule might be a rare occurrence, but it suggests something that might need further study. (We also see this in some non-reported rule lists.) Next, we see that if a user does billing along with logging in right after the start of the session, then they are likely to make a purchase. If we reach rule 3, using the camera is likely to lead to a purchase as the user is probably creating pictures to send with the letters. Finally, if we get through all the rules then the user is extremely unlikely to make a purchase.

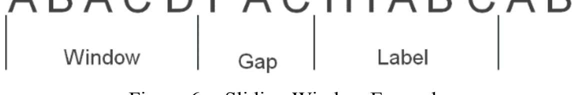

Figure 6. Sliding Window Example

The third experiment is based on a sliding window over the event sequence. The idea is that we want to make a timely prediction about whether a user is about to make a purchase. We use a window size of 15 events. We want advanced notice that the user is going to make a purchase so we don't look for purchase events right after the end of the window. Instead, we have a gap of 5 events. To give us more flexibility, we look for a purchase that occurs in the 5 events that follow the gap. Fig. 6 gives a graphical explanation based on window of size 5, a gap of size 3 and label of size 5. Because this instance generation technique creates a large number of instances that don't have a purchase in their label window, we de-skew the labels by randomly selecting negative instances to ensure that 10% of the instances have a purchase label. Fig. 7 gives the results of the FRL algorithm, but only reports the first five rules.

rule 1 (36%)
    new_recipient_next_pressed
    letters_new_next_pressed
rule 2 (28%)
    compose_new_letter_screen+compose_letter_kb
    launch_camera
rule 3 (26%)
    confirmation_for_purchase_letters
    compose_new_letter_screen+compose_letter_kb
rule 4 (20%)
    launch_camera+compose_letter_kb
    compose_new_letter_screen
rule 5 (19%)
    new_recepient+compose_letter_kb
    compose_letter_kb>1

Figure 7. FRL applied to **post office** with sliding window

All of these rules show that a user in the process of creating a letter has a higher probability of making a purchase. This is not surprising. However, it is interesting how the ordering of the rules can be interpreted as giving insight on the relative importance of the various actions.

What we are lacking with previous FRL example is the kind of actions lead to a user not making a purchase. This is a result of how FRL builds a decision list with the highest probabilities first. To study events that do not lead to a purchase, we flip the labels on the problem. This creates a decision list where the top rules are more likely to predict that the user does not make a purchase. Fig. 8 gives the first five rules generated by the FRL algorithm.

rule 1 (0%)
    menu_opened+units_pressed_from_menu
    units_pressed_from_menu
rule 2 (0%)
    2=menu_closed
    menu_opened
rule 3 (0%)
    1=menu_closed
    menu_closed
rule 4 (0%)
    1=menu_opened
rule 5 (0%)
    edit_profile_pressed_from_menu
    menu_closed+menu_opened

Figure 8. FRL applied to **post office** with sliding window and flipped labels

Here we observe interesting evidence that a user who uses the menu is unlikely to make a purchase in the near future. This matches our previous result dealing with sessions and suggests that any enticement for a user to make a purchase should wait until they are done using the menu.

C. *Clustering*

For many applications it is useful to organize similar users or sessions. We can automate this task by applying an unsupervised clustering algorithm. One problem with clustering is that it is difficult to understand the results. In this section we give results on using the FRL algorithm to better understand clusters and therefore help generate possible event patterns that can be used for generating actionable alerts.

The simplest form of our algorithm is easiest to explain with an example. Assume we create three clusters. We use these clusters to create three supervised learning problems. For each problem the instances in the cluster get a label of 1 while the instances in other clusters get a label of 0. Now we can apply FRL to learn a decision list that concisely describes each cluster. We use the FRL algorithm because we want our decision list to give preference to the high probability rules that describe what examples are in the cluster.

A further refinement of the algorithm is to add the ability to do dynamic hierarchical clustering. By this we mean that after the initial clustering one can use the insights provided by the rules returned to select clusters to expand. For example, assume we want to expand cluster B. We just rerun our simple algorithm to create a new clustering and new set of rules just using examples from cluster B. We can repeat this process to create a clustering tree with rules to explain the various clusters.

We believe our novel technique has many advantages over a traditional hierarchical clustering.

- The decision lists give an understanding of the various clusters. This includes how one cluster relates to another based on their ancestry.
- Instead of having to interpret a dendogram and picking an appropriate level of granularity, one can use our top down dynamic procedure to explore the data.
- The technique has a computational advantage in that we can decide to only explore the interesting part of the data to a depth that is informative.

Next, we give experiments for applying our clustering technique to the basketball data. Like many problems, the basketball data does not have an obvious label for supervised learning, so clustering is the logical approach. Note that clustering is also useful with problems amendable to supervised learning - such as the post office problem.

Our FRL based clustering code works with any clustering algorithm that returns a hard clustering. (A hard clustering is a clustering algorithm that assigns each example to one cluster.) We performed experiments with two clustering algorithms: k-means [10] and spectral clustering [11]. We found that, with our representation, k-means returned more meaningful clusters with the added bonus of being much faster; k-means only took a minute to build the clustering while spectral clustering took 30 minutes.

For our experiments, we use the same features as described in the start of Section V with the following modifications. First, we removed the features that represent the last three events in the sequence. These features are not needed as there is no special significance to the last events. Second, we added a feature called Short for any session that has less than four events. Based on an early analysis using our clustering technique, we found this useful to allow the clustering to identify the large number of short sessions in our data. Research shows that many users only try an application once before uninstalling potentially creating many short sessions.

Our experiments are based on clustering sessions. Using sessions can be useful for a range of applications. For example, one might want to understand users based on how they transition from various session types. For example, a user who continually returns to the tutorial might need extra assistance in using the app.

Fig. 9 show the results of generating two clusters for the basketball data. We have modified our decision list output to include information on the cluster and the number examples that are covered by the individual rules. As can be seen in the output, cluster 0 deals primarily with the game's tutorial while cluster 1 deals largely with the sessions that are very short.

```
Cluster 0 has 805 out of 2746 examples
    rule 1 (100%) 799 examples
            tutorial_sr_for_power_completed
            tutorial_drag_left_completed
    rule 2 (86%) 7 examples
            tutorial_sr_for_power
    default (0%) 1940 examples

Cluster 1 has 1941 out of 2746 examples
    rule 1 (100%) 1219
            Short
    rule 2 (95%) 194
            user_info+ pvp_pressed
    rule 3 (78%) 438
            user_info+story_mode_pressed
    rule 4 (47%) 104
            me_pressed
    default (18%) 791.0
```

Figure 9. Top level FRL clusters for **basketball**

Based on this analysis, we decided that cluster 0 is more interesting. We expand cluster 0 by creating two new clusters only using the examples from cluster 0. We show these decision lists in Fig. 10. Cluster 0-0 deals with the tutorial, but cluster 0-1 is interesting in that it is primarily deals with users who have started playing the game and completed some of the initial levels. These can be sessions where the user goes straight from the tutorial to playing the game.

VI. CONCLUSION

In this paper, we discussed the machine learning techniques used by the JumpStart real-time event analytics service in the context of two example mobile applications. JumpStart applies supervised and unsupervised machine learning techniques to application event streams for identifying users exhibiting similar application behavior and discovering event patterns that are strongly correlated with specific user activities within an application. In addition, we presented a novel approach that uses the Falling Rule List algorithm to concisely describe clusters generated by unsupervised machine learning algorithms. Cluster descriptions via decision lists enable hierarchical clustering by creating new clusters from existing ones.

Machine learning enables JumpStart customers to gain deep insights into how users interact with their applications and discover user behaviors that are correlated with specific application actions. These capabilities go far beyond the computed metrics and aggregate user actions offered by existing mobile analytics solutions.

```
Cluster 0-0 has 420 out of 805 examples
    rule 1 (87%) 79
            tutorial_sr_power+ tutorial_sr_power_completed
            missed_shot_0+tutorial_completed
    default (48%) 726

Cluster 0-1 has 385 out of 805 examples
    rule 1 (99%) 323
            missed_shot_1+shot_punk
            1_level_0_completed
    rule 2 (96%) 24
            1_level_1_completed
            shot_made
    rule 3 (39%) 103
            missed_shot_0+shot_punk
            tutorial_pull_back_to_aim_at_target
    default (0%) 355
```

Figure 10. Second level FRL clusters for **basketball**

REFERENCES

[1] InterDigital Inc., “JumpStart”, [Online]. Available from: http://www.interdigital.com/jumpstart, accessed: 2016.08.26.

[2] B. Falchuk, K.C. Lee, S. Loeb, E. Panagos, and Z. Yao, “Just-in-time reconnaissance and assistance for video game streams and players”, Proc. IEEE Consumer Communications and Networking Conference, Las Vegas, pp. 99-102, 2016.

[3] EsperTech Inc., “EsperTech event series intelligence”. [Online] Available from: http://www.espertech.com, accessed: 2016.08.26.

[4] T. Leaver and M.A. Wilson (eds.), “Social, casual and mobile games: the changing gaming landscape,” Bloomsbury Academic, New York, 2016.

[5] M.A. Alsheikh, D. Niyato, S. Lin, H. Tan, and Z. Han, “Mobile big data analytics using deep learning and apache spark, IEEE Network, 30(3), pp. 22-29, June 2016.

[6] B. Letham, C .Rudin, T.H. McCormick, and D. Madigan, “Interpretable classifiers using rules and Bayesian analysis: building a better stroke prediction model”, Annals of Applied Statistics, 9(3), pp.1350-1371, 2015.

[7] F. Wang and C. Rudin, “Falling rule lists”, *JMLR* Workshop and Conference Proceedings 38, pp. 1013-1022, 2015.

[8] J.R. Quinlan. “Induction of decision trees.” Machine Learning Journal, 1(1) , pp. 81-106, March 1986.

[9] H. Yang, C. Rudin, and, M. Seltzer, “Scalable Bayesian rule lists”, arXiv:1602.08610, 2016.

[10] E.W. Forgy, “Cluster analysis of multivariate data: efficiency versus interpretability of classifications”, Biometrics, 21, pp. 768-769, 1965.

[11] A.Y. Ng, M.I. Jordan, and Y. Weiss, “On Spectral Clustering: analysis and an algorithm”, Advances in Neural Information Processing Systems, pp. 849-856, 2001.