A Reusable Methodology for Player Clustering Using Wasserstein Autoencoders

Jonathan Tan and Mike Katchabaw(✉)

Department of Computer Science, Western University, London, ON, Canada
{jtan97,mkatchab}@uwo.ca

Abstract. Identifying groups of player behavior is a crucial step in understanding the player base of a game. In this work, we use a recurrent autoencoder to create representations of players from sequential game data. We then apply two clustering algorithms–k-means and archetypal analysis–to identify groups, or clusters, of player behavior. The main contribution to this work is to determine the efficacy of the Wasserstein loss in the autoencoder, evaluate the loss's effect on clustering, and provide a methodology that game analysts can apply to their games. We perform a quantitative and qualitative analysis of combinations of models and clustering algorithms and determine that using the Wasserstein loss results in better clustering.

Keywords: Data modeling · Autoencoders · Cluster analysis · Quantitative user studies · Interpretability

1 Introduction

Video games are differentiated by their genres, artistic designs, stories, and gameplay mechanics. In addition, they have varying budgets, team sizes, and production qualities. However, the backbone of all video games is the players, so understanding the player base is critical for a game studio. This has given rise to the discipline of game analytics, the study of play and players to provide insights into making better games and making games in better ways.

Our current work involves clustering players in the popular mobile game *My Singing Monsters*.[1] Creating clusters, or groups, of players based on their in-game behavior can help game analysts identify trends in the player base and better understand the parts of a video game that make it enjoyable or displeasing. Game producers can then use the clustering results to suggest improvements to the game or continue releasing content that players will enjoy. Understanding the player base through clustering is a win-win: the players' experiences improve, and the game studio enjoys increased success.

In Sect. 2, we go over some necessary background for our work. Next, we describe related work in Sect. 3. Then, we walk through the method for our

[1] http://www.mysingingmonsters.com.

B. Göbl et al. (Eds.): ICEC 2022, LNCS 13477, pp. 296–308, 2022.
https://doi.org/10.1007/978-3-031-20212-4_24

analysis in Sect. 4 and present the results of the models in Sect. 5. Finally, we discuss the results as a whole in Sect. 6. The source code of our work is available at https://github.com/tanjo3/wae-clustering.

2 Background

2.1 Recurrent Neural Networks

Neural networks are a practical and versatile form of function approximation. However, in their simplest form, they require their inputs to be feature vectors. Unfortunately, many real-life data, such as images or audio sequences, do not fit this description, so either we must adapt the input to the model or the model to the input. Constructing features for this type of data can be laborious, and it is not always clear how to do so. Recurrent neural networks (RNNs) are designed to work with sequential data as their inputs to address this issue. Effectively, a recurrent neural network recursively applies its hidden state over each item in the input sequence.

2.2 Autoencoders

The goal of an *autoencoder* is to recreate its input. Thus, an autoencoder consists of two parts: the encoder, which creates a latent representation, or embedding, of the input, and the decoder, which reconstructs the original input from the embedding. If we implement the autoencoder as a neural network, we can train it end-to-end via backpropagation to learn the weights. We use the standard mean squared error for the loss function, though any reconstruction error is possible. Here, we use an autoencoder as a feature extraction method in our work, creating vector embeddings from sequential player data and using them as input to clustering algorithms.

2.3 Adversarial Training and Generative Adversarial Networks

For autoencoders, the training procedure looks to minimize the reconstruction error. However, there is benefit in adding a secondary, competing criterion that instead seeks to be maximized. *Generative adversarial networks* (GANs) [10] are generative models that use such adversarial training. The discriminator tries to maximize its success rate by separating actual and generated samples, while the generator aims to generate realistic output to minimize it.

2.4 Wasserstein Distance

While GANs are handy and flexible models, they are infamously susceptible to unstable training [2]. Arjovsky, Chintala, and Bottou [3] introduce the *Wasserstein GAN* to fix this instability. They show that optimizing for the Wasserstein-1, or Earth-Mover (EM), distance results in more stable training than optimizing for the Jensen-Shannon (JS) divergence as the original GAN formulation does.

2.5 Clustering Metrics

Clustering is an unsupervised learning task that looks to group similar examples meaningfully. However, without knowing exactly how many clusters there are and to which each player should be assigned, there is no strict way to evaluate the quality of a clustering. However, we can use metrics that evaluate what one would expect from a "good" clustering, such as dense clusters and clear separation between clusters. These are, of course, not necessary nor sufficient criteria for good clustering, but they serve as a qualitative measure without ground truth labels. We use two different metrics: the Calinski-Harabasz index [5] and the Davies-Bouldin index [7]. The Calinski-Harabasz index prefers high values, while the Davies-Bouldin index prefers low values.

2.6 Archetypal Analysis

In this work, we compare the performance of two different clustering algorithms. Both algorithms produce cluster representatives, which we will use to interpret the properties and behaviors of the players in those clusters. The first is the traditional k-means algorithm [12], which produces k centroids representing their cluster. The centroids represent an average over its cluster members. However, the algorithm has a couple of key weaknesses. The first is that the centroids identified by the algorithm do not correspond to an actual member of the data set, which can result in illegitimate values for features, such as fractional values for features like player level. The second weakness is that centroids of similar clusters may be difficult to distinguish. For example, if the centers of mass of two clusters are close, the average values represented by the corresponding centroids could also be comparable.

We also apply archetypal analysis (AA) [6] to our data set for these reasons. AA instead finds k archetypes representing extreme members of each cluster. Since archetypes are members of the original data set, their features will always be valid and directly interpretable. Additionally, they are generally easily distinguishable among other archetypes. For example, consider a data set of all sports athletes. The k-means centroids for tennis and badminton players might be similar, and since they do not represent real players, they are possibly hard to interpret. On the other hand, the archetypes for the same clusters might be Roger Federer and Lin Dan. Comparing these two data points is easier since they are real players and exemplars of their respective sports.

3 Related Work

Understanding player behavior in games is an active area of research. For example, Drachen *et al.* [8] use self-organizing maps (SOMs) to identify four player classes in *Tomb Raider: Underworld*. However, the input to their model uses six hand-crafted features extracted from game data. Our work uses recurrent neural networks to avoid hand-crafting such features. In another study, Drachen *et al.*

[9] compare and contrast k-means and simplex volume maximization (SiVM) for clustering players in *Tera* and *Battlefield 2: Bad Company 2*. SiVM is an algorithm based on archetypal analysis. The authors find that while both methods produce a similar number of clusters, behaviors were easier to distinguish when using SiVM. In our work, we compare k-means to archetypal analysis on the players of *My Singing Monsters*, while also comparing different autoencoder models.

To our knowledge, the Wasserstein loss was first adapted to autoencoders by Tolstikhin *et al.* [15]. They demonstrate the training procedure and provide two different implementations differing in the penalty used in the autoencoder. One of the implementations uses the GAN penalty, which we adapt for our work. Furthermore, we modify the training procedure to use recurrent neural networks and the gradient penalty presented by Gulrajani *et al.* [11]. Our contribution is to evaluate the Wasserstein autoencoder's performance in clustering, providing both a quantitative and qualitative analysis of player data.

Boubekki *et al.* [4] introduce the Clustering Module (CM), which adds a single-layer autoencoder to the existing autoencoder structure. This autoencoder minimizes the reconstruction error on the embeddings and additional loss terms that derive from viewing the k-means clustering algorithm as a Gaussian mixture model. Their autoencoder system demonstrated performance on par with other state-of-the-art deep clustering methods as measured by several clustering metrics that use the true labels of the dataset. Unfortunately, we do not know the true labels for our work, so we cannot use these same metrics. Additionally, optimizing for k-means clustering may not be entirely appropriate for our data. Regardless, we demonstrate the performance of CM-augmented autoencoders with and without the Wasserstein loss in our work.

4 Method

This section describes the process we take to prepare and perform our clustering analysis. Below, we provide an enumerated list of the steps in performing our clustering analysis.

1. **Game Identification and Understanding.** The first step is to identify the game to be analyzed clearly. In our case, we first introduce the core gameplay mechanics for *My Singing Monsters* to better understand what exactly might be the gameplay features we use for analysis. After identifying which gameplay we will accumulate, we need to decide the period during which and the frequency with which we will collect the data. What is appropriate depends on the game that is being analyzed. In our case, we opt to aggregate features by day over 60 days, but being more or less granular may be appropriate for different games.
2. **Data Collection and Preprocessing.** The next step is to collect the data and preprocess it. Ensuring that the data passed to the model makes sense is vital. We want to look at interesting gameplay behavior, and as a result, we only choose to look at players with activity past a certain threshold. Other

analyses may make use of different criteria. It is also important that we do not impose too many biases on the data selection and preprocessing at this stage. We want the sampling of players to be random; otherwise, our inferences will also be biased.

3. **Model Definition and Construction.** The next step is to define and construct the models. Models such as neural networks will have hyperparameters, and it is important to do a hyperparameter search at this stage to compare models as closely as possible. As a result, the optimization criterion needs to be clearly defined. For our work, this would be the reconstruction errors of our autoencoders. A key contribution of this work is the exploration of autoencoder models augmented with the Wasserstein loss. This methodology can also be applied to other models and approaches to data modeling, as we will demonstrate with the Clustering Module [4].
4. **Analysis of Results.** We then compare the clustering performance of the models. Again, the comparison needs to be quantitatively defined. We use two clustering metrics to determine how many clusters we will consider and compare one model against another quantitatively.
5. **Translation Into Actionable Interpretations.** The last step is to visualize both the embeddings and the clusterings. The former helps us understand the distribution of the player base. The latter allows us to infer the representative behaviors of the players in our game. In a game studio, not everyone is familiar with data science. Game analysts can more easily communicate the results to non-technical parties through visualization. Translating the data into actionable interpretations is arguably the most critical step, and so we split this discussion in our work here into its own section below.

4.1 Game Identification and Understanding: My Singing Monsters

My Singing Monsters is a mobile game developed by Big Blue Bubble that is free to download and start playing. It was first released on iOS in 2012 and has since been released on numerous other platforms. It is a world-building game where players buy and breed singing monsters to place on their island. Each monster has a unique timbre and will sing or play a part of a song depending on which islands the player places them. One implicit goal of the game is to collect all the different types of monsters available on an island to get the complete island song. The usual method of obtaining new monsters is to breed two monsters that the player already owns to receive a monster egg. The player must then incubate the egg until it hatches. Both the breeding and incubation processes take real-world time. The player can speed up this process through diamonds or watching advertisements. The primary way to acquire diamonds is to purchase them from the store using real-world currency, but the player can also earn them passively in-game, albeit very slowly.

Additionally, the player can buy decorations using coins to beautify their islands. The singing monsters generate coins at a fixed rate depending on their breed and level. The player can level up a monster by feeding them treats. The player bakes treats at the cost of coins, and baking takes real-world time. Again, the player can speed up this process by using the premium currency of diamonds or watching in-game advertisements.

4.2 Data Collection and Preprocessing

Ultimately, each action that the player makes in the game is anonymized, timestamped, and recorded into a database. Thus, gigabytes of data are collected daily with a large player base. So how should we represent the player? We could look at every player's action since account creation, but that could be thousands of data points to consider. Traditionally, analysts will instead aggregate the data. However, selecting features that represent players well takes time and domain expertise. There is also the risk of human biases leaking through, intentionally or not.

For this reason, we try to adhere to the natural representation of the data as a sequence of events. We aggregate events by day since looking at individual actions might be too much. In particular, for each player, we consider 16 different gameplay features aggregated by day. These features are (1) the number of sessions (i.e., the number of times they log in that day); (2) the number of seconds played; their (3) minimum and (4) maximum player level; the number of (5/6) coins, (7/8) diamonds, and (9/10) treats earned/spent; the number of (11) monsters bred, (12) bought, and (13) sold; and the daily number of (14) ads watched, (15) in-game purchases made, and (16) offers completed. We collect these daily features for each player over the 60 days since account creation. We perform a preliminary filter to include only US players who (a) created their accounts between January 1, 2018, and June 1, 2018; (b) who have at least one game session after 60 days; and (c) who reached a maximum level of at least 4. We also remove players for whom we have incomplete data. From the remaining players, we choose 99,840 of them at random as our training set.

4.3 Model Definition and Construction

We prepare four recurrent autoencoder models. All models are constructed using PyTorch [14] and have hyperparameters as determined by an `optuna` [1] hyperparameter study over 200 trials. We execute the optimization study on a Linux GPU server consisting of 4 NVIDIA GeForce GTX 1060 6 GB GPUs. We utilize the study's default `optuna` parameters, and record results locally to a database file. Each trial runs for 16 training epochs, and we select the model with the lowest objective value for the clustering analysis.

In our work, we first construct a basic recurrent autoencoder, the RAE. For our second model, we follow Tolstikhin *et al.* [15] and construct a Wasserstein autoencoder (WAE) that uses the Wasserstein distance. Our third model augments the base RAE model with the Clustering Module [4]; we refer to this model as RAE-CM. Similarly, we augment the WAE with the Clustering Module for our fourth and final model to construct the WAE-CM. We omit details on the hyperparameters of these models here for brevity, but the interested reader can find the code and final hyperparameters at https://github.com/tanjo3/wae-clustering.

We then apply two different clusterings to their learned embeddings with our trained models: k-means and archetypal analysis. To determine the appropriate number of clusters, we first apply these algorithms to the entire data set while varying the number of clusters. We then evaluate the clusterings based on the metrics described in Subsect. 2.5. Finally, we proceed to analyze the composition of these clusters empirically. Note that the number of clusters acts as a hyperparameter for the RAE-CM and WAE-CM models. Regardless, we will select the number of clusters using the same clustering metrics as we use for the RAE and WAE models to unify the selection procedure for the number of hyperparameters.

***k*-means Clustering.** We focus our attention on clustering metrics for k-means. Since we do not know the "true" number of clusters in our data set, we run the k-means algorithm multiple times, varying the number of clusters from 2 to 8. We plot the results of this analysis in Fig. 1. A first observation is that the RAE-CM has quite aberrant values in the Davies-Bouldin index for two clusters. A second observation is that models that use the Wasserstein distance have better values than their non-Wasserstein counterparts, suggesting that the Wasserstein distance helps produce better clusters. Finally, a third observation is that models with the Clustering Module typically perform better than without it. Based on this analysis, we choose three clusters for the RAE, WAE, and RAE-CM models and six clusters for the WAE-CM model.

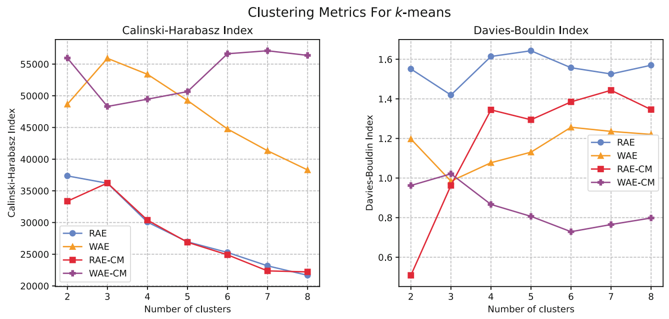

Fig. 1. Clustering metrics for different numbers of clusters using k-means. Recall that a higher value of the Calinski-Harabasz index is better, while we prefer a lower value for the Davies-Bouldin index. Our final determination for the number of clusters is three for the RAE, WAE, and RAE-CM and six for the WAE-CM.

Archetypal Analysis. We perform a similar analysis using archetypal analysis and plot the clustering metrics on these results in Fig. 2. Generally speaking, the RAE performs the worst of the four models. The WAE model performs the best on the Calinski-Harabasz index and improves over the RAE and RAE-CM for the Davies-Bouldin index. On the other hand, the WAE-CM model performs comparably to the RAE and RAE-CM for the Calinski-Harabasz index. The less-noticeable performance improvements of the models that use a Clustering Module may be due to the different clustering criteria that archetypal analysis uses over k-means. Based on these results, we choose three archetypes for the RAE, WAE, and WAE-CM models and five archetypes for the RAE-CM model.

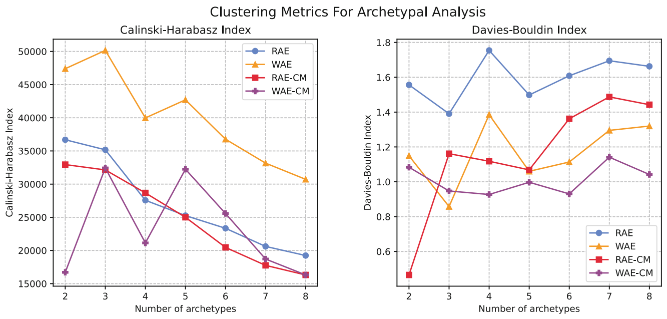

Fig. 2. Clustering metrics for different numbers of clusters using archetypal analysis. Our final determination for the number of clusters is three for the RAE, WAE, and WAE-CM and five for the RAE-CM.

5 Translation into Actionable Interpretations: Clustering Analysis

In the previous section, we found that utilizing the Wasserstein loss in our autoencoders' training can improve the clusterings' performance, as given by our metrics. Furthermore, combining this with the Clustering Module can add further performance. This section performs an empirical analysis of the clustering and interprets some of our formed clusters.

5.1 Player Visualizations

To start, we first project the learned embeddings into 2D space for visualization purposes. To accomplish this, we use UMAP [13]. We then overlay various player metrics to understand if there are correlations between player features and these player metrics.

Day-120 LTVs. A player's day-120 lifetime value (D120 LTV) is defined as the total revenue that the player has generated after 120 days since account creation. We are interested to see if there is any correlation with the gameplay over the first 60 days with long-term LTV. We map the log-transformed D120 LTV of each player with their UMAP embeddings for all four models in Fig. 3.

Immediately, we can see some correlation of D120 LTV with clustering. Players in the sparser parts of the clustering tend to have lower LTVs than those in the denser parts. Further analysis, omitted here for brevity, generally shows that the denser the cluster, the higher the D120 LTV. Additionally, these dense areas are where the smallest cluster of players exists, suggesting that high LTV players have more distinctive playstyles than low LTV players.

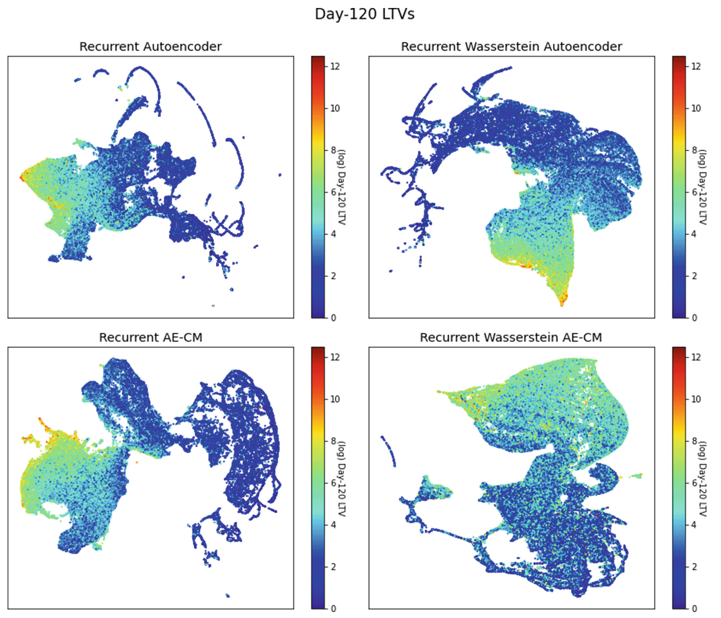

Fig. 3. A scatter plot showing the players and their log-transformed D120 LTVs. We show the color scales for the LTVs to the right of each plot. (Color figure online)

Acquisition Source. Next, we observe how organic vs. non-organic players are clustered. An organic player is one that was not introduced to the game via a user acquisition (UA) campaign. These campaigns are expensive to start and maintain. We plot the players color-coded by their status in Fig. 4. We can see that most of the players are organic and that the distribution of organic players is largely uniform. We can conclude that the game features do not indicate whether a player is organic, contrasting with D120 LTVs. It would seem to suggest that a player acquired through a UA campaign is not predisposed to spend more on the game or that the UA campaigns run in the past did not necessarily attract higher LTV players.

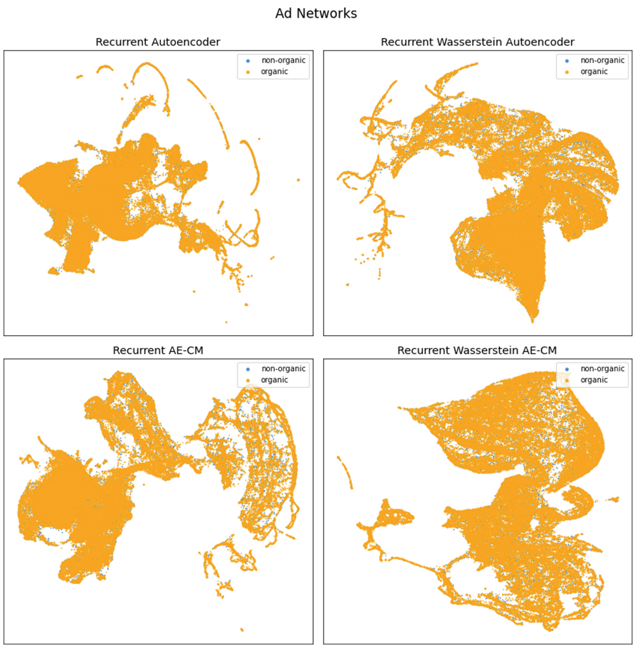

Fig. 4. A scatter plot showing organic and non-organic players.

5.2 Representative Analysis

Finally, we take a look at the daily features of the representatives of some of the clusters. For k-means, we look at the average of all players in the group, as represented by the cluster centroid. For archetypal analysis, we look at the player selected as the archetype of that cluster. We only consider some of the features for brevity and only look at one model each for both clustering algorithms. For k-means, we choose to look at the centroids of the WAE-CM model. For archetypal analysis, we will compare archetypes from the RAE-CM model.

***k*-Means: WAE-CM Model.** We show the plot of a selection of the daily features for the k-means WAE-CM clustering in Fig. 5. There are six centroids in the WAE-CM model, and we can immediately see that the smallest cluster, Cluster 6, is interesting. Because of the extreme early activity in the clusters, the scale of many plots is distorted, making it hard to discern the values for the other centroids. Interestingly, based on the number of sessions and seconds spent on islands, the centroids of all clusters seem to reach similar values after 60 days. Players in Cluster 4 seem to watch more ads in the first 30, but Cluster 6 takes over for the last 30 days. Players in Cluster 5 seem to breed more monsters in the first 30 days before Cluster 6 again takes over.

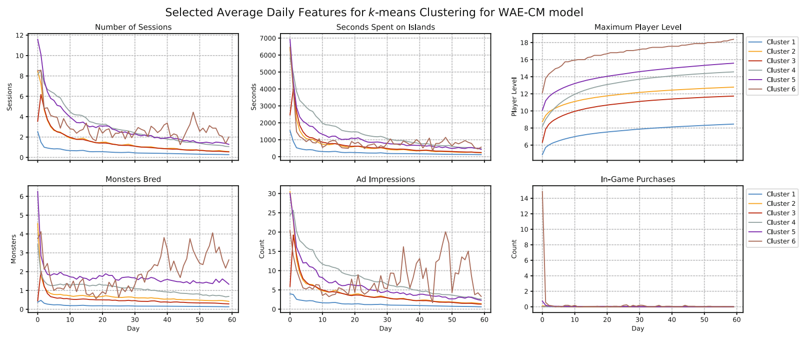

Fig. 5. Line plots showing the 6 of the 16 daily features of the centroids of the WAE-CM model. These values are averages of the values over all players in the cluster.

Archetypal Analysis: RAE-CM Model. Next, we look at a selection of the daily features of the archetypes identified by the RAE-CM model. We plot the daily features in Fig. 6. We can note that with the maximum level feature, the lines are stepwise since these are the features for an actual player. For this reason, the behavior exhibited in these plots is less subdued than with the k-means clustering. We see that the archetype for Cluster 2 gets to level 7 early on but is inactive until after 50 days. The archetype for Cluster 1 is more active but does not progress past level 10 over the 60 days. The remaining clusters have fairly consistent activity. The archetype for Cluster 5 makes many in-game purchases and achieves the highest level but does not log any more sessions than the other active clusters do. The archetype for Cluster 3 almost reaches the same level but with much fewer in-game purchases. They seem to play most of the five archetypes and might represent primarily active free-to-play players.

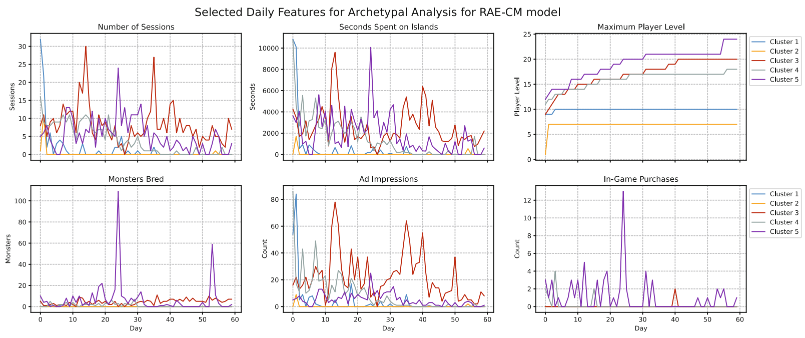

Fig. 6. Line plots showing the 6 of the 16 daily features of the archetypes of the RAE-CM model. Note that these are the features of an actual player, restricting the values to legal values. Note the stepwise appearance of the maximum level feature and the spikiness of the other features.

6 Discussion

This work has provided a reusable methodology for player clustering that game analysts can adapt and follow with their games, giving valuable insight into the game and its players. We demonstrate the methodology's application by an analysis of the players of *My Singing Monsters*. This methodology includes the application of creating player representation from sequential game data, the application of a clustering algorithm, and the evaluation of the results.

We use UMAP to visualize player embeddings, giving us a sense of distance between players. The visualization clearly shows that the clusterings are correlated to D120 LTV but not the player's organic statuses. Displaying this information as text or tables would not be as effective. Through contrasting k-means and archetypal analysis, we also demonstrate differences in conveying player behavior within a cluster. We note that by using archetypal analysis, particularities of a cluster's behavior are more apparent. By not having averaged values, the analysis is more interpretable.

Future work includes drilling down into these clusters to identify more specific trends, possibly even labeling the clusters with player profile names such as "optimizers" and "island decorators". In addition, Wasserstein models add a level of regularization. Exploring other regularization methods could yield better clustering results; we explore the Clustering Module's effect in this chapter. Another avenue for future work includes identifying or developing other metrics for evaluating clusterings. Metrics that do not inherently favor dense clusters could be of particular interest. Finally, we can apply the methodology outlined in this chapter to other games or refine the process further to suit the specific needs of a game studio.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
2. Arjovsky, M., Bottou, L.: Towards principle methods for training generative adversarial networks (2017). arXiv:1701.04862
3. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN (2017). arXiv:1701.07875
4. Boubekki, A., Kampffmeyer, M., Brefeld, U., Jenssen, R.: Joint optimization of an autoencoder for clustering and embedding. Machine Learning **110**(7), 1901–1937 (2021). https://doi.org/10.1007/s10994-021-06015-5
5. Calinski, T., Harabasz, J.: A dendrite method for cluster analysis. Commun. Stat.-Theory Methods **3**(1), 1–27 (1974). https://doi.org/10.1080/03610927408827101
6. Cutler, A., Breiman, L.: Archetypal Analysis. Technometrics **36**(4), 338–347 (Nov 1994). https://doi.org/10.1080/00401706.1994.10485840
7. Davies, D.L., Bouldin, D.W.: A Cluster Separation Measure. IEEE Trans. Pattern Anal. Mach. Intell. PAMI-1(2), 224–227 (1979). https://doi.org/10.1109/TPAMI.1979.4766909

8. Drachen, A., Canossa, A., Yannakakis, G.N.: Player modeling using self-organization in tomb raider: underworld. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 1–8. IEEE, Milano, Italy (2009). https://doi.org/10.1109/CIG.2009.5286500
9. Drachen, A., Sifa, R., Bauckhage, C., Thurau, C.: Guns, swords and data: clustering of player behavior in computer games in the wild. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 163–170. IEEE, Granada, Spain (2012). https://doi.org/10.1109/CIG.2012.6374152
10. Goodfellow, I., et al.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27. Curran Associates, Inc. (2014), https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
11. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.: Improved Training of Wasserstein GANs (2017). arXiv:1704.00028
12. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Infor. Theory **28**(2), 129–137 (1982). https://doi.org/10.1109/TIT.1982.1056489
13. McInnes, L., Healy, J., Melville, J.: UMAP: Uniform manifold approximation and projection for dimension reduction (2020). arXiv:1802.03426
14. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F.d., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
15. Tolstikhin, I., Bousquet, O., Gelly, S., Schoelkopf, B.: Wasserstein Auto-Encoders (2019). arXiv:1711.01558