STOCKHOLM SCHOOL OF ECONOMICS
Department of Economics
5350 Master's Thesis in Economics
Academic Year 2016–2017

Cluster Analysis from a Game Theoretical Framework

Rachael Ahn (41068)

Abstract: Although regression analysis is the predominant method of statistical analysis in econometrics, unsupervised methods such as cluster analysis can bring novel insight into detailed data sets. In this paper, we analogize game theory with cluster analysis, and find that the *k*-means algorithm is a Nash equilibrium in a repeated game framework. Along with the clustering process, we also present dimensionality reduction methods in a game theoretical framework.

We then present the results of cluster analysis on corporate data from the mobile game company *King*. Our results show that cluster analysis can extract valuable patterns from large data sets in both static and time-series data by segmenting the population into groups with overarching similarities, rather than forking on various combinations of variables.

Keywords: cluster analysis, algorithmic game theory, Nash equilibrium, repeated games

JEL: A12, C10, C61, C63, C72, C73

Supervisor: Karl Wärneryd
Date submitted: May 11, 2018
Date examined: May 30, 2018
Discussant: Tadas Gedminas
Examiner: Maria Perrotta Berlin

Acknowledgements

I would first like to thank my supervisor Karl Wärneryd and program director Anna Dreber at Handelshögskolan i Stockholm - Stockholm School of Economics for helping me navigate through this thesis and providing guidance throughout the Master of Economics Program.

I also want to thank the entire AI R&D team at *King* for their help. I am especially grateful to Lele Cao and my supervisor Stefan Freyr Gudmundsson for their advice and feedback.

Last but not least, I want to thank my parents and friends for supporting me through the master program, especially Ruiqi and Karolina for never failing to make me laugh.

Table of Contents

1 Introduction **5**

1.1 Why Study Cluster Analysis in Economics? 5

1.2 Literature Review 6

2 Data Science Background **8**

2.1 Cluster Analysis 8

2.2 Types of Models 9

2.3 Silhouette Score 10

2.4 Dimensionality Reduction 12

2.5 A/B Testing 18

3 Game Theory **19**

3.1 Types of Games 20

3.2 Nash Equilibrium 21

4 The Model **23**

4.1 K-Means Algorithm 23

4.2 The Game 24

4.3 Dimensionality Reduction 25

4.4 K-Means Strategy 26

5 Business Application - Method **29**

5.1 Data 30

5.2 A/B Test 31

6 Results **32**

6.1 Dimensionality Reduction 32

6.2 Results on A/B Test 37

7 Conclusion **45**

List of Figures

Section 2

2.1 Silhouette Analysis for $k = 3$ to $k = 5$ 11
2.2 PCA from Two-Dimension to One 13
2.3 Simple Autoencoder 15
2.4 Deep Autoencoder 16
2.5 t-SNE vs PCA 17
2.6 t-SNE with perplexity= 2 18

Section 4

4.1 K-Means Iterations 24

Section 6

6.1 Dimensionality Reduction for Static, Active Gamers, ... 34
6.2 Dimensionality Reduction for Time-Series, All Gamers 36
6.3 Dimensionality Reduction for Time-Series, Active Gamers 38
6.4 t-SNE of Latent Space, $k = 5$ 39
6.5 Bar Plot, Static Clusters 40
6.6 Heatmap of Static Clusters, $k = 5$ 41
6.7 99% Confidence Intervals, Static Clusters 41
6.8 Bar Plot, Time-Series Clusters 42
6.9 Heatmap of Time-Series Clusters, $k = 5$ 43
6.10 99% Confidence Intervals, Time-Series Clusters 44

List of Tables

Section 2

2.1 Types of Learning Methods 8

Section 3

3.1 Two Player Games 19

Section 8

8.1 Static Silhouette Scores, Active Gamers 49
8.2 Time-Series Silhouette Scores, Active Players 49
8.3 Static Silhouette Scores, All Players 50
8.4 Time-Series Data, All Players 50
8.5 Deep Autoencoder with Over-Complete Layer, Silhouette Scores 50

1 Introduction

1.1 Why Study Cluster Analysis in Economics?

The goal of this paper is to model cluster analysis and advanced machine learning techniques in a game theoretical framework. The modern field of data science itself is a combination of statistics and computer science, making use of technological computing advancements in solving more complex statistical models, including the methods used in econometrics. Many economists do not know that they are already data scientists: regression analysis in econometrics is one of the predominant *supervised methods* (See Section 2) in data science (Lupták, 2015). Although data science is closely associated to econometrics, it shares many similarities with game theory as well. They stem from a similar framework - both take inspiration from observations in nature and iterate to stable equilibria. Hugo Steinhaus, one of the founders of the K-Means algorithm that we focus on this paper, was also an early founder of game theory (Steinhaus, 1956). By exploring similarities in cluster analysis with econometrics and classic game theory, we attempt to bridge the gap between more classical economics and modern data science methods.

By utilizing the latest techniques in data science, game theorists as well as econometricians can take advantage of the latest innovation in machine learning. Data science can be used to solve complex economic models in game theory, where games with higher dimensions cannot be easily represented with 2D payoff matrices. It can also be used to segment the population in applied game theory experiments, where noisy results can mask underlying behavior patterns. Lastly, unsupervised methods allow researchers to broadly analyze a multivariate data set and make hypotheses without over-segmenting the data set by different variables and trying to pick out any statistically significant relationships between different variables. This prevents p-hacking or fishing, when researchers pick out statistical significant p-values in a data set using multiple comparisons rather than only testing the stated hypothesis.

1.2 Literature Review

Many top computer science journals such as IEEE publish a great deal of papers and even entire issues in an economic framework, particularly in algorithmic game theory. Dhamal et al. (2011) develop a new cluster algorithm using the shapley value and other concepts from game theory. Pelillo (2009) describes the connections between game theory and data science in an article, as well as work with Bulò in his Ph.D. thesis and subsequent paper on hypergraph clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* published an entire journal in game theory (Vasilakos et al., 2010). The 13 papers in this journal use classical game theory to study wireless network resource allocation and auctions, anomaly detection, and virus-drug interaction.

Despite regression analysis being a core statistical method in both economics and computer science, there were few such technical collaborations in the economics field. Only recently have data science methods been used to further insight into economics data, but still rarely in economics journals. For example, Best et al. (2017) uses a text classification machine learning method to analyze responses to policies, particularly individuals and organizations tasked with implementing government policies. Unsupervised methods, particularly cluster analysis, can be used to group similar entities by a range of factors. Řezanková (2014) uses agglomerative hierarchical clustering, k-medoids, and K-Means algorithms to group Eastern European economies by multiple levels of economic activity by age group and group various household goods as economic indicators. After this exploratory analysis, they then go back and see the defining factors for each cluster, and how similar or dissimilar different countries are by variable using hierarchical dendograms. This exploratory analysis could be the baseline for more classical regression analysis in subsequent econometric papers. Florczak et al. (2015) cluster micro household behavior in Poland to analyze if the variables of interest have significant multivariate correlations. Setyaningsih (2012) implement cluster analysis to study small businesses in Indonesia, using micro data such as employee education level and management styles. Hollenstein (2000) implement similar research, but use firm-level data measure innovation levels using a variety of indexes to compare between different types of Swiss companies. In all these papers, a non-linear analysis allows researchers to look for multivariate patterns without segmenting the testing population or having

to choose between multiple indicator variables.

Exploring a data set through unsupervised methods such as cluster analysis can help find novel patterns in the data. The practical application in this paper partially focuses on cluster analysis on time-series data, as it is a commonly used data type in economics. The cluster analysis process we focus on in this paper can be highly useful in econometrics, as it replaces manually segmentation of the population of interest. It can be used to find meaningful patterns in the data without picking out mere statistical anomalies (Benjamin et al., 2017; Camerer et al., 2016; Gelman and Loken, 2013). Cluster analysis hinders the researcher from manually segmenting the data set or choosing between input variables. In the result section of this paper, we show how cluster analysis can be used to segment A/B test results (see Section 6).

In Bulò and Pelillo (2009), a hypergraph cluster algorithm is analogized to an evolutionary stable strategy game. Inspired by Bulò and Pelillo's paper *A game-theoretic approach to hypergraph clustering*, we use the framework of game theory to model the K-Means algorithm and its related validation technique, but from a more economics-focused perspective.

Outline

In this paper, we start with a detailed technical background in cluster analysis (Section 2) and game theory (Section 3), for readers unfamiliar with either field. In Section 4, we detail the K-Means algorithm and model it in a game theoretical framework, similar to Bulò and Pelillo (2009). We describe the method in which we applied this model on a corporate data set from the mobile game company *King* in Section 5, and the results on static and (time-series) activity features in Section 6. Lastly, we conclude with final comments and ideas for further research in Section 7.

Although this is a thesis for the Master of Science in Economics, we incorporate data science research methodology such as Background and Method akin to data science papers.

2 Data Science Background

The following section chapter provides background information for those unfamiliar with data science approaches. In the field of economics, the most commonly used statistical method is regression analysis. Although it can be very effective in fitting a set of inputs $x = \{x_1, x_2, ...x_n\}$ to output y by a set of weights $\{\beta_1, \beta_2, ...\beta_n\}$, it is limited by the availability of such an output label y. Supervised learning methods are defined in the use of a y value, called the ground truth, that is used to fit the model and measure its accuracy. On the other hand, unsupervised learning has no ground truth label, only the input data set x. Unsupervised learning methods focus on extracting patterns of similarity and difference from just the input data set (Friedman et al., 2009). Table 2.1 shows the core supervised and unsupervised methods.

Table 2.1: Types of Learning Methods

Supervised	**Unsupervised**
Maps x to ground truth label y	Maps x to some grouping based on x
Regression Classification	Clustering Anomaly detection Association

2.1 Cluster Analysis

Cluster analysis is one of the core unsupervised learning method in the field of data science and machine learning. Instead of fitting the data set into a set of weights that predict the output label, the data set is partitioned into groups with similar characteristics (Aggarwal and Reddy, 2013). It is commonly used in fields such as business and marketing analytics, where there is ample data available, such as characteristics of the customer base, but generally few heuristics on which information is actually valuable for the business.

Because the results cannot be measured for accuracy with a known output label, the results must instead be validated using various measures of inter-cluster

similarity or intra-cluster separation. The features used for cluster analysis must be carefully chosen, and may need dimensionality reduction such as principal component analysis or autoencoder (more in Section 2.4). A complicated optimization model such as this requires a more complex algorithm using machine learning, as well as robust validation to justify the results of the model.

2.2 Types of Models

There are various methods in which to cluster the data set. Probabilistic models such as Expected Maximization (EM) algorithms are a type of generative models that assume the data set follows a certain distribution, and measures the likelihood of each point following this distribution. The sum of probabilities is maximized to find the likely distribution of the data set. Probabilistic models can sometimes be simplified to distance-based algorithms, which divide data points to the closest cluster. Distance-based algorithms are either flat or hierarchical.

Agglomerative hierarchical methods merge points or sets together, while divisive hierarchical methods split sets into smaller ones. In contrast, flat methods cluster the data set in one shot. The most common flat methods are K-Means (distances are measured to the mean of each cluster, called the centroid), k-medians (distance is measured to the median value of each cluster, therefore less susceptible to outliers), and k-medioid (clusters are chosen by selecting specific data points as centers).

Different methods have varying strengths and weaknesses depending on the type of data set or underlying pattern. The most common and simple clustering method is the K-Means model, in which the cluster centroids are represented by the mean of the data points assigned to each cluster. Although we experiment with alternate methods such as k-prototypes (K-Means optimized for categorical data) and hierarchical models, we found that loss rate and results were similar to K-Means. We will be using K-Means model for the rest of the paper, so that methods can be more consistently compared using validation such as Silhouette Score. K-Means is explained in greater detail in Section 4.1.

2.3 Silhouette Score

In practical applications, the "correct" value of k, the number of clusters to group the data set, is unknown. The silhouette score can be used to measure both the homogeneity within each cluster, as well as the heterogeneity between clusters:

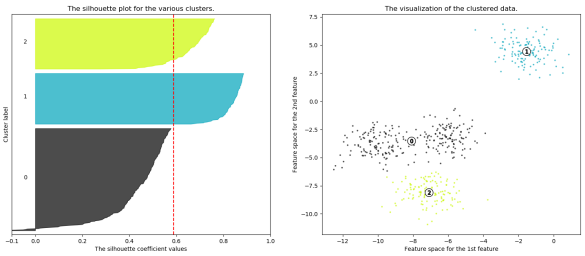
$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}}$$

where $a(x_i)$ is the average distance from point x_i to other points in the same cluster, and $b(x_i)$ is the average distance from x_i to points in the closest neighboring cluster (Rousseeuw, 1987). The average silhouette score for each clustering result can be used to compare both the intra- (within) and inter- (between) cluster variance between different k values. Figure 2.1 shows the silhouette analysis for $k = 3$ to $k = 5$. The area graphs show each point's silhouette score, with colors representing different clusters. When $k = 3$, the largest cluster (black) is too widely distributed, and $a(x_i)$ is too large, implying it should be split up to multiple clusters. When $k = 5$, the two smallest clusters (yellow and light blue) are close together, and $b(x_i)$ is too small, implying they should be joined to one cluster. It shows that $k = 4$ has the highest average silhouette score (red dotted line), because points in the same cluster are close together, while points in other clusters are far apart.

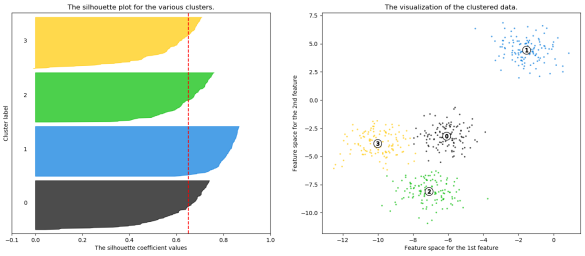
Silhouette score has the advantage of taking into account the size and distance between different clusters, whereas other commonly used methods such as distortion or elbow method only looks at the variances of the data set within its assigned cluster (Aggarwal and Reddy, 2013). Because it takes into account the variance within as well as between clusters, it is more consistent for different values of k. We use the silhouette score for the main quantitative comparison method between clustering algorithms.

However, we also look at alternate values of k when the results are not useful - for example, if one cluster contains a disproportionate amount of the data. In practical uses where k is not pre-determined by some external requirement, there is also a constraint of choosing a "reasonable" number of clusters k.

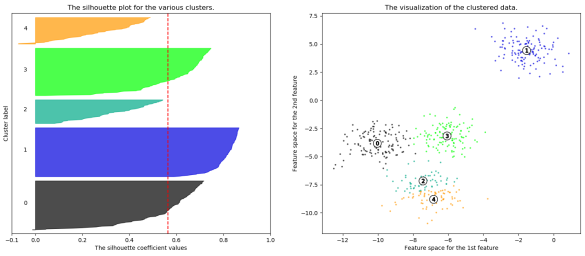
$$k < \bar{k}$$



(a) $k = 3$



(b) $k = 4$



(c) $k = 5$

Figure 2.1: Silhouette Analysis for $k = 3$ to $k = 5$ (Scikit-Learn)
$k = 3$ shows one cluster that is too large, $k = 5$ has two clusters that could be compiled together. $k = 4$ is "just right" - clusters are similar sizes, well segregated from each other, and contain points close to each other. [This figure is best viewed in color]

For example, a market analyst may set $k = 10$ if that is the number of categories the business has decided to segment the market by. If there is not a pre-determined rule, she may set $\bar{k} = 15$, because any further market segmentation will not be practical for the business. If she plans to manually cluster some

similar clusters together after the K-Means algorithm, $\bar{k}$ may be even higher.

2.4 Dimensionality Reduction

With the emergence of big data and micro-data collection, ample information is available for cluster analysis. A novel problem that emerges, however, is that there is *too much* information, and it is often difficult to determine which variables are important or correlated. Therefore, dimensionality reduction methods are used to condense the data set and reduce noise, leaving only essential information.

For example, imagine a clothing company has a data set of various body measurements from their customers, from height and waist circumference to arm length and weight. A dimensionality reduction method may reduce all these features down to a couple variable, maybe representing dress sizes the company could have. This reduces the large variances in people's body shapes to a singular number that still contains the general information of someone's body dimensions. This kind of dimensionality reduction prevents over-personalization or over-fitting.

Over-fitting is particularly a problem in unsupervised learning methods, when the accuracy of the model cannot be measured with a known output. This is when a model fits the sample data set, but is too specific to use for the entire population. In this example, personalizing a specific size for each individual is impractical. Reducing all of their measurements to one or two variable allows a company to produce a limited number of different sizes and still satisfy most of their customers.

There are various ways to reduce the number of features in a data set, from linear combinations to deep learning algorithm.

Principal Component Analysis

Principal Component Analysis (PCA) is one of the simplest and widely used dimensionality reduction technique (Pearson, 1901). The goal of PCA is to create a linear combination of the data set features (Shlens, 2014). It searches for the principal component "axis" along which various features are correlated,

and transform the data along this axis to reduce variance and the number of features. When two variables have a strong linear correlation, one variable can be used to describe both.

Figure 2.2 shows the same two-dimensional data set in its original form (left) and PCA-transformed form (right). In the original form, the data roughly fits the line $y = \frac{3}{5}a + 2$. By "turning" the data along this axis, it reduces the y dimension to $pc2 = 0$, and only one dimension $pc1$ is needed to describe the data set. The number lines below the graphs show the variance along the x and y axis is reduced to only variance along $pc1$, while $pc2$ is centered close to 0.

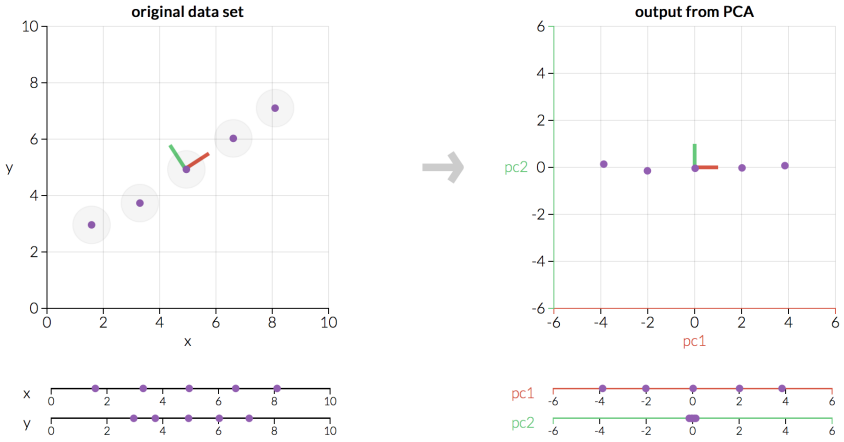


Figure 2.2: PCA from Two-Dimension to One (Powell, 2014)
The original data set is converted from two dimensions (x, y) to one dimension (pca1) using PCA by reducing the variance along the diagonal trend. pca2 is reduced to 0 for all points and can be taken out.

While beneficial for simple and fast dimensionality reduction, PCA has the drawback of assuming that the true variables of interest are the ones with the greatest variance. However, this may not always be the case, so it is critical to normalize the data. It also assumes linear relationships between different features, and that the principal components are orthogonal. For more complex relationships, an autoencoder may be more suitable.

Autoencoders

Autoencoder (AE) are a form of neural network used to compress and decompress large data sets using hidden layers (Stanford). While a supervised learning method fits a set of input values to a set of output values, autoencoders try to fit a set of values to itself, using a learned form of the identity function. However, it is limited by the number of hidden units in the inner layers, which is less than the original number of features. The identity function aims to find a set of weights w that keep as much of the original data set's information as possible, but with less values. The first half is called the encoder. The encoder is like a non-linear form of the PCA, in that the data set can be reduced to less variables or features. The second half is called a decoder, and uses the inverse of the encoder to reconstruct the data set. A predetermined loss functions measures the difference between the original data set and the reconstructed data set, and the autoencoder optimizes on this function. The resulting data set is compared to the original, and weights are iteratively adjusted to reduced the error between the reproduced and original data sets.

It is often used to find structures within the data, such as patterns in images or correlations among variable sets. The smallest layer with m^* features is called the latent space, and these encoded values are used for dimensionality reduction purposes. A simple autoencoder is visualized in Figure 2.3. The first set of blue circles Y represent the input - the original data set - before being encoded, with 8 features in this case. The green circles V are the encoded representation, and have 6 features. The last column of blue circles $\hat{Y}$ shows the reconstructed data set output.

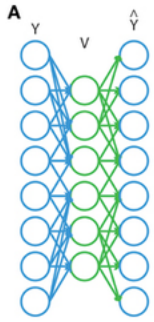


Figure 2.3: Simple Autoencoder (Berniker and Kording, 2015)
The first set of blue circles Y represent the original data set before being encoded, with 8 features in this case. The green circles are the encoded representation, and have 6 features. The last column of blue circles $\hat{Y}$ shows the reconstructed data set. This autoencoder reduces the data set from 8 features to 6, then back to 8. It aims to keep the reconstructed data as similar to the original data set as possible.

Autoencoders can also have more complex forms that can capture more complex and deep information. Different loss functions and metrics can measure the difference between the original data set y and the recreated data set $\hat{y}$. A common metric is the mean square error (MSE)(Keras Documentation, a):

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

There are also various optimizer functions to measure (Keras Documentation, b). We chose the Adam optimizer due to its efficiency in large and sparse data sets (Brownlee, 2017; Diederik P. Kingma, 2015).

Deep Autoencoder

Instead of having just one encoded layer, Deep Autoencoder (DAE) can have multiple layers to reduce dimensions gradually. Figure 2.4 shows a DAE with 5 inner layers. Instead of reducing 8 features to 2 in one shot, each layer reduces the most important information down to the essentials, to prevent important information from being lost, and uses mirroring layers to decode back to the recreated data set.

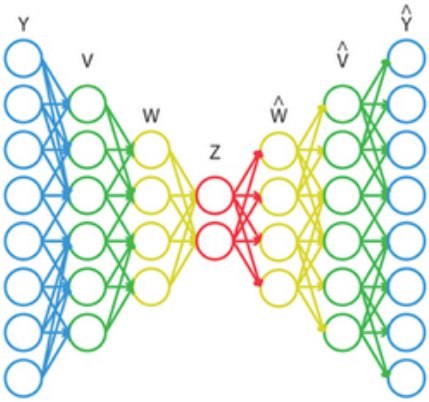


Figure 2.4: Deep Autoencoder (Berniker and Kording, 2015)
This deep autoencoder reduces the data set from 8 features down to 2. By adding multiple layers, this process is more gradual and aims to keep the more important patterns in the data set.

Over-Complete Autoencoder

When the data set is too sparse, an autoencoder with an over-complete layer can help fill the missing data points. An over-complete layer has more nodes than the original data set, and this can help predict the missing parts of the original data set. In an over-complete deep autoencoder, one layer can be greater than the original number of features, before reducing down to the smallest latent space.

t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is used for dimensionality reduction and data visualization (va der Maaten and Hinton, 2008). It minimizes the Kullback-Leibler (KL) divergence between the data distribution and embedded distribution by measuring the distance or dissimilarity between data points and converts them to conditional probabilities. Points that are more similar are grouped closer together, making it easier to see which values are similar in a 2D or 3D space. Figure 2.5 shows the difference in dimensionality reduction using t-SNE and PCA on the same data set. t-SNE makes it easier to see how similar values are grouped together than PCA.

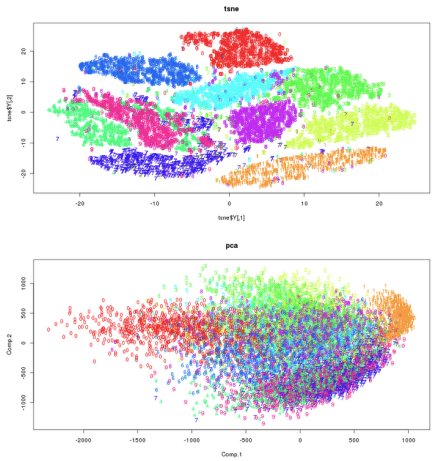


Figure 2.5: t-SNE vs PCA (R-Bloggers, 2017)
The two graphs show dimensionality reduction of the same data set with colors representing which cluster they are assigned to. While both reduce the number of features/dimensions, t-SNE aims to visualize the segregation of the data by grouping similar data points together.[This figure is best viewed in color]

While useful for visualization, t-SNE should not be used for dimensionality reduction in K-Means. There are several parameters in the t-SNE algorithm that drastically change the results, therefore the patterns in the t-SNE results may not be real patterns in the data set. Altering one parameter may cause the visualization to show relationships that do not actually exist in the data (Distill, 2016). Figure 2.6 shows the original data set of random values from a Gaussian distribution, and what it looks like after running t-SNE with perplexity 2. Some points seem clustered together when they are in reality random. Consequently, t-SNE is only used for appealing visualization, not for further analysis.

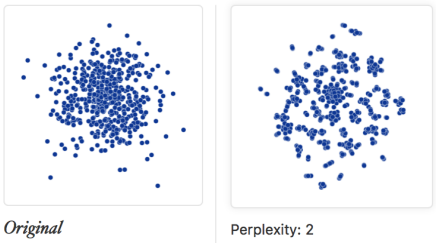


Figure 2.6: t-SNE with perplexity= 2 (Distill, 2016)
t-SNE sometimes shows patterns in the data that do not exist. Above, the original data set is a random Gaussian distribution, but t-SNE makes it appear to group several points together.

2.5 A/B Testing

In the practical application of this paper, we compare the dimensionality reduction methods and choose one to cluster using K-Means. We then compare the different clusters' reactions to A/B testing. A/B testing is an experimental process commonly used in web or tech based analytics, in which the population is randomly divided into a control (A) and test (B) group. The control group is kept the same, while the test group is given a new feature. The difference-in-differences is measured to see if there is a positive or negative reaction to the new feature, controlling for change over time. This is generally measured by changes in some Key Performance Indicators (KPI), such as number of clicks or amount spent.

3 Game Theory

Game theory studies the interaction of multiple decision makers. It analyzes the action of multiple players in maximizing their payoffs or utilities. The framework described in this section can be used to model K-Means algorithm in Section 4.1. In their research, Bulò and Pellilo use game theory to model a hypergraph cluster algorithm. Similarly, we use game theory to model the K-Means cluster algorithm, but with further analysis on the game theory framework.

A set of players interact with each other in a strategic form game. Each player i chooses a strategy, and the combination of strategies will result in utility for all the players. Each player can choose a pure strategy to maximize utility π_i, or choose several in a mixed-strategy with certain probability distribution to maximize her expected utility $E(\pi_i)$. Two payoff matrices of a 2 player game are shown in Table 3.1. The first value is the utility of player 1, with each strategy row labeled on the left. The second value is the utility of player 2, with each strategy as a column labeled above the utility grid. In the first game on Table 3.1a, it is optimal for player 1 to choose a strategy that is the same as player 2, while player 2 wants to choose a strategy that is the opposite of player 1. In the classic Prisoner's Dilemma depicted in Table 3.1b, it would be optimal for both if they both chose Quiet than both betrayed the other. However, if one chose Quiet, the other could betray to get even higher utility.

Table 3.1: Two Player Games

(a) Left Right

	Left	Right
Left	1, -1	-1, 1
Right	-1, 1	1, -1

(b) Prisoner's Dilemma

	Quiet	Betray
Quiet	2, 2	-1, 3
Betray	3, -1	0, 0

3.1 Types of Games

There are several different categories of games, as well as extensive form games that include additional aspects or qualities to the game. They can drastically change the framework and the strategies players choose.

Cooperative vs Non-Cooperative

Cooperative games force players to form coalitions and commit to their strategy. They require some sort of binding agreement, such as contracts, that ensure that the agreed upon strategy is played, even if a player wants to deviate. In non-cooperative game theory, there are no such binding rules. Players choose a strategy that is utility maximizing, typically under Nash equilibrium (see Section 3.2).

Symmetric vs Asymmetric

Symmetric games are when all players have the same payoff function. In Table 3.1, Left Right is asymmetric: player 1 and player 2 receive different payoffs when playing the same strategy, and switching strategies does not switch payoffs. Prisoner's Dilemma is symmetric: When player 1 chooses Betray while player 2 chooses Quiet, player 1 receives the same payoff as player 2 would if player 1 chose Quiet and player 2 chose Betray.

Simultaneous vs Sequential

All players choose strategies at the same time in simultaneous games, while players take turns in sequential games. Table 3.1 games are simultaneous, but can be converted to a sequential game where one player goes first. Sequential games can have perfect information (in which all players know which strategies have been chosen previous to theirs), or non-perfect information (in which some players do not know all of the previous decisions).

One-Shot vs Repeated

A One-shot game is only played once, while repeated games play multiple or even infinite times. In repeated games, players may choose a strategy that takes into account the past and the future. They may play a different strategy

depending on past history of other players, or play certain strategies to signal a favorable reputation and gain more utility in subsequent rounds. In Table 3.1b, players will be more likely to choose Quiet if they know they will be playing the game again. The increase in payoff from Betraying once will be outweighed by the potential of getting (Quiet, Quiet) instead of (Betray, Betray) in subsequent periods.

3.2 Nash Equilibrium

In non-cooperative game theory, Nash equilibrium (NE) is a set of strategies in which no player is able to deviate from her strategy and increase (expected) utility, given all other players play the NE strategy (Nash, 1950, 1951). Formally put, a game consists of a set of players $C = \{c_1, c_2, ...c_k\}$. Each player c_j plays a strategy s_j from her set of possible strategies S_j to maximize expected utility:

$$\max_{s_j \in S_j} E(\pi_j(s_j, s_{-j}))$$

where s_{-j} denotes the strategies of all players except player j. A NE is a set of strategies for every player $\{s_1^*, s_2^*, ...s_k^*\}$ such that no player can deviate to increase utility, given that all other players play the NE strategy. For all players $c_j \in C$ and all other strategies $s_j' \in S_j$:

$$E(\pi_j(s_j^*, s_{-j}^*)) \geq E(\pi_j(s_j', s_{-j}^*))$$

In the game shown in Table 3.1a, the NE is the mixed strategy $s_1^* = (0.5, 0.5)$, $s_2^* = (0.5, 0.5)$. Both players need to play completely randomly to prevent the other one from deviating to another action more often. If player 1 plays Left more often, player 2 would play Right more often, to player 1's detriment. Likewise, if player 2 played Left more frequently, player 1 would play Left more frequently as well, hurting player 2's expected utility.

In the Prisoner's Dilemma in Table 3.1b, the NE is the pure strategy $s_1^* = (0, 1)$, $s_2^* = (0, 1)$. That is, both players will always choose to betray. This is because if they choose Quiet, it would be better for either to deviate to Betray. Therefore they will both end up choosing (Betray, Betray), even though (Quiet, Quiet) is better for both players.

If the game is repeated infinitely, however, the Prisoner's Dilemma may lead

to a (Quiet, Quiet) NE. If both players care about their future utility, they will be better off choosing to cooperate and choose Quiet to assure that the other player also plays Quiet. This is similar to cartel collusion in industrial organization. Players do not "betray" the other companies in the cartel by decreasing the price in the market, due to threat of the other companies also lowering prices in the subsequent periods down to competitive prices:

$$\pi_{collusion} = 2 + 2\delta + 2\delta^2 + ...$$

where δ is the discount factor per period (how much the future matters compared to the present).

$$\pi_{deviation} = 3 + 0\delta + 0\delta^2 + ...$$

will not be worth it if $\pi_{collusion} > \pi_{deviation}$, or $\delta > \frac{1}{3}$.

4 The Model

In the following section, we detail the K-Means algorithm and its process. We then present a game in which each player is a cluster with utility based on inter-cluster similarity and how dimensionality reduction is modeled under game theory. Finally, we show that K-Means strategy results in a Nash equilibrium in this game.

4.1 *K*-Means Algorithm

K-Means is one of the most commonly used algorithm in cluster analysis (MacKay, 2003). It takes a data set $\{x_1, x_2, ...x_n\}$ comprised of n points of length m, with m denoting the number of features or variables in the data set. The algorithm groups the points into k clusters using the function that maps $f : \mathbb{R}^m \to \mathbb{R}^m$, from the point to the location of the nearest cluster. K-Means algorithm aims to find the local minimum of the sum of squared errors from the data points to its assigned cluster's centroid:

$$\min_{c_j} \sum_{i=1}^{n} d(x_i, f(x_i))^2 = \min_{c_j} \sum_{j=1}^{k} \sum_{x_i \in S_j} d(x_i, f(x_i))^2$$

where $d(x, y) = ||x - y|| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_m - y_m)^2}$ is the Euclidean distance function and S_j is the set of points in Cluster $j \in \{1, 2, ...k\}$. After choosing the value k, the K-Means iteration process is as following:

- ☐ *Iteration 1*: Cluster centers $\{s_1, s_2, ...s_k\}$ are chosen randomly. Function $f(x_i) = s_j$ takes the data set and uses the distance function to find the closest cluster center, and assigns each point with that cluster.

- ☐ *Iteration 2*: The mean value of the data points in each cluster is calculated as the new cluster centers. For each cluster strategy $s_j \in S$:

  $$s_j = \mu_j := \frac{1}{|S_j|} \sum_{x \in S_j} x$$

  where $|S_j|$ is the cardinality of S_j. Data points are assigned to the closest cluster with the new cluster centers.

- ☐ *Iteration 3, 4, 5...*: The process from Iteration 2 is continued for a predetermined number of iterations, or until a stable equilibrium is reached.

In practice, iterating over the mean of each cluster's assigned points leads to an equilibrium that minimizes the sum of square errors. Figure 4.1 shows 6 iterations of this process in a random data set of two dimensions $x = [-2, 2]$ and $y = [-0.5, 3]$ ($m = 2$). Because the first random iteration cluster centers are all chosen close to $(0, 2.5)$, it takes several iterations for the centers to spread out and reach equilibrium.

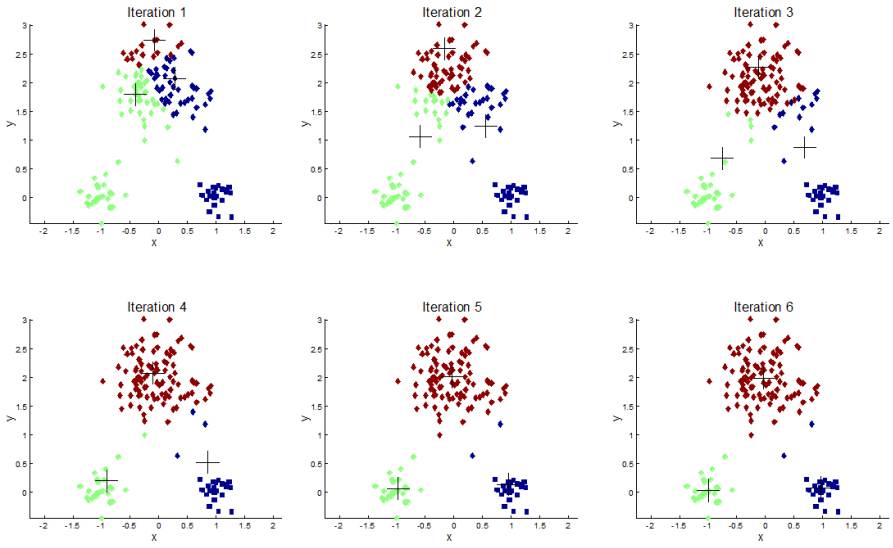


Figure 4.1: K-Means Iterations (Pandre)
The figure above shows 6 iterations of the K-Means process. The 3 cluster centers start from random locations and gradually iterate to the optimal locations.

4.2 The Game

K-Means clustering can be framed as a k player game. It is a non-cooperative, symmetric, simultaneous, repeated game. Although the game is run for multiple iterations to reach equilibrium, we only calculate utility from the payoffs after stable equilibrium is reached and the strategy no longer changes. All previous iterations are exploratory "searching" of the equilibrium, and frame the strategy set of each player for the next period.

The game takes a data set and divides it among the players. Utility is then measured by intra-cluster similarity, or how similar the values in each cluster are to each other. Each player chooses a strategy by choosing location in the m dimensional space to maximize utility. Each player chooses the strategy that will minimize its average distance from its centroid to its assigned data points. Therefore, the clusters must find a strategy that will divide all the points in the data set between the players, and consider how this will affect its centroid, which is the mean of its cluster values. Like most games, the strategy of each player affects the utility of other players.

Put formally, the game consists of k players with their strategies $\{s_1, s_2, ...s_k\}$ and a set of points $\{x_1, x_2, ...x_n\}$. Each point has m values for each variable in the data set. Each player chooses a location in the space $\mathbb{R}^m$. Afterwards, the data set is partitioned to the closest cluster using the function $f(x_i)$. Each player maximizes her utility by maximizing on the negative sum of square error, or minimizing the sum of square error:

$$\max \pi_j = \max_{s_j} \sum_{x_i \in S_j} -d(x_i, s_j)^2 = \min_{s_j} \sum_{x_i \in S_j} d(x_i, s_j)^2$$

The strategy set is continuous rather than discrete, but is limited by the minimum and maximum value found for each feature. This way, players cannot optimize by choosing a location that is too far from the data set to be allocated any points, therefore having a "mean distance" of zero. That is, each player must be allocated at least one data point.

Lastly, we limit the range of movement from one location to inside the space of its assigned points from the previous iteration. This is to prevent sudden random movement, and is akin to the learning rate of multiple iterations. By limiting the strategy set, it decreases the number of iterations towards equilibrium. In this sense, past history is important. The random starting points of the K-Means process may change which final NE is reached.

4.3 Dimensionality Reduction

In the game theory framework, dimensionality reduction is akin to reducing the strategy set from a m dimensional point to size m^*, making computation

easier and reducing noise, but also losing detailed information. Dimensionality reduction methods in K-Means reduces the strategy space of the game from $\mathbb{R}^m$ to $\mathbb{R}^{m^*}$. While it simplifies the game strategy set, it also reduces some of the details of the game and fundamentally changes the game structure. Neural networks such as autoencoders can help limit the amount of information lost when reducing dimensions, compared to manually excluding certain variables from the game.

4.4 *K*-Means Strategy

Once equilibrium is reached using the K-Means algorithm, the cluster centers are stabilized to the mean values of the points in each cluster. To be a NE, each cluster cannot unilaterally move to increase utility, given all other clusters play the same strategy.

To be a Nash equilibrium of the game, it must not be optimal to deviate unilaterally. For K-Means to reach NE, it is key that the strategy set is limited to within the area of its current cluster points. In this way, it prevents one cluster from choosing the location of another cluster.

Suppose Cluster j chooses alternate strategy s', which is a distance of $\bar{d} = \{d_1, d_2, ...d_m\}$ away. By changing strategy from $s_j^* = \mu_j$ to another location s'_j, at least one of the following things will happen to Cluster j:

1. Keep the same set of points
2. Move closer and gain set of points X'
3. Move further away and lose assignment of set of points X''

1. For a given set of points, the mean value is the utility maximizing strategy.

Proof *Given a set of points assigned to each cluster, K-Means finds the local maximum utility.*

$$\pi_j = -\sum_{x_i \in S_j} d(x_i, s_j)^2 = -\sum_{x_i \in S_j} ||x_i - s_j||^2$$

$$= -\sum_{x_i \in S_j} \left(\sqrt{(x_{i,1} - s_{j,1})^2 + (x_{i,2} - s_{j,2})^2 + ... + (x_{i,m} - s_{j,m})^2}\right)^2$$

where $x_{i,l}$ is the value of the lth dimension for point x_i. The utility is then equivalent to

$$= - \sum_{x_i \in S_j} \Big((x_{i,1} - s_{j,1})^2 + (x_{i,2} - s_{j,2})^2 + ... + (x_{i,m} - s_{j,m})^2 \Big)$$

Because each $(x_{i,l} - s_{j,l})^2$ values are always positive, where the square root to the second power is equivalent to the identity function. Taking the first order condition:

$$\frac{d\pi}{ds_j} = \sum_{x_i \in S_j} \Big(2(x_{i,1} - s_{j,1}) + 2(x_{i,2} - s_{j,2}) + ... + 2(x_{i,m} - s_{j,m}) \Big) = 0$$

$$\sum_{x_i \in S_j} \Big(2(x_{i,1} - s_{j,1}) \Big) + \sum_{x_i \in S_j} \Big(2(x_{i,2} - s_{j,2}) \Big) + ... \sum_{x_i \in S_j} \Big(2(x_{i,m} - s_{j,m}) \Big) = 0$$

For each dimension $1, 2, ...m$, the summation becomes zero when $s_{j,l} = \mu_{j,l}$. Therefore the utility maximizing strategy is the local maximum of the utility function.

2. For each point in set X', assigning it to Cluster j will decrease its utility. Even if the cluster moves closer to other points already assigned to itself, it will be increasing distance for at least one point (since $s_j = \mu_j$ implies clusters on both sides of every dimension). Therefore Cluster j would not change strategies to gain more points.

3. When a point moves to decrease the number of assigned points, it is crucial to consider the "punishment" strategy of the other clusters. Because we have switched the framework from an algorithm to a game with multiple players, strategies must account for the reaction of other players.

For a point $x' \in X'$, let us denote the difference between the K-Means cluster strategy and new deviation strategy as d. This strategy loses assignment of one point x_i that would otherwise be in the K-Means strategy s_j^*, and utility changes from π_j^* to π_j'.

In this case, it is crucial that the game is repeated, and other clusters have the chance of "punishing" this deviating strategy. Given the clusters were previously at equilibrium - that is, $s_j = \mu_j$ for all $j \in k$ - point x_i must have previously been further than d^* from all other clusters. The utility of the second closest cluster g would decrease by $||x_i - s_g||^2$ as it would now be assigned

to the point, so Cluster g can discourage this move from Cluster j by simply moving $-\bar{d} = \{-d_1, -d_2, ... - d_m\}$ away from this point, assigning point x_i back to Cluster j. It is a NE in the sense that Cluster j will not deviate from s^* because this punishment would decrease utility. This punishment is not always a sub-game perfect strategy - if the strategy is not credible and Cluster g may lose utility by punishing Cluster j.

The key difference between the K-Means algorithm and the game theoretical K-Means strategy is that we must also account for possible deviations. Besides deviation strategies, the K-Means strategy and original algorithm are very similar to each other, only differing in framework. Using game theory to explain the clustering process, we created an economical interpretation of a primary statistical method in data science. With this insight, we applied the K-Means algorithm in a corporate setting.

5 Business Application - Method

Using the new game theory interpretation of K-Means algorithm, we hope to have a better understanding in applying cluster analysis to a business case. We applied the described method at *King Digital Entertainment plc*, an international mobile game company with headquarters in Stockholm, Sweden. We focused on the data of who play *Candy Crush Saga*, *King*'s most popular game. It is a free-to-play game with millions of daily active users.

We hope that the combination of K-Means algorithm and dimensionality reduction methods can segment the gamers base into meaningful profile types and provide more comprehensive insight to different types of customers, rather than categorizing customers based on one or two variables.

In the practical application of the game theory K-Means framework at *King*, we first use a small data set ($n \approx 10,000$) to compare the different types of dimensionality reduction methods on static, activity, and combination data sets. After comparing their silhouette score and applicability, we use the selected models to predict clusters for the A/B test population in the period before the A/B test, trained with a larger data set ($n \approx 100,000$). Lastly, we assign the A/B test population ($n \approx 10,000,000$) to different clusters and compare the difference in reactions to the A/B test. To make up for multiple comparison or p-hacking correction, we used $p = 0.01$ significant values instead of $p = 0.05$.

Because the game theoretical setting and the data science application at *King* use similar terms such as games and players, the following table shows the equivalence between game theory and what these values are at *King*:

Variable	Game Theory	K-Means at *King*
$\{c_1, c_2, ...c_k\}$	(set of) strategies	(set of) clusters
k	number of players	number of clusters
m	strategy dimensions	number of features
$\mathbb{R}^m$	strategy space	feature space
$\{x_1, x_2, ...x_n\}$	used to calculate utility	data set of gamers
$\sum_{x \in C_i} d(x, c(x))^2$	utility function	K-Means function
$\{\mu_1, \mu_2, ...\mu_k\}$	(set of) strategies values	(set of) cluster centers

5.1 Data

We used two data sets of *Candy Crush* gamer information: static and time-series. The static data set contained $m = 91$ demographic and aggregated information (ex: total number of games played), indexed by unique player identifiers. The static cluster analysis is used to see how fundamentally different players (casual vs. intense, advanced vs. beginner) react to A/B tests.

We also created a time-series data set of each gamers' *Candy Crush* activity over the span of 4 weeks. We took the weekly average of 4 weeks of game play, and summed up 3 hour periods into one feature. This was done to stabilize values and mitigate the effects of unusually high play. For example, Feature 19 comprised of the amount of game activity from noon to 3 P.M. on a Wednesday, from calendar week 1 to week 4 of 2017. We chose to divide each activity by the total amount over the week to represent the distribution of play, rather than the absolute amount (which is captured by the static data set). This helps us analyze if the time of play is important for A/B tests involving a waiting period.

To choose the optimal model to cluster the data, we use a data set of $n = 10,000$ to $20,000$ active *Candy Crush* gamers. After deciding on a dimensionality reduction method, we increased the same size to $n = 500,000$ for static data and $n = 100,000$ for time-series data to fit the model parameters. Both data sets were limited to active players only, defined by at least an hour of play during the period of interest.

5.2 A/B Test

We used an A/B test to measure the business value of the clusters, by comparing the differences in the change in important business factors called Key Performance Indicators (KPI). If clusters show vastly different reactions to an A/B test, it may be beneficial to implement the feature to only the clusters that show a positive reaction. This A/B test was implemented February 2, 2017, and it examines the effect of lowering the friction of additional game play. A feature that limited game play was kept for the control group, and taken out for the testing group in this A/B test.

We looked at a KPI that measured activity level. Overall, the test group showed an increase in activity after this A/B test feature, but we hope to capture further information on which groups showed a higher increase in this KPI.

6 Results

6.1 Dimensionality Reduction

We compared the silhouette score of $k = 10$ and $k = 16$ for various dimensionality reduction methods. PCA and autoencoders were used to reduce each data set down to 16 features. We found that dimensionality reduction methods improve silhouette score over K-Means on the static data set. Although manually removing features had the highest silhouette score, it also reduced important information contained in those features. Table 8.1 in the Appendix shows the silhouette score of static data with 16 clusters as well as 10 for comparison and robustness in a sample of $10,000$ gamers, as well as the ratio between the largest and smallest cluster when $k = 10$.

For both the static and time-series data sets, the more advanced dimensionality reduction allowed more patterns to be captured for each cluster. Although this decreases variance by individual features, overall variance increases and clusters identified players differing over a set of variables, not just one or two. Dimensionality reduction methods also helped even out the distribution of players, because gamers with extreme values were mitigated when using the dimensionality reduction methods to find over-arching patterns in the data, rather than creating clusters of outliers. We decided to use a deep autoencoder with an over-complete layer for both static and time-series data, as this has the most even distribution of the data set between clusters.

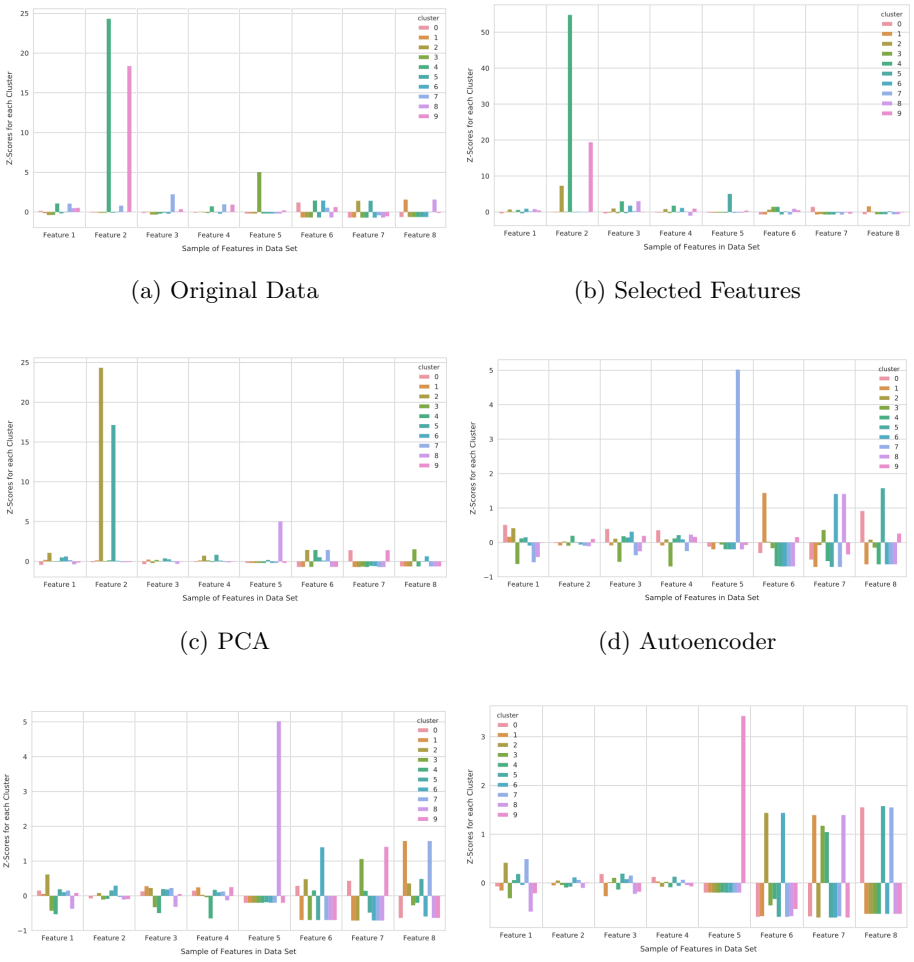
Static

Figure 6.1 shows bar plots displaying the different cluster results when using a large data set of static information on gamers. Each set of $k = 10$ bars show the average z-score of each feature. The z-score was used to show multiple variables at once, and for confidentiality reasons.

Figure 6.1a shows the cluster results after running K-Means on the original data set after standardized to z-score. Figure 6.1b shows the cluster results of

the data set of only selected features, while Figure 6.1c shows K-Means cluster results after using PCA down to 16 features. Figures 6.1d - 6.1f show the results of K-Means after encoding $m = 56$ to the mirroring layers {16}, {30, 16, 30}, and {70, 50, 30, 16, 30, 50, 70}, respectively, and using the smallest latent space $m^* = 16$.

While the level of variance between different model change drastically, the autoencoders added additional value in evening out the size of the clusters. In the original data set, a few outlier players ended up in their own cluster, due to their extreme z-scores (for example, Cluster 5 in Figure 6.1a contains only 1 gamer with z-score of over 20 for Feature 2). Using the autoencoders, the highest z-score was around 3.5, and the number of players in each cluster was quite even. Because gamers were clustered based on their overall profile, not just one extreme variable, their z-score was lower but more meaningful, and there was variance along all the variables, not the one of two highly skewed variables.



(a) Original Data

(b) Selected Features

(c) PCA

(d) Autoencoder

(e) Deep Encoder

(f) Deep Encoder (over-complete)

Figure 6.1: Dimensionality Reduction for Static, Active Gamers
Each bar plot shows the differences between the $k = 10$ clusters on 8 selected features. The use of autoencoders drastically even out the distribution of players between clusters, shown by the lack of extreme clusters like Cluster 5 in Figure 6.1a. [This figure is best viewed in color]

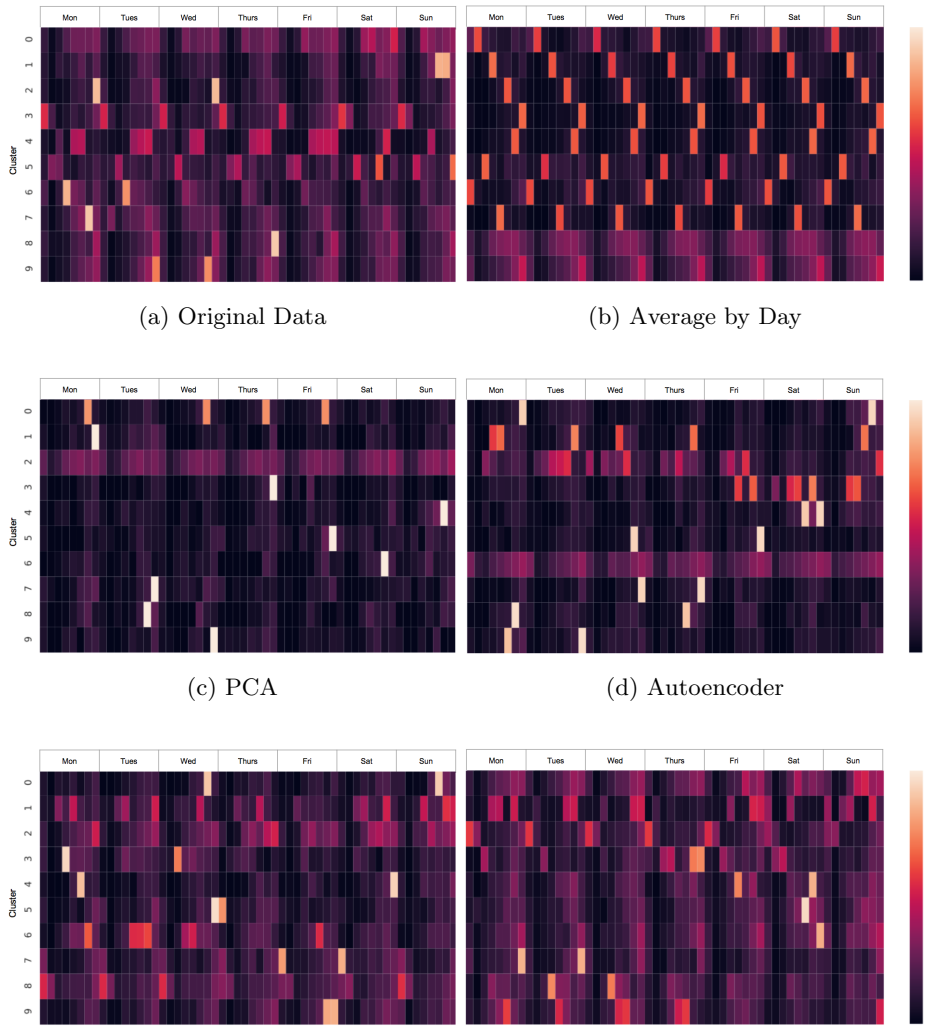
Time-Series

When comparing models in the time-series data set, we found that taking the daily average of everyday vastly improved silhouette score. However, this was mostly because of the reduction in number of features and simplification of the

data set. Because the daily average does not differentiate between those who play more on weekends and more on weekdays (which could have crucial impact on the A/B test), we decided to not use this model. PCA had the highest silhouette score, but there was too much variance between cluster sizes. For example, static data was clustered based on highly right-skewed variables, where outliers had z-scores over 4. This lead to one cluster with around 80% of the data set. When using a data set of active gamers, this problem was drastically reduced. However, using dimensionality reduction methods on active gamers still helped even out the size of the different clusters.

To show the value of using dimensionality reduction methods, Figure 6.2 shows a heatmap displaying the activity of the different clusters when using the time-series data of activity of all gamers. Each row represents one cluster, and each column is a 3 hour period over a week. For example, the first row and first column represents the average play on Monday from midnight to 3 A.M. over a span of 4 weeks for Cluster 0. The color scales range from no (black) activity to high (light yellow) activity. While cluster analysis on the original data set and PCA only captures anomalies in a specific time period (for example, the last cluster for the original data K-Means have very high play Sunday 9 P.M. to midnight), daily averages and encoders can capture time-series trends. Clusters shown in Figures 6.2a, 6.2c, and 6.2d had one cluster that contained over half the data set - the most colorful cluster that capture all players without a strong activity concentration in one time period (Cluster 1 in Figure 6.2a, Cluster 2 in Figure 6.2c, Cluster 6 in Figure 6.2d). All other clusters were characterized by a single or a few period of high play.

Although taking the average by day captures daily trend, it loses information on the difference between days, for example between weekdays and weekends. The deep encoders, particularly the deep encoder with an over-complete layer of $m' = 70$, were more able to capture trends throughout the week. The deep encoders were more able to capture overall trends, since they are more distributed between clusters by patterns than a single period of high play. This shows the value of autoencoders' abilities to capture time-series patterns over only using K-Means or PCA.



(a) Original Data (b) Average by Day

(c) PCA (d) Autoencoder

(e) Deep Encoder (f) Deep Encoder (over-complete)

Figure 6.2: Dimensionality Reduction for Time-Series, All Gamers
Each heatmap contains a row for each of $k = 10$ clusters, with a cell for the amount of activity in each time period. The row spans a week of activity, and contains all gamers, both active and inactive.

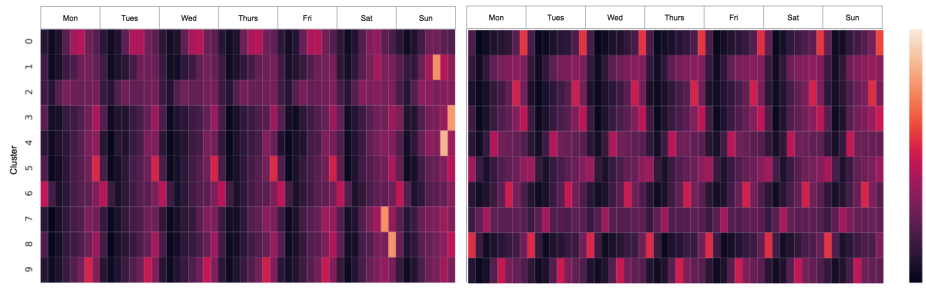
Figure 6.3 shows the same information when limiting the data set to active gamers (at least a hour of play during the 4 week period). Figure 6.3a shows the clusters using simple K-Means on the entire activity data set. Using only active gamers, you can already start seeing daily trends. For example, Cluster

0 shows gamers who play on the afternoon of every day, while Cluster 9 gamers seem to play more in the late evening. Figure 6.3b shows the cluster results of the data set after summing the same period each day (ex: sum of noon-3 P.M. for the entire week), so that the data set is reduced to daily information instead of weekly. When using daily averages, it is hard to capture daily differences, such as higher level of play on the weekends. Figure 6.3c shows K-Means cluster results after using PCA down to 16 features. Figures 6.3d-6.3f show the results of K-Means after encoding $m = 56$ to the mirroring layers {16}, {30, 16, 30}, and {70, 50, 30, 16, 30, 50, 70}, respectively, and using the smallest latent space $m^* = 16$.

Using only active gamers, all cluster results show a daily pattern. However, the deep autoencoders add additional value in showing weekday vs. weekend differences. Table 8.2 in the Appendix shows the silhouette score and cluster ratio of normalized activity data using only active gamers. Although the loss decreased with more autoencoder layers, the silhouette score also decreased, and we found that this was because the cluster sizes were more evenly distributed in the autoencoders with more layers.

6.2 Results on A/B Test

The final model chosen for A/B Test analysis was the deep autoencoder with over-complete layer. For the static data set, $m = 91$ features were first encoded to 120 features, then down to 50, 30, then the final latent space of 16. For the time-series data set, $m = 56$ features were first encoded to 70 features, then down to 50, 30, then the final latent space of 16. The decoding layers mirrored the encoding layers. They were then clustered using K-Means separately and together. A t-SNE visualization of the encoded data set are shown in Figure 6.4. While the static data set is quite segregated after encoding, the time-series data set is still close together.



(a) Original Data

(b) Average by Day

(c) PCA

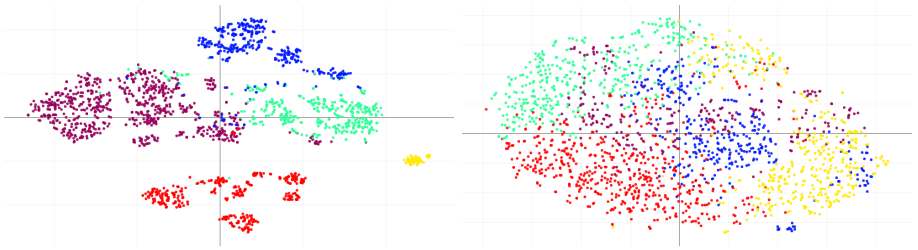
(d) Encoder

(e) Deep Encoder

(f) Deep Encoder (over-complete)

Figure 6.3: Dimensionality Reduction for Time-Series, Active Gamers

Each heatmap contains a row for each of $k = 10$ clusters, with a cell for the amount of activity in each time period. The row spans a week of activity.



(a) Static Data (b) Time-Series Data

Figure 6.4: t-SNE of Latent Space, $k = 5$
The data sets are first encoded down to 16 features using a deep autoencoder with an over-complete layer. We then used t-SNE to reduce the latent space further to 2 features. The Static data set is quite segregated, while the time-series data set is not. [This figure is best viewed in color]

Static Results

For the static data set, we chose the DAE with an over-complete layer as it had the least variance of cluster sizes. From $m = 91$ features, we expand to $m' = 120$, then reduced the number of features to $m'' = 50$, $m''' = 30$, then finally to the latent space $m^* = 16$. We cluster the A/B test population to $k = 5$ based on the silhouette score and applicability for further analysis. (see Table 8.5). We found that the cluster sizes were still highly varied with $k = 5$ having one cluster with 40% of the data set, while the smallest had 2%. Figure 6.4a shows the t-SNE graph for the latent space. While varied in size, the clusters are highly segregated.

A bar plot of the clusters in Figure 6.5 shows the z-score of the each cluster on various variables. As seen from the dimensional reduction section, the difference between clusters is not very high. The deep autoencoder mitigates the effect of extreme outliers.

The heatmap in Figure 6.6 shows each cluster's average play per 4 hour period, with one row for each cluster. It shows the distribution of the period when they play. The time-series of these clusters are quite similar to each other, which is consistent with the fact that the data set used to cluster did not contain information on time of play. It implies that static information do not correlate

highly with the time of day in which gamers play.

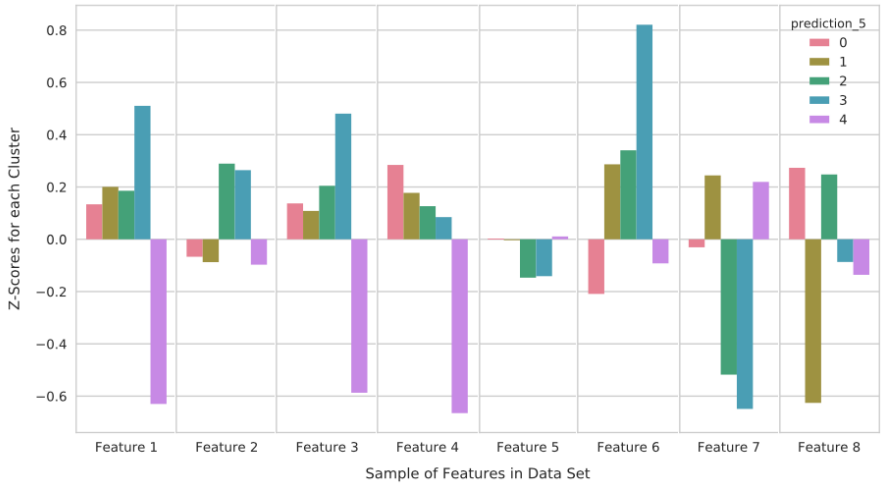


Figure 6.5: Bar Plot, Static Clusters

The variance between clusters is shown for selected features. Cluster 3 contains the gamers with the highest values in Feature 1 and 3, while Cluster 4 shows the opposite profile. [This figure is best viewed in color]

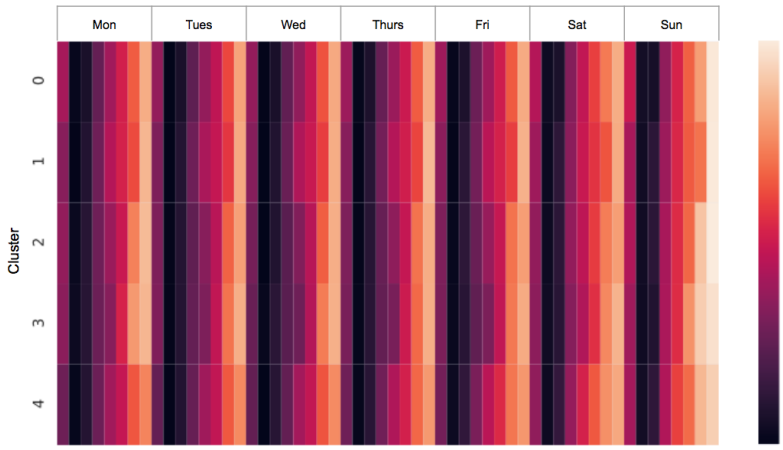


Figure 6.6: Heatmap of Static Clusters, $k = 5$
Each row represents the average time-series for each cluster, over the span of a week. There is very little variance between time-series activity for these clusters, because this information was not used in the clustering.

In Figure 6.7, the cumulative KPI uplift values are shown for each cluster, as well as the 99% confidence interval of these values. The clusters shown slightly different reactions to the A/B test. Cluster 4 in particular seem to be showing a higher increase in KPI compared to the other clusters.

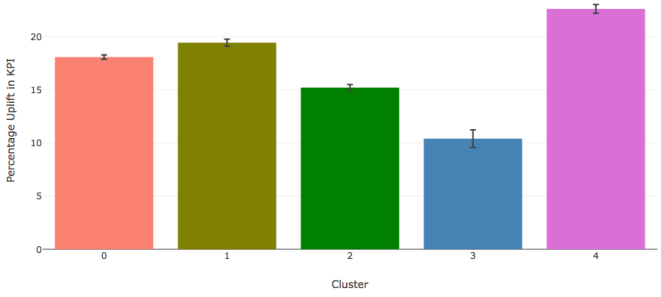


Figure 6.7: 99% Confidence Intervals, Static Clusters
The bars represent the cumulative KPI uplift for each cluster, as well as the 99% confidence interval of these values. Cluster 4 in particular seem to be showing a higher increase in KPI compared to the other clusters.

Time-Series Results

Using only active gamers, we chose to use the DAE with an over-complete layer, because it has the most even distribution of the data set into different clusters. We then clustered the latent space of $m^* = 16$ into $k = 5$ clusters, due to its high silhouette score and applicability for further A/B test analysis. It had an even distribution of cluster sizes, with the biggest cluster containing 28% of the data set while the smallest contained 13%. The t-SNE graph in Figure 6.4b shows that the latent space is still quite consolidated. In particular, the purple cluster is scattered throughout different clusters.

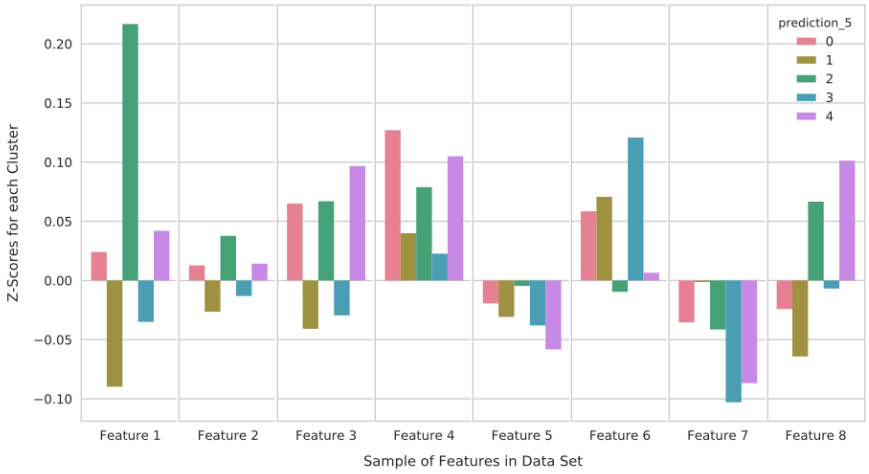


Figure 6.8: Bar Plot, Time-Series Clusters
The variance along static data is smaller for time-series cluster, because these features were not used for the clustering process. [This figure is best viewed in color]

A bar plot of the clusters in Figure 6.8 shows the z-score of the each cluster on selected variables. As these variables are not part of the data set used to cluster, there is almost no difference between the clusters.

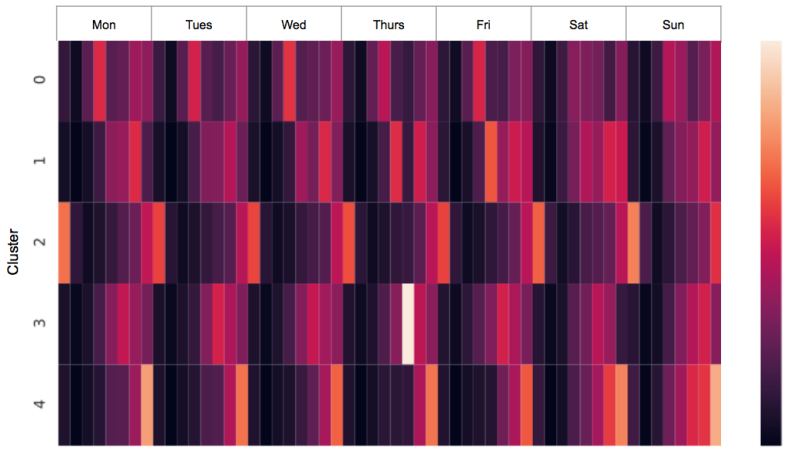


Figure 6.9: Heatmap of Time-Series Clusters, $k = 5$
Each row represents the average time-series for each cluster, over the span of a week. Cluster 0 shows "commuter" players, Cluster 1 and 3 show evening players, and Cluster 2 and 4 show night-time players.

The following heatmap shows each cluster's average play per 3 hour time period, with one row for each cluster. Figure 6.9 shows the distribution of the period when they play. Starting from Column 1 - Monday from 0 to 3 A.M to Column 56 - Sunday 9 P.M to 12 A.M, we can see a different daily trend for each day. Cluster 0 shows high level of play 9 to 12 P.M, as well as a secondary increase in activity 6 to 9 P.M. There is also a low but dispersed level of activity over the weekend. This implies this cluster consists of "commuter players," or players that play during their commute to work. Cluster 1 consists of afternoon to evening players who play during their free time. Players in Cluster 2 play late into the night, from 9 P.M. to 3 A.M. Cluster 3 also play in the afternoon, but less in the evening. Lastly, Cluster 4 consists of night players that play up until midnight, but rarely after.

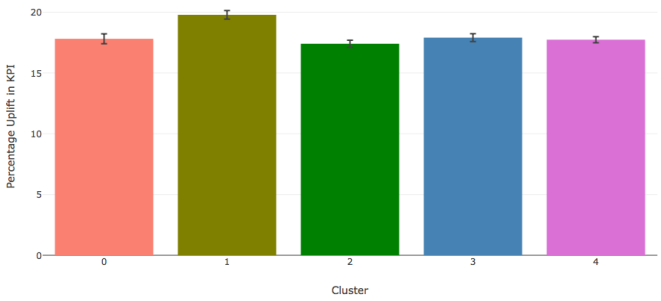


Figure 6.10: 99% Confidence Intervals, Time-Series Clusters
Each bar represents the cumulative uplift in KPI after the A/B test. They are quite similar to each other, and only Cluster 1 is statistically significantly different from the other.

Figure 6.10 shows the 99% confidence interval for each cluster uplift from the A/B test. Although Clusters 0, 2, 3, and 4 all overlap each other, Cluster 1 shows significant different behavior. While the KPI uplift between clusters were not dramatically different, they showed statistically significant differences.

7 Conclusion

In this thesis, we use a game theoretical framework to bring a novel perspective into cluster analysis in economics. Although unsupervised methods such as cluster analysis are seldom used in economics, they can bring great value in finding similarities and patterns beyond those found using common econometric tools such as linear regressions. Using dimensionality reduction methods, large data sets of both aggregate and time-series information can be compressed to underlying patterns within the data, and the data set can be meaningfully partitioned across multiple variables. This gives less opportunity for p-hacking or forking, as it replaced partitioning the data set on various combinations of researcher-chosen variables. Stemming from a similar framework, both Nash equilibrium strategies in game theory and clustering algorithms optimize on a function that leads to stabilizing equilibrium. In particular, the K-Means algorithm can be framed as a k player game, where the equilibrium found is a NE when the cluster's utility function is the negative of the distance function used in K-Means.

There is a large potential for further research exploring other aspects of data science in an economics framework, particularly game theory. For example, evolutionary game theory provide a comprehensive framework for definitions of equilibrium stability measured by the amount of mutation that still leads back to the evolutionary stable Nash equilibrium. This can be used to measure cluster algorithm stability, a valuable but understudied aspect of clustering robustness (von Luxburg, 2010). Alternatively, more data science methods could be used for econometric analysis. Classification can be used to label new industries, businesses, or economies to a particular category. Anomaly detection can be used to measure cartel collusion or signs of economic downturn. Expanding on the analysis of this thesis, dimensionality reduction and cluster analysis on experimental game theory and econometrics can help segment the population into different utility types. Advanced methods such as recurrent and convolution neural networks could be used to find patterns in more complicated multivariate data sets. Further cooperation of data science and economics can bring new and valuable insight and influence the advancement of both fields.

Bibliography

Aggarwal, C. C. and Reddy, C. K. (2013). *Data clustering: algorithms and applications*. Chapman & Hall/CRC.

Benjamin, D. J., Berger, J. O., and Johnson, V. E. (2017). Redefine statistical significance. *Nature Human Behavior*.

Berniker, M. and Kording, K. P. (2015). Deep networks for motor control functions. *Frontiers in Computational Neuroscience*, 9(32).

Best, M. C., Hjort, J., and Szakonyi, D. (2017). Individuals and organizations as sources of state effectiveness, and consequences for policy design. *NBER Working Paper*.

Brownlee, J. (2017). Gentle introduction to the adam optimization algorithm for deep learning. `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`.

Bulò, S. R. and Pelillo, M. (2009). A game-theoretic approach to hypergraph clustering. *Advances in Neural Information Processing Systems*.

Camerer, C. F., Dreber, A., Forsell, E., Ho, T., Huber, J., Johannesson, M., and Wu, H. (2016). Evaluating replicability of laboratory experiments in economics. *Science*.

Dhamal, S., Bhat, S., Anoop, K. R., and Embar, V. (2011). Pattern clustering using cooperative game theory. Centenary Conference. Electrical Engineering, Indian Institute of Science, Bangalore.

Diederik P. Kingma, J. L. B. (2015). Adam: A method for stochastic optimization. *ICLR 2015*.

Distill (2016). How to use t-sne effectively. `https://distill.pub/2016/misread-tsne/`.

Florczak, A., Jabłonowski, J., and Kupc, M. (2015). In pursuit for patterns of economic behaviours using cluster analysis and correspondence analysis. *Narodowy Bank Polski*.

Friedman, J. H., Tibshirani, R., and Hastie, T. (2009). *The elements of statistical learning*. Springer, 2 edition.

Gelman, A. and Loken, E. (2013). The garden of forking paths: Why multiple comparisons can be a problem, even when there is no "fishing expedition" or "p-hacking" and the research hypothesis was posited ahead of time.

Hollenstein, H. (2000). Innovation modes in the swiss service sector: a cluster analysis based on firm-level data. 3rd Workshop of the Focus Group on Innovative Firms and Networks. OECD.

Keras Documentation. Losses. `https://keras.io/losses/`.

Keras Documentation. Optimizers. `https://keras.io/optimizers/`.

Lupták, M. (2015). 4 reasons why economists make great data scientists (and why no one tells them). `https://medium.com/@metjush/`.

MacKay, D. (2003). *Information theory, inference and learning algorithms*. Cambridge University Press.

Nash, J. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1).

Nash, J. (1951). Non-cooperative games. *The Annals of Mathematics*, 54(2).

Pandre, A. Cluster analysis: see it 1st. `https://apandre.wordpress.com/visible-data/cluster-analysis/`.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2.

Pelillo, M. (2009). What is a cluster? perspectives from game theory. NIPS Workshop on Clustering: Science of Art.

Powell, V. (2014). Principal component analysis, explained visually. `http://setosa.io/ev/principal-component-analysis/`.

R-Bloggers (2017). Playing with dimensions: from clustering, pca, t-sne... to carl sagan! `https://www.r-bloggers.com/playing-with-dimensions-from-clustering-pca-t-sne-to-carl-sagan/`.

Řezanková, H. (2014). Cluster analysis of economic data. *Statistika*, 94.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 20.

Scikit-Learn. Scikit-learn example of silhouette analysis. `http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html`.

Setyaningsih, S. (2012). Using cluster analysis study to examine the successful performance entrepreneur in indonesia. *Procedia Economics and Finance*, 4.

Shlens, J. (2014). A tutorial on principal component analysis. *Google Research*.

Stanford. Ufldl tutorial - autoencoders. `http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders`.

Steinhaus, H. (1956). Sur la division des corp materiels en parties. *Bulletin Polish Acad. Sci.*

va der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9.

Vasilakos, A. V., Kannan, R., Hossain, E., and Kintis, H. (2010). Special issue on game theory. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(3).

von Luxburg, U. (2010). Clustering stability: an overview. *Foundations and Trends in Machine Learning*, 2.

8 Appendix and Acronyms

Hyperparameters

All autoencoders were initially run for 500 epochs using the Adam optimizer, and the loss was measured using the MSE (see Section 2.4). We used a batch size of 200 and learning rate of 0.001. Once the model was chosen, the autoencoders were run again on the larger sample data set using maximum 5,000,000 iterations. We used a match size of 128 with a learning rate of 0.0001.

All t-SNE graphs were run with perplexity 30 for 500 iterations.

Table 8.1: Static Silhouette Scores, Active Gamers

Model	***k*=10**	***k*=16**	**Cluster Ratio**
K-Means (all features)	0.11557	0.14206	4608
K-Means (selected features)	0.24750	0.25193	5366
PCA	0.21851	0.21791	3733
Autoencoder	0.29715	0.24304	39
DAE	0.26518	0.23617	6
DAE (over-complete)	0.19595	0.17198	5

Table 8.2: Time-Series Silhouette Scores, Active Players

Model	***k*=10**	***k*=16**	**Cluster Ratio**
K-Means (all features)	0.02503	-0.01870	16
K-Means (daily average)	0.16037	0.15614	5
PCA	0.06549	0.05843	17
Autoencoder	0.09828	0.07773	6
DAE	0.15269	0.11668	14
DAE (over-complete)	0.10568	0.08400	4

Table 8.3: Static Silhouette Scores, All Players

Model	***k*=10**	***k*=16**	**Cluster Ratio**
K-Means (all features)	0.10507	0.15838	425
K-Means (selected features)	0.15582	0.33049	480
PCA	0.23318	0.24147	369
Autoencoder	0.30301	0.29207	23
DAE	0.26961	0.28746	14
DAE (over-complete)	0.25658	0.25011	3

Table 8.4: Time-Series Data, All Players

Model	***k*=10**	***k*=16**	**Cluster Ratio**
K-Means (all features)	0.27042	0.31185	78
K-Means (daily average)	0.37974	0.37303	15
PCA	0.54085	0.60244	67
Autoencoder	0.25420	0.34003	22
DAE	0.17012	0.21870	19
DAE (over-complete)	0.13812	0.13297	9

Table 8.5: Deep Autoencoder with Over-Complete Layer, Silhouette Scores

Model	**Static**	**Activity**
$k = 2$	0.65407	0.14695
$k = 3$	0.29554	0.13182
$k = 4$	0.21385	0.11938
$k = 5$	0.23253	0.11508
$k = 6$	0.22461	0.09271
$k = 7$	0.17990	0.09306
$k = 8$	0.17273	0.08931
$k = 9$	0.17007	0.08009
$k = 10$	0.17064	0.08046

Acronyms

AE Autoencoder. 13

DAE Deep Autoencoder. 4, 15, 39, 41, 49, 50

KL Kullback-Leibler. 16

KPI Key Performance Indicators. 18, 30, 41, 43, 44

MSE mean square error. 15, 49

NE Nash equilibrium. 21, 23, 25, 26, 27, 45

PCA Principal Component Analysis. 3, 12, 13, 16, 32, 34, 36, 49, 50

t-SNE t-Distributed Stochastic Neighbor Embedding. 3, 16, 17, 37, 39, 41, 49