University of Waterloo
Department of Management Sciences

Understanding and Predicting Team Success in DOTA2

Prepared For: Dr. Lukasz Golab

November 28, 2018

Table of Contents

1.0 Abstract

DOTA2 is a team oriented multiplayer online battle arena (MOBA) game with 6.5 million monthly players published by Valve Corporation. In each match, two teams of five players select and control 'heros' which have unique attributes and play one of nine possible roles on a team. These teams compete against each other, often through team fights, in an attempt to occupy the opposing team's area of the map while simultaneously defending their own.

Optimal behaviour in MOBA games is not only difficult to for novice players to learn - often requiring many hours of gameplay and frustration. In this report, game log data were collected and mined to determine guidelines for team composition and strategy. Apriori association rule mining is used to identify essential hero roles that should be present on each team. Clustering is used to identify types of teams that players can form. Lastly, linear regression is used to identify factors that that increase the chance of winning a combat engagement called a team fight, which result in a the temporary death of 3 or more players.

Apriori association rule mining not only identified that teams rely on the carry role to achieve success, but also identified three essential champion roles that should be present - support, nuker, and carry. From clustering, four types of teams were identified - a defensive composition, a balanced composition, and two aggressive compositions; one of which is stronger early in the game, and one of which is stronger later in the game. Lastly, linear regression identified that the most important factor by far in predicting the success of a team fight within a game is the number of team members participating relative to the other team. This highlights the importance of teamwork to success in-game. Also it is notable that the coefficients for static statistics (trueskill, matches played, and matches won) were consistently more impactful than in-game metrics, which suggests that even if a player is behind in the game, if they can play skillfully, that team will have a good chance of winning a teamfight.

2.0 Introduction

2.1 Purpose

The purpose of this project is to discover detailed insights to inexperienced beginner players to the implicit strategies of the game. New players can use the insight developed on the team composition to create more successful teams and have a more enjoyable gameplay experience. This in turn can drive up early player retention and engagement in the game because users will have the desire to return to a game they are successful in. Another objective of this project is for the developers of Dota 2, the increase in these metrics can increase the number of regular players they have, improve the user experience, and drive up other essential metrics such as revenue.

2.2 How DOTA2 works

DOTA2 is a multiplayer online battle arena video game in which two teams of five players compete to destroy a large structure defended by the opposing team, while defending their own. The concept of the game can be likened to a game of basketball, where there are five players on each team trying to score as much as they can on the opposing team's net, while protecting their own.

Each player in a match controls one of the game's 113 playable characters, known as "heroes". Each hero has their own design, strengths, and weaknesses. Heroes can be likened to the positions of players on a basketball team. Heroes are categorized into 9 different roles: Carry, Disabler, Lane Support, Initiator, Jungler, Support, Durable, Nuker, Pusher, and Escape. Players select their hero before the game, and can work with their teammates to determine strategies and hero matchups.

In a match, the two teams are known as the Radiant and the Dire - this is simply to distinguish between the two teams. The teams occupy bases in opposite corners of the map, and work to get to the other side to destroy the other team's structure. In the space between the bases are defensive towers that attack any opposing unit that gets within line of its sight. There is also a small group of "creeps", weak computer-controlled creatures that travel in the general area to attack anything they see, including heroes. Creeps are generated from barracks that belong to each team and attack the opposing team.

2.3 Main Hypotheses

Linear Regression

With all the factors that come into play: player's prior experience (measured by number of games played), games won, trueskill rating, gold, current ingame experience, and number of last hits - gold should matter more than all any other factors in the game to determine a team's success. The reasoning behind this is that heroes that are categorized as "carries" are essential to the team's ultimate victory in the end. However, to make these carries powerful is to equip them with items that can only be bought with gold.

K Means Clustering

K Means should result in three different team types (3 clusters should be most optimal) such that one type is offensive, one type is defensive, and the last type is a balanced team. A balanced team is a team that has a good mix of defensive heroes and offensive heroes.

Association Rule Mining

The are two hypotheses associated that Association Rule Mining method will be used to help accept or reject. The first hypothesis is that teams revolve around the “carry” hero. Specifically, non carry roles rely on the carry role to bring the team to success. The second hypothesis is that there are at least three hero roles that are crucial to each team.

3.0 Related Work

Because of Dota 2's popularity and the availability of the game's data, there exists several works of scholarly analysis and projects on the game. Previous scholarly work on Dota 2 covers a range of topics - match outcome prediction, analysis on AI bots in the game, and neural networks and genetic algorithms. The topic selected for this project is determining the success of teams in Dota 2 based on the match composition.

The first article mentioning Dota 2 was qualitative and analyzed correlation of leadership styles of players with roles in the game they choose to play [2]. The study used data collected from close-ended surveys given to users in online and written formats at the end of games played. The collected data were quantitatively analyzed using statistical analysis tools with methods of factor analysis and multinomial logistic regression analysis. The purpose of this research is to explore whether or not behaviors in games are correlated with leadership behaviors in real life. This study is different than the work completed in this project - this study analyzes input from users given through surveys, and ties in personal features of users to the results set.

A second article on Dota 2 analyzed cooperation within teams, national compositions of players, role distribution of heroes and other stats [3]. The data was analyzed from Dota 2 web forums and game log data. This paper analyzes factors related to the role that players can take within a game, the experiences of the players, and the friendship ties within a team. This study is different than the work completed in this project - this study looks at various components that affect team composition.

A third article analyzes topological patterns of Dota 2 teams based on area, inertia, diameter, distance and other features derived from their positions and movements of the players around the map to identify which of them are related with winning or losing the game [4]. The data used in this analysis comes from approximately 1200 games played. This study is different than the work completed in this project because it looks at a completely different problem to predict.

A fourth article uses Neural Networks and Genetic Algorithms to analyze and optimize patterns of heroes' movements on the map in Dota 2 [5]. The data was analyzed from replay files (files that allow another player's client to replay the match) shared in the Dota 2 playing community and online sites. The purpose of this study is to look at skill-based differences of teams based on their location in the game. This study is different than the work completed in this project because it looks at a completely different problem to predict.

A fifth article applied applied graph theory to identify patterns in combat and hence analyze teams ' tactics and predict fight results with them with 80% accuracy on test data [6]. The dataset used team composition data to predict the winning team from the drafting stage of the game, calculating optimal jungling paths, predict the result of teamfights, recommendation engine for the draft, and detection of in-game roles with emphasis on win prediction from team composition data. This study is different than the work completed in this project because it looks at predicting a winning team via combat patterns, whereas the work completed in this project looks at predicting the success of a team by team composition.

The sixth article builds up on the approach of the fifth article and takes into account range of attack and spells for each hero to make a better algorithm for encounter detection and team performance

evaluation [7]. The study used match data obtained by parsing replay files from DOTA, obtained from online replay repositories. Replay files are generated by the game client for the purpose of enabling match record sharing and analysis. The files include all of the information needed for another player's game client to replay the match. Matches with incomplete data or where players left the game early were not included. This study is different than the work completed in this project because it looks at predicting a winning team via combat patterns, whereas the work completed in this project looks at predicting the success of a team by team composition.

The seventh and eighth articles used Logistic Regression and Random Forest to detect hero roles for hero ids for both public and professional games and for hero positions [8] [9]. This paper investigates the applicability of supervised machine learning to classify player behavior in terms of specific and commonly accepted but not formally well-defined roles within a team of players of the game Dota 2. The study used a dataset of 708 labelled players and applied logistic regression on it. This study is different than the work completed in this project because it focuses on detecting abstract hero roles and how players behave in these roles.

A ninth article used a Naive Bayes classifier to analyze team lineups and predict the outcome according to the lineups, while also giving an improved Naive Bayes classifier [10]. Using the Dota 2 data set published in the UCI Machine Learning Repository, the study tests Naive Bayes classifier prediction of respective winning probability of both sides in the game. This study is different than the work completed in this project because the study uses team composition and Naive Bayes to predict winnings of teams, whereas this project combines team composition by role type, linear regression, and kill-death ratio calculations to predict if a team will be successful in Dota 2.

A tenth article used fuzzy controller based artificial intelligence (AI) for dynamic difficulty adjustment in Dota 2 [11]. Even though this game is more focused on its multiplayer players versus players mode, it's bot match mode is still a viable feature provided by Valve for beginner players to start learning Dota 2 and even for seasoned players it is still used to testing builds or sharpening their skills. However static AI implemented in this game are often mismatched between the player and the AI difficulty because player themselves don't know how far their limit is. This research offer enrichment in gaming experience for players with implementation of Dynamic Difficulty Adjustment (DDA) by applying right difficulty based on player in-game records. This study is different than the work completed in this project because it looks at the AI controlled portion of the game and how it should be adjusted based on previous games to make the user experience better for users, whereas the work completed in this project looks at how successful teams can be based on their team composition.

4.0 Data

4.1 Kaggle Data Collection

A dataset from Kaggle, Dota 2 Matches, was used in this project. It contains a diverse collection of Dota 2 data on players, matches, locations, as well as some in-game details. The Kaggle dataset originates from opendota, an open source site where Dota 2 data is dumped. The general dataset contains data for 50000 uniques matches. However, there are some sets that describe similar characteristics for 100000 unique matches. All of the downloaded data is stored in the form of CSVs. There is also an example JSON file that contains approximately 100 samples of the data in its original form, and a large JSON file from which the data was derived.

4.1.1 General Exploratory Data Analysis

Exploratory data analysis revealed that the datasets are arbitrarily split into testing and training data. It was found that datasets were indexed by ascending match ID. Any data with "test" in its filename start with match IDs that training datasets had ended with. Consequently, not all of the datasets can be used with each other. When data mining, the testing datasets should only be used with each other, and the training datasets should only be used with each other.

Another insight discovered is that when players opt to play anonymously, their account ID is set to 0. This means that many players have had this account ID, currently have this account ID, and will have this account ID in the future.

4.2 In-game Data Collection

The DOTA2 user interface was also used to add additional data relating to heroes to the Kaggle dataset. For each hero, the team dug into the DOTA2 UI (as seen in the following figure), and looked at the hero categories section. For each hero, the most dominant trait was selected as its role. If there was a tie between two attributes, the trait was selected based on the precedence in the hero description section. The DOTA2 UI was the only place to get this information.

4.3 Dataset Details

The following tables of the dataset will be used for analysis and mining. Their relevant columns and column value semantics are discussed in detail below.

4.3.1 hero_names.csv

This dataset will be used to determine the role or name of a hero based on hero_id.

Interesting insight from the dataset itself is that the carry role is the most common of the nine possible roles. 22% of all heroes, or thirty heroes have a primary role of carry. Second most common is the support role, which is the primary role of 20 heroes. The least common role is that of jungler, which is the primary role of only three heroes.

A bar chart demonstrating the percentage of heroes with each role as their primary is shown below in Figure 1.

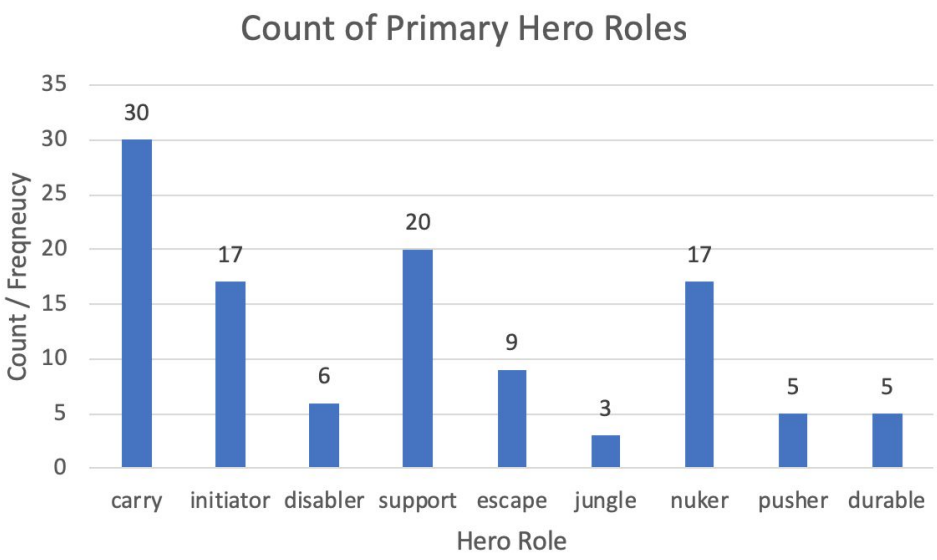

Figure 1: Bar graph showing count of primary hero roles

A sample of the data is show below, along with Table 1, which describes the data in detail.

	name	hero_id	localized_name	role
0	npc_dota_hero_antimage	1	Anti-Mage	carry
1	npc_dota_hero_axe	2	Axe	initiator
2	npc_dota_hero_bane	3	Bane	disabler
3	npc_dota_hero_bloodseeker	4	Bloodseeker	carry
4	npc_dota_hero_crystal_maiden	5	Crystal Maiden	support

Figure 2: Sample data from hero_names.csv

Table 1: hero_names.csv dataset details

# of Rows	133	
Row Semantic	Each row represents a distinct hero, along with their names and primary role	
# of Columns	4	
Column Details		
hero_id	Primary Key Type: Integer	Lists the distinct IDs of the heroes
name	Type: string	Lists the variable name of each hero corresponding to the hero ID
localized_name	Type: string	List the name of each hero in the local language
Role	Type: string	Lists the roles of each hero

4.3.2 players.csv

This dataset contains summary statistics of a player's status at the very end of a match.

match_id	account_id	hero_id	player_slot	gold	gold_spent	gold_per_	xp_per_m	kills	deaths	assists	denies	last_hits	stuns	hero_dam	hero_hea	tower_da	item_0	item_1	item_2
0	0	86	0	3261	10960	347	362	9	3	18	1	30	76.7356	8690	218	143	180	37	73
0	1	51	1	2954	17760	494	659	13	3	18	9	109	87.4164	23747	0	423	46	63	119
0	0	83	2	110	12195	350	385	0	4	15	1	58	None	4217	1595	399	48	60	59
0	2	11	3	1179	22505	599	605	8	4	19	6	271	None	14832	2714	6055	63	147	154
0	3	67	4	3307	23825	613	762	20	3	17	13	245	None	33740	243	1833	114	92	147

Figure 3: Sample data from players.csv

Table 2: players.csv dataset details

# of Rows	500000	
Row Semantic	Each row is distinct combination of match ID, account ID, and hero ID, providing information on a player for that match	
# of Columns	65	
Relevant Columns There are four relevant columns in this table, out of the total of 65 columns. The rest of the columns look at summary statistics of a player's status at the very end of a match. Since none of the analyses in this report evaluate end of game performance, the other columns are not useful for the analysis of this project.		
Match_id	Primary Key Type: Integer	List of match IDs
Account_id	Primary Key Type: Integer	List of account IDs
Hero_id	Type: Integer	List of hero ID corresponding to the primary key of the hero_names tables.
Player_slot	Type: Integer	The player slot that a champion or account is assigned to for a match is used only to determine which team they are on. Each match of two teams has a total of 10 player slots. Each team consists of five player slots. **Semantic:** ● player slots 0 to 4 represent the five players the radiant team (rad=1) that starts at on the bottom half of the map. ● Player slots 128-132 represent the five players on the "dire team" (rad=0) that starts on the top half of the map.

4.3.3 player_ratings.csv

This table contains a number of statistics that describe a player's overall skill. This information will be used to help predict the success of a team in a team fight.

Of particular note, the mean trueskill is 25.112, the 75% trueskill is 27.24, while the max trueskill is 48.83. This suggests that a small population players are far more skilled than the average. Furthermore the mean trueskill standard deviation is 7.27, the 25 percentile trueskill standard deviation is 6.95, while the minimum true skill standard deviation is 1.128. This suggests that certain skilled players are much more consistent with their in game performance.

A sample of the data is shown below, along with Table Player Ratings, which describes the data in detail.

	account_id	total_wins	total_matches	trueskill_mu	trueskill_sigma
0	236579	14	24	27.868035	5.212361
1	-343	1	1	26.544163	8.065475
2	-1217	1	1	26.521103	8.114989
3	-1227	1	1	27.248025	8.092217
4	-1284	0	1	22.931016	8.092224

Figure 4: Sample data from player_ratings.csv

Table 3: player_ratings.csv dataset details

# of Rows	834226	
Row Semantic	Each row represents a distinct account along their statistics.	
# of Columns	5	
Column Details		
account_id	Type: Integer	Primary Key
total_wins	Type: Integer	Total number of matches that the account has participated in where their team won
total_matches	Type: string	Total number of matches that the account has participated in
trueskill_mu	Type: float	This describes the average skill level demonstrated by a player in all their games. A higher number means that the player is more skilled.
trueskill_sigma	Type: float	This describes the variability of a player's skill level based on their previous performance in matches. A lower number means that a player is more consistent in their abilities.

Statistics:

	account_id	total_wins	total_matches	trueskill_mu	trueskill_sigma
count	8.342260e+05	8.342260e+05	8.342260e+05	834226.000000	834226.000000
mean	-9.225868e+07	5.479852e+00	1.095979e+01	25.112577	7.270275
std	8.103222e+07	1.760984e+03	3.629559e+03	3.231603	1.128826
min	-2.991940e+08	0.000000e+00	1.000000e+00	4.993478	1.404098
25%	-1.499249e+08	0.000000e+00	1.000000e+00	22.906655	6.957458
50%	-9.585022e+07	1.000000e+00	2.000000e+00	25.018193	7.732504
75%	4.883475e+04	3.000000e+00	6.000000e+00	27.240350	8.058739
max	3.305130e+05	1.608398e+06	3.315071e+06	48.825892	8.333689

Figure 5: Descriptive statistics of variables from player_ratings.csv

4.3.4 player_time.csv

This table reports the progress of player in a match. For each match, the accumulated gold, experience, and number of last hits is reported at 60 second intervals.

This table will be used in the linear regression section of this report in order to determine the overall progress of teams and their members at the start of a time fight.

A sample of the data is shown below, along with Table Player Time which describes the data in detail.

	match_id	times	gold_t_0	lh_t_0	xp_t_0	gold_t_1	lh_t_1	xp_t_1	gold_t_2	lh_t_2	...	xp_t_129	gold_t_130	lh_t_130	xp_t_130	gold_t_131	lh_t_131	xp_t
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	60	409	0	63	142	1	186	168	0	...	62	345	6	351	100	0	
2	0	120	546	0	283	622	4	645	330	0	...	543	684	12	805	200	0	
3	0	180	683	1	314	927	9	1202	430	0	...	842	958	16	1135	300	0	
4	0	240	956	1	485	1264	11	1583	530	0	...	1048	1500	26	1842	400	0	

Figure 6: Sample data from player_time.csv

Table 4: player_time.csv dataset details

# of Rows	834226	
Row Semantic	Each row represents a distinct account along their statistics.	
# of Columns	32	
Primary Key	(match_id, times)	
Column Details Note: For the columns below, i is a set of players where each i represents an individual player. i can have the values of 0,1,2,3,4,128,129,130,131,132. i = [0,4] represents players on team 0 i = [128,132] represents players on team 1. This notation is used so to avoid avoid describing the same three columns ten times, once for each player.		
match_id	Type: Integer	Partial Key

		This indicates the match that the data in the rest of the columns correspond to.
times	Type: Integer	Partial Key This indicates the time in seconds that the rest of the data is being reported at. For example, if the value is 60, then it means that the exp, gold, and lh column values are reported after 60 seconds of the match have elapsed.
Gold_t_**i,** where i ∈ [0,4] and [128,132]	Type: string	Gold_t_i represents the gold that a player 'i' has accumulated at time 't' in a specific match. **Semantic**: A higher value is better since gold can be used to purchase items that increase player strength for the match.
Lh_t_**i** where i ∈ [0,4] and [128,132]	Type: float	The number of killing hits that player 'i' has accumulated at time 't' in a specific match. **Semantic**: A higher value is better killing hits reward more gold and experience that increase player strength for the match.
Xp_t_i where i ∈ [0,4] and [128,132]	Type: float	The experience that player 'i' has accumulated at time 't' in a specific match. **Semantic**: A higher value is better since higher experience means higher player strength.

4.3.5 teamfights.csv

Team fights are defined as engagements where three or more players die as a result. This table records the team fights that occur in a match, when they start, end, when the last player died, and the number of deaths resulted.

This table will be used in the linear regression section of this report in order to determine the overall progress of teams and their members at the start of a time fight.

A sample of the data is shown below, along with Table X which describes the data in detail.

	match_id	start	end	last_death	deaths
0	0	220	252	237	3
1	0	429	475	460	3
2	0	900	936	921	3
3	0	1284	1328	1313	3
4	0	1614	1666	1651	5
5	0	1672	1709	1694	3

Figure 7: Sample data from teamfights.csv

Table 5: teamfights.csv dataset details

# of Rows	539047	
Row Semantic	Each row represents a distinct team fight in a match along key statistics.	
# of Columns	5	
Primary Key	(match_id, start, end)	
Column Details		
match_id	Type: Integer	This indicates the match that the data in the rest of the columns correspond to.
start	Type: Integer	This indicates the time in seconds elapsed since the start of a match that a team fight starts at
end	Type: Integer	This indicates the time in seconds elapsed since the start of a match that a team fight ends at
last_death	Type: Integer	This indicates the time in seconds elapsed since the start of a match where the last death of the team fight occurs
deaths	Type: Integer	This indicates the total number of players death that resulted from the team fight

4.3.6 teamfights_players.csv

This table will be used in the linear regression to determine the deaths on each team, and gold/experience gains that occur for each team as a result of a team fight.

A sample of the data is shown below, along with Table X which describes the data in detail.

	match_id	player_slot	buybacks	damage	deaths	gold_delta	xp_end	xp_start
0	0	0	0	105	0	173	536	314
1	0	1	0	566	1	0	1583	1418
2	0	2	0	0	0	0	391	391
3	0	3	0	0	0	123	1775	1419
4	0	4	0	444	0	336	1267	983

Figure 8: Sample data from teamfights_players.csv

Table 6: teamfight_players.csv dataset details

# of Rows	539047
Row Semantic	Each row is a unique combination of match ID and player slot, representing their impact in team fights in the game
# of Columns	8
Primary Key	index

Relevant Columns	Relevant Columns Info	
Match_id	Type: Integer	List of match IDs
Player_slot	Type: Integer	List of account IDs
buybacks	Type: Integer	This represents the # of times a player died and paid gold to revive themselves.
damage	Type: Integer	This represents the amount of damage that a player dealt to the other team during a team fight.
deaths	Type: Integer	This represents the number of times a player_slot died in a team fight
gold_delta	Type: Integer	This represents the difference in gold between the start and the end of the match. A player death often results in a negative gold_delta value since gold is lost upon death.
xp_end	Type: Integer	This is the experience that a player has at the end of a team fight. A gain of 0 exp compared to xp_start means that the player was not involved in teh team fight.
xp_start	Type: Integer	This is the experience that a player has at the start of a team fight.

4.3.7 test_players.csv

This table will be transformed to determine the hero_ids and the hero roles that are present on each time in a match. This information will then be mined through association rule mining and clustering to draw insights on team composition.

A sample of the data is shown below, along with Table X which describes the data in detail.

	match_id	account_id	hero_id	player_slot
0	50000	117784	96	0
1	50000	158361	84	1
2	50000	158362	46	2
3	50000	137970	85	3
4	50000	1090	39	4

Figure 9: Sample data from test_players.csv

Table 7: test_players.csv dataset details

# of Rows	100000
Row Semantic	Each row is distinct combination of match ID, account ID, and hero ID, providing information on a player for that match

# of Columns	4	
Primary Key	(match_id, account_id, player_slot)	
Relevant Columns		
Match_id	Type: Integer	List of match IDs.
Account_id	Type: Integer	List of account IDs present in a match.
Hero_id	Type: Integer	This represents hero selected by the player in a specific match. This value corresponds to the primary key of the hero_names tables.
Player_slot	Type: Integer	The team number of the player **Semantic:** ● player slots 0 to 4 represent the five players the radiant team (rad=1) that starts at on the bottom half of the map. ● Player slots 128-132 represent the five players on the “dire team” (rad=0) that starts on the top half of the map.

The data was transformed in different ways for each method discussed in this report (Linear regression, Association, and Clustering). How the data was transformed is discussed in the respective sections of the report.

5.0 Results

5.1 Predictive Analysis - Linear Regression

Pre-Analysis

Linear regression is selected to predict the death ratio of a team fight instead of logistic regression because oftentimes a winner is ambiguous if both teams have the same number of kills. Additionally, the kill-death ratio is a more insightful piece of information as it describes how successful the fight will be rather than just the probability of success. At a higher level perspective, linear regression is used to predict how successful a team will be based on the multiple factors (gold, experience, etc.) each players have.

Data Preprocessing

Data was cleaned and ready for preprocessing by removing fights where a player had more than three deaths, as anything above three deaths is unlikely to happen over the course of a single fight. Then the win and match counts for anonymous players with account_id = 0 were replaced with the average value for all registered players. Unfortunately, because of complicated transformations with the data in order to get it into a usable format, the computational power to run the regression on all the data was not available.

The input data had to be normalized for the sake of consistency throughout a match. Values such as gold and experience tend to increase with time, and when trying to run a regression on values that vary with time an extra layer of noise is added to the fitting and testing processes. To avoid this issue and to make the coefficients easier to analyse after the regression all the variables were normalised to values between 1 and -1, by dividing the difference between a given variable for both teams by the total value across the teams. This gives a negative value if the opposing team has an advantage, or positive if the radiant team is at an advantage.

Motivation & Intention

By applying linear regression, the intention is to determine what factors have the biggest impact on the success of a team fight. The factors include how much gold is accumulated during a match and how much experience each hero has. Team fights play a huge role in the enjoyment and outcome of a game. By predicting pre-game statistics, players will be well informed on how prepared they will be relative to the current game play when engaging in enemy combat.

Hypothesis

With all the factors that come into play: player's prior experience (measured by number of games played), games won, trueskill rating, gold, current ingame experience, and number of last hits. Gold should matter more than all any other factors in the game to determine a team's success. The reasoning behind this is that heroes that are categorized as "carries" are essential to the team's ultimate victory in the end. However, to make these carries powerful is to equip them with items that can only be bought with gold.

Data Interpretation

The following abbreviations are defined as follows for the sample data table below:

- KDD = Kill Death Difference
- Gold = ingame gold that a player accumulates
- Xp = experience
- Lh = last hit, the number of killings a player performed as the last one to hit an enemy before their death
- Trueskill = an arbitrary metric that was calculated by a third party organization
- Count = number of players active in a fight

In the table shown below, KDD (Kill Death Difference) is the class variable whereas all the rest are the explanatory variables to predict the KDD. Each row for the sample data table represents a team fight for a specific team. Taking one's perspective of a team, having a positive value in a cell means the opponents team has more of a specific factor.

For example in row 0 of the sample data table below, the cell underneath gold is -0.10224 meaning that the other team has more gold. The values within each cell are normalized relative to one another and can only take on values between -1 to 1. They are considered to be deltas between the two teams. Negative values mean that specific factor for that fight is favoured to the opponents.

However, what matters most is the class variable, the KDD. Row 0's KDD is potentially an outlier/exception because its value is positive whereas all the explanatory variables are negative. What this means is that although every explanatory factor favours the opponents (for example, the opponent has more gold), the team that is against the opponents have won. The values for wins means the opponents has more historical total wins.

The following formulas are to describe how the value of certain non-trivial factors are computed:

- count = (team 1 deaths- team 0 deaths) / total deaths
- KDD = (radiant kills - radiant deaths)/sum of deaths
- count: # active team members in fight
- gold = (radiant gold - dire gold) / total gold of both teams

At the beginning of each fight, each team chooses what their theme is. Radiant and Dire are the two themes that the teams can choose from.

	KDD	gold	xp	lh	wins	matches	trueskill	count
0	0.333333	-0.10224	-0.01138	-0.26984	-0.45103	-0.35946	-0.09031	-0.14286
1	1	-0.06674	0.006387	-0.16757	-0.36254	-0.28142	-0.07345	-0.11111
2	1	-0.00584	0.024111	-0.09455	-0.29322	-0.19392	0.021952	0
3	1	0.0616	0.107129	-0.03831	-0.36254	-0.28142	-0.07345	-0.11111
4	0.2	0.070763	0.122887	0.004274	-0.36254	-0.28142	-0.07345	-0.11111
5	-1	0.175214	0.129874	0.026506	-0.06463	-0.02191	-0.04458	-0.2
6	0.333333	0.064958	0.109541	0.018905	-0.36254	-0.28142	-0.07345	-0.11111
7	0.2	0.062988	0.098301	0.024248	-0.36254	-0.28142	-0.07345	-0.11111
8	0.2	0.099898	0.149258	0.125755	0.008075	0.056626	0.05314	0
9	0.5	0.07333	0.131814	0.051095	-0.32461	-0.23616	0.02947	0
10	0.5	0.088404	0.114646	0.030912	-0.36254	-0.28142	-0.07345	-0.11111
11	0.333333	0.114114	0.119197	0.028257	-0.45103	-0.35946	-0.09031	-0.14286

Figure 10: sample of training data used for linear regression

From running the linear regression, a set of coefficients were produced representing the relationship between each variable that have been identified above, and the normalised kill death difference. Some of these values were surprising, but others turned out as expected.

One of the smallest coefficients was xp (experience points). This was surprising as experience determines how powerful a hero is, as it levels the hero up to give better attack and defense abilities, as well as a new set of skills to use in battle

Another surprising result from this regression is the weakness of wins and matches in the model. These factors have very low coefficients, but they should correspond to hours logged in game, and hence a players skill level. This contrasts with the trueskill mu however, as that metric was fairly effective at predicting the fight outcome.

This is an interesting outcome, but one factor that may be affecting these values is the anonymous players. As discussed above, since about a third of the players are anonymous, it would be impossible to remove them without skewing the data in a serious way for all the fights, so they were assigned the average number of wins and matches. This can make matches where players are rated much higher than they actually are in terms of matches and wins, which probably had a big impact on the model in terms of wins, matches, and trueskill.

Another notable result is that the coefficients for gold, xp, and last hits are smaller than those for count and trueskill. One explanation for this difference is that the actual experience of a player has a larger impact on their success in a team fight than the power of their hero. This suggests that even if a player has fallen behind the other players in a match, they can still do well in a team fight if they are skilled at the game, or if they communicate well with their team to get a numbers advantage in the next fight.

Finally, the most important predictive variable by far is the count of players in the fight. This makes intuitive sense, as having fewer teammates to help in a fight will be a much greater impact on the fight than the skill of the team members in most cases.

Evaluation

Using K-fold evaluation with k = 10, the average score for the model is 0.194. This means that the model does not do a very good job of predicting the kill death difference for each teamfight. It does do better than simply guessing the average kill death difference by a significant amount, but it is there is variance in the data that is not accounted for or explained. There are a number of confounding variables that are difficult to account for, such as the health of each hero going into the fight and how well the player is doing on that particular day. Since a team fight is such a volatile situation, a good prediction is very difficult to make given the data.

	gold	xp	lh	wins	matches	trueskill	count
0	0.031688	-0.023092	-0.01833	-0.092977	0.090788	0.205173	0.926242

```
#Model Evaluation:
score = []
kFold = KFold(10, True, 1)
for test, train in kFold.split(X1['wins']):
    model = lm.fit(X1.iloc[train],Y1.iloc[train])
    score.append(lm.score(X1.iloc[test], Y1.iloc[test]))
```

```
sum(score)/len(score)
```

```
0.1940227198076577
```

Figure 11: iPython output of linear regression variables and k-fold r^2 score

5.2 Association - Apriori

Association rule mining is applied to find the characteristics of heroes that go together. Specifically, the goal is to find hero “roles” that go together.

5.2.1 Motivation

In a round of DOTA2, there are two teams that each have five players. On each team, a player selects a “hero” or a character that they intend to play as for the upcoming round. Although each champion has its own unique abilities, they are generalized into one of nine roles - carry, support, initiator, durable, jungler, nuker, escape, and disabler.

This is an important problem for several reasons. Although DOTA2 is played as a team strategy game, the onboarding and tutorials do not have any information towards team composition or strategy, leaving beginner players confused. Furthermore, since there are only five player slots on each team and nine unique champion roles, selecting a set of five heroes that can fulfill the correct roles affects match outcome even before the game starts.

Thus, the motivation of Association Rule Mining is to understand whether the team composition of winning and losing teams differs by leveraging association rule mining to determine frequent champion roles that are played together on winning and losing teams.

5.2.2 Hypothesis

The are two hypotheses associated that Association Rule Mining method will be used to help accept or reject. The first hypothesis is that teams revolve around the “carry” hero. Specifically, non carry roles rely on the carry role to bring the team to success. The second hypothesis is that there are at least three hero roles that are crucial to each team.

5.2.3 Pre-processing and Data Interpretation

In order to obtain a dataset where itemsets contains the hero roles present a team, three datasets are processed and transformed. Firstly, the heroes present on each team must be determined by transforming test_player.csv. Secondly, the winning team of each match is added by appending the

'radiant_win' column from outcomes.csv. Lastly, compare the "hero_id"s on each team against the 'role' column hero_names.csv in order to produce a itemset of roles present in each team.

[['durable', 'support', 'carry', 'durable', 'nuker'], ['escape', 'disabler', 'escape', 'nuker', 'escape'], ['carry', 'escape', 'support', 'initiator', 'escape'], ['jungler', 'pusher', 'carry', 'carry', 'disabler'], ['nuker', 'initiator', 'durable', 'durable', 'disabler']**]**

The list of hero roles present on each team and the game outcome is then transformed again using TransactionEncoder from the mlxtend.preprocessing library. The data is now ready for Apriori association rule mining. 5 sample rows of data are shown in the tables below.

	carry	disabler	durable	escape	initiator	jungler	nuker	pusher	support
0	True	False	True	False	False	False	True	False	True
1	False	True	False	True	False	False	True	False	False
2	True	False	False	True	True	False	False	False	True
3	True	True	False	False	False	True	False	True	False
4	False	True	True	False	True	False	True	False	False

Figure 12: sample of data processed by apriori algorithm

The class variable (which of the two teams wins) was not included due to inability to account for the level of competition. For example, if a team of very skilled player competes with a team that is not very skilled, regardless of the roles selected, the lesser skilled team will lose. Furthermore, if two very strong teams play, one of them will be a losing team regardless of team composition.

5.2.4 Results & Analysis from Association Rule Mining

Four iterations of association rule mining were performed with varying minimum support and confidence levels. A total of 16 association rules were obtained and carefully analyzed.

Key insights from iterations two and three include a reciprocal relationship between the carry role and support role, as well as the carry role and the nuker role. The relationship is stronger where the nuker or support role is the antecedent and the carry role is the consequent. Because of this, hypothesis #2 is failed to reject (accept). Teams appear to have three essential roles - the nuker, support, and carry.

Key insights from iteration four includes a set of eight rules where the eight non carry roles are the antecedent, and the carry role is the consequent. A low support level was selected because some hero roles are a lot less common than others; there are 30 carry heroes, but only 3 jungle heroes. The minimum confidence across the four eight rules is 77.2%. Because this strong relationship proves that all non-carry roles rely on the presence of a carry role for success, hypothesis #1 is failed to be rejected (accepted).

5.2.4.1 Iteration 1: Minimum Support: 60%, Minimum Confidence: 95%

In order to test the hypothesis that teams revolve around the carry role, a high support value of 60% and a confidence value of 95% is initialized. This means that any rules mined exist in the majority of teams, and that there is a very high confidence in the rule.

At a minimum support of 0.6 and minimum confidence of 0.95, zero association rules are returned. Analysis of the support for individual items, however, shows that the carry role has the highest support with presence in 87.4% of teams; this partially validates the hypothesis that teams revolve around carries.

	support	itemsets
0	0.874320	(carry)
5	0.707660	(support)
4	0.658295	(nuker)

Figure 13: itemset support rules generated

5.2.4.2 Iteration 2: Minimum Support: 60%, Minimum Confidence: 85%

For the second iteration, the minimum confidence level is reduced slightly to 85%. With these hyperparameters, one association rule was returned where the support role is the antecedent, and the consequent is the support role. This rule had a support of 0.615 and a confidence of 0.87.

The appearance of this singular association rule at the high confidence and support level comes as no surprise. Champions whose primary role is support exist solely to enable the rest of the team, and carry roles are present in 87.4% of all teams.

5.2.4.3 Iteration 3: Minimum Support: 50%, Minimum Confidence: 60%

For the third iteration, the minimum confidence level is further reduced to 60% while the minimum support level is reduced to 50%.. With these hyperparameters, four interesting association rule are determined.

	antecedents	consequents	support	confidence
2	(support)	(carry)	0.61502	0.869090
0	(nuker)	(carry)	0.55371	0.841127
3	(carry)	(support)	0.61502	0.703427
1	(carry)	(nuker)	0.55371	0.633304

Figure 14: all association rules generated by apriori algorithm upon iteration 4

The rules with the two highest confidence values (#2 and #0) demonstrate that support and nuker roles rely on the presence of a carry. The is expected because the 87.4% of team formations have at least one carry role.

More interestingly, the rules #3 and #1 where the carry role is the antecedent, and the consequents are the nuker and support roles highlight a reciprocal relationship between the support and carry roles, and between the nuker and carry roles. Due to the high levels of confidence and support for all four rules, the second hypotheses is fail to be rejected (accepted) - that teams have three essential roles.

Although the confidence for these rules is roughly 10-15% lower than the rules where the carry is the consequent, this aligns with the game mechanics where the carry can succeed without the nuker and support, but it is difficult for the nuker and especially the support to succeed without the presence of a carry role in a team.

5.2.4.4 Iteration 4: Minimum Support: 5%, Minimum Confidence: 75%

In an attempt to prove the first hypothesis (that teams revolve around the carry), the minimum support was reduced to 5%. This is because certain heroes roles are not as common in the data - for example, only three out of 113 heroes are junglers so the required support must be reduced in order to capture information relating to junglers and other less common hero roles.

The minimum confidence, however, is raised to 75% in an attempt to prove a strong relationship where the other eight hero roles rely on the carry.

	antecedents	consequents	support	confidence
7	(support)	(carry)	0.615020	0.869090
4	(jungler)	(carry)	0.052315	0.847618
3	(initiator)	(carry)	0.460495	0.844999
5	(nuker)	(carry)	0.553710	0.841127
0	(disabler)	(carry)	0.275445	0.835555
13	(initiator, support)	(carry)	0.296135	0.833784
14	(nuker, support)	(carry)	0.357340	0.827262
10	(support, disabler)	(carry)	0.164195	0.820298
12	(nuker, initiator)	(carry)	0.269640	0.798082
6	(pusher)	(carry)	0.050965	0.787469
8	(initiator, disabler)	(carry)	0.115945	0.786441
9	(nuker, disabler)	(carry)	0.156050	0.784703
1	(durable)	(carry)	0.165765	0.778020
2	(escape)	(carry)	0.273955	0.772825

Figure 15: all association rules generated by apriori algorithm upon iteration 4

Sorted in order of confidence, the selected hyperparameters produce rules #7, 4, 3, 5, 0, 6, 8, 9, 1, and 2 - all of which have the carry as the consequent, and the eight non carry roles as the antecedent.

Based on the high levels of confidence, the first hypothesis can be accepted since a strong relationship can be demonstrated where all other hero roles rely on the presence of a carry.

5.3 Clustering - K Means

5.3.1 Motivation and Intention

K-means clustering is used to understand the Hero composition of teams. In Dota 2, users are not given any information on what makes a team a good one. The intention of this clustering is to help beginners form a good team when they initially play the game. By understanding which type of Heroes are often chosen and common team compositions, beginner players can develop a team that will help them succeed in the game. This improves the user experience for players, because they are succeeding early in the game. In turn, this result drives up early-game retention. This is beneficial for the developers of Dota 2, because it will drive more players to remain in the game and play more of it.

5.3.2 Hypothesis

The hypothesis is that K Means should result in three different team types (3 clusters should be most optimal) such that one type is offensive, one type is defensive, and the last type is a balanced team. A balanced team is a team that has a good mix of defensive heroes and offensive heroes.

5.3.3. Pre-Processing & Data Interpretation

K-Means clustering was applied on the matches table. Each row in the matches table contains data for a distinct match. In each match, there are two teams playing against each other. Each team has five player slots, which contain the hero IDs selected by their respective players. The purpose of clustering this data is to understand which heroes are commonly chosen together on a team.

The initial step in pre-processing the Matches data was to read the data into a dataframe. The columns of the table are match IDs that distinguish each match, and the columns are the player slots for both teams. The cells corresponding to the player slots are the hero IDs. Because the player slot columns correspond to the two teams, the next step in pre-processing the data was to split the data into two tables. Each of the split tables contains data for one team in the match.

Next, the second of the split tables was appended into the first table. Each row in the resulting table represents a match for a given team. The matches and teams are easily identifiable by the row numbers and player slot values, and hence there is no need for match IDs.

The altered table was then transformed into a binary matrix. Each row in the binary matrix represents a particular match for a single team. The columns indices correspond to hero IDs and the value of each cell is a binary value. If the cell is true, or equal to 1, then the hero it corresponds to was selected for that specific match shown the table below. Otherwise, the hero ID was not selected for that match.

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	102	103	104	105	106	107	109	110	111	112
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	True	False	False	False	False	False	False	False	...	False	False	False	False	True	False	False	False	False	False
2	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	True	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

5 rows × 111 columns

Figure 16: sample of input data for k-means clustering

As a side note, the binary matrix begins at the index of 0 but according to the dataset, the hero Ids starts off at 1. Similarly, in the data set, hero id 24 is not present and hence the hero name corresponding to hero ID 25 shifts to slot 24. To account for these offsets, a function was built to shift data back to their original column. The documentation of the code explains more in depth on how the procedure is done.

The binary matrix is then applied to the KMeans function offered by Sklearn. A potential discovery has been made that using the K-Means offered by Sklearn is not ideal for clustering due to the function using euclidean distances as the distance metric for clustering. Euclidean distance does not work well with binary data. Other distance metrics can be considered such as Jaccard's distance similarity or Multiple Correspondence Analysis (Link).

However, after confirmation with Professor Lukasz Golab and TA Liuyan Chen, using K-Means offered by Sklearn (that implements Euclidean distance as the distance metric) is an acceptable approach for the binary matrix input due to KMeans clustering data at 111 dimensions. The 111 dimensions is from the binary matrix consisting of 111 columns (111 instead of 112 because of a missing value at column 24).

5.3.4 Results and Analysis

K-Means clustering have been applied on to different number of clusters. Using the Elbow method to help select the most optimal number of clusters by calculating the cluster's inertia scores, the cluster number should be set to 4. The table below represents the number of clusters corresponding with their inertia score. The graph below the table is plotted against the table values. From observation, there is an elbow-like bend when the number of clusters equal to 4. Therefore the number of clusters will be set as 4.

Number of clusters	Inertia Value
2	446337.1978
3	438014.248
4	424893.57
5	417254.7471
6	412575.794
7	406560.1441
8	404524.7333
9	397998.7033
10	396081.3589
11	393936.8044
12	391545.804

Figure 16: inertia values generated for k-means clusters from k=2 to k=12

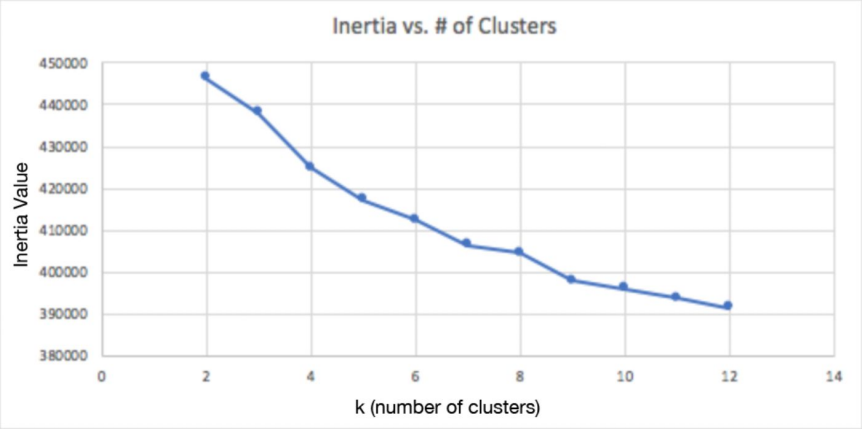

Figure 17: graph of intertia values at each k vs. k # of clusters

Cluster Centroids Interpretation

Running the K-Means function with the number of clusters set as 4 resulted in an array of 4 subarrays. Each sub array has 111 elements because there are 111 hero IDs. Hence, each index within the subarray represents a hero. Since there are 111 heros, there are 111 dimensions in each of the 4 clusters. Each value within the subarray is the centroid of the cluster for a specific dimension. By taking top 5 highest values and their corresponding array index (hero ID), the results are the five heros that are most frequently chosen with respect to the other 111 heroes for a given team.

A simplified version example is given in the following two charts, but in 3 dimensions:

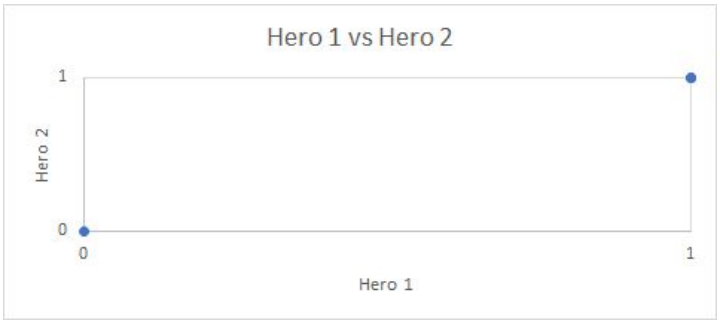

Figure 18: graph demonstrating whether hero 1 and hero 2 are selected together on a team given matches

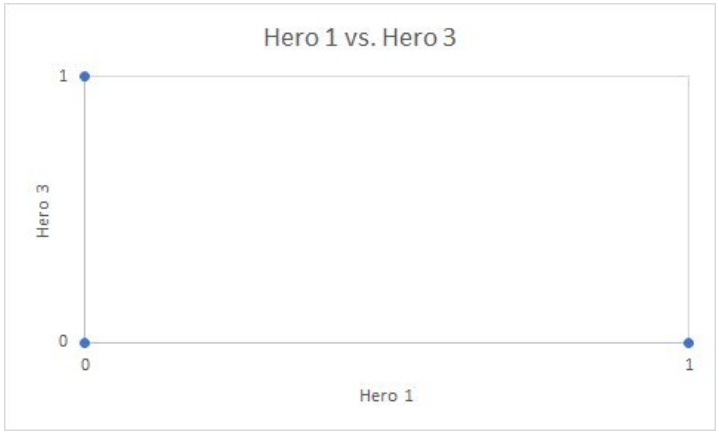

Figure 19: graph demonstrating whether hero 1 and hero 3 are selected together on a team given matches

The interpretation of the graphs above is that for some games, hero 1 and hero 2 are both chosen (first graph) or neither is selected. Then in other games, either hero 2 is chosen or hero 1 are chosen (second graph) or neither is selected. In the second graph, values are either on the y axis or x axis whereas on the first graph, values are at the upper-right most corner of the graph. As a result, the value for the centroid for hero 1 and 2 will be relatively higher than hero 1 and hero 3. If the centroids were available on both graphs, the cluster centroid value for the first graph will be higher than the centroid value for the second graph.

The clustering outcomes for the actual project are shown in the following tables: In Figure 20 below, the hero names associated with each cluster are shown. In Figure 21 below, the hero roles associated with each cluster are shown.

```
In [12]: #Table of hero names for a given cluster
         pd.DataFrame(clusterMapName)
```

Out[12]:

	Cluster1	Cluster2	Cluster3	Cluster4
0	Skywrath Mage	Skywrath Mage	Mirana	Mirana
1	Juggernaut	Tidehunter	Tidehunter	Juggernaut
2	Earth Spirit	Mirana	Skywrath Mage	Skywrath Mage
3	Invoker	Juggernaut	Clockwerk	Tidehunter
4	Venomancer	Zeus	Silencer	Phantom Lancer

Figure 20: list of hero names present in each cluster centroid where k=4

```
resultCategory
```

Out[14]:

	Cluster1	Cluster2	Cluster3	Cluster4
0	nuker	nuker	escape	escape
1	carry	initiator	initiator	carry
2	nuker	escape	nuker	nuker
3	nuker	carry	initiator	initiator
4	support	nuker	disabler	carry

Figure 21: list of hero roles present in each cluster centroid where k=4

With regards to the initial hypothesis, the result turned out to have 4 types of team compositions: one early-game aggressive offensive team, one late-game aggressive offensive team, one balanced team, and one defensive team.

Further analysis on the types of heroes in each cluster, cluster one appears to be offensive and early game aggressive. There are three nukers, one carry, and one support. Nukers are capable of ending opponents' heroes efficiently via high damaging spells, which makes this team effective damage-wise. Having a carry gives the team an advantage in the late game because it will eventually bear the

responsibility for the ultimate victory. The support helps the team strategically throughout the game letting the nukers and carry effectively know when and how to attack. Hence this cluster offered by K-Means for team composition is considered to be offensive and can deal significant damage in the beginning.

Cluster two consists of two nukers, an initiator, a carry, and an escape. The initiator is effective at the start of a fight. The two nukers assist with high damaging attacks to end the opponents' heroes. The support can be expected to provide strategies while the carry will offer significant contribution later in the game. Cluster two can be thought as balanced compared to cluster one due to having the initiator present and the escape. The nukers and initiator can do a lot of damage earlier on in the game and the carry can finish it off later in the game. The escape hero is fast at agility and can escape attacks when roaming enemy territory. Later in the game, the initiator gains defensive ability and can likely pair with the hero categorized as hero.

Cluster three is analyzed to be defensive by having two initiators, one disabler, one escaper, and one nuker. This team is considered to be defensive in the game. At the beginning, having two initiators and a nuker can help the team perform. The disabler's ability is to prevent opponents from attacking while the escape hero has an escape mechanism to avoid getting defeated. The hero will often be paired with a carry or nuker. Initiators have durability so that they can start a team fight and absorb enough damage to see it through. The only offensive hero is the nuker and thus makes the team oriented towards the defensive side.

Cluster four consists of two carries, an initiator, a nuker, and an escaper and so this team composition is analyzed as offensive but late-game aggressive. The nuker can quickly defeat enemies, the initiator starts off the fight and having to be more durable than the rest of the categorizes, the initiator will give the two carries time to farm and charge. The escaper roam alone into enemy territory providing insights while capable of avoiding attacks. Then two carries becomes essential later in the game as once they are charged and the team will become power mid-late in the game.

In addition, none of the top five heroes in each cluster were categorized as "durable", "jungler", and "pusher". Durable heroes are able to last longer in fights but if playing strategically, players generally want to avoid prolonged fights. Jungler type heroes can farm quickly to charge carries. Heroes that are "pushers" are primarily used to quickly siege towers, destroy them and barracks. However, this hold lesser importance to winning the game since killing the opponents' champions are the main objective.

6.0 Conclusion

Main findings from the analyses performed were three hero roles that are essential to a team, four common team compositions that vary in the aggressiveness/defensiveness of their playstyle, and XYZ critical factors that affect success in team fights.

From association rule mining, two conclusions are made. Firstly, the hypothesis that for all non carry roles, their presence has a strong relationship with the presence of a carry hero should be failed to reject (accept). This makes sense because carry roles obtain the greatest offensive power as a game progresses and are relied on to secure a victory. Secondly, the second hypothesis is failed to be rejected (accept) that teams have three essential roles; these were determined to be carry, support, and nuker roles through association rule mining.

Based on clustering using K Means, the results did not exactly match the hypothesis that there are 3 clusters representing three types of team compositions: offensive, balanced team (mix of offensive and defensive heroes), and defensive team. Using the elbow method, 4 clusters were determined to be most optimal. The first and second clusters were oriented towards offensive play styles, with the first cluster being more powerful in the early game, and the second cluster gaining power in the late game. The third cluster represented a balanced team composition whereas the 4th clustered represented a team composition suitable for a defensive playstyle.

The linear regression produced in this report had a coefficient of determination R^2 of 0.194, which suggests that although the model is has some power in predicting success in a team fight, there are is variance present in the data that is not explained or accounted for. With a coefficient of 0.92, the number of players involved in a team fight was determined to be the most powerful predictor of a team's success. Another notable insight is that statistics that are not specific to a game (i.e. a team's overall skill, number of matches played, and matches won) have a higher coefficient than in game statistics (i.e. gold, experience, and last hits). This suggests that even if players have fallen behind the other players in a match, they can still do well in a team fight if they are skilled at the game. Novice players can use these insights to not only to understand when they should and should not engage the enemy team, but also to inform in game strategy so that they can better see and respond to scenarios where enemies are grouping and preparing for a team fight.

Novice players can extract multiple benefits from the analyses performed. Firstly, they can use the association rule to not only adjust tailor their in game behaviour to better enable the carry hero, but also to explore and improve their proficiency towards controlling the three essential hero roles. Secondly, they can use the clusters from K-means to identify the type of team they play best with, or identify gaps in their skill and learn a different play style - whether that be more or less aggressive than what they are accustomed to.

Critical lessons learned about data mining in general was the difficulty and effort required for feature engineering and the limitations of computing power. To begin, the majority of effort in this project was spent towards feature engineering. For example, the class variable for linear regression had to be adjusted to a kill death difference because situations where a team had 0 deaths in a team fight resulted in a denominator with a zero value. Furthermore, pre-processing the raw data for linear regression required not only the engineering of total gold, experience, last hit, matches won, and trueskill of players from each team involved in a team fight, but also required normalizing gold and experience

differences based on elapsed time during a match. As well, both apriori rule mining and clustering required the transformation and merging of datasets in order to determine the hero roles present on each team. In terms of computing power limitations, generating an elbow method graph using the Silhouette metric within Python required 75GB of ram, while calculating gold, experience, and last hit differences between each team required an hour and a half of compute time on a quad core processor. In the future, cloud compute methods such as Amazon EC2 should be explored.

To conclude, there were three hero roles that are essential to a team, four common team compositions that vary in the aggressiveness/defensiveness of their playstyle, and the number of players and their level of experience that affect success in team fights. This project is interesting and non-trivial because it uses Supervised and Unsupervised learning to build an understanding of the types of roles in Dota 2, and how they can impact the success of users. Using the results from this survey, beginner players to Dota 2 can form teams to be successful in their early games, improving their experience playing the game. The overall improvements made to user experience can in turn improve the retention and other key metrics for Dota 2.

7.0 Appendix - Source Code

(Starts on next page)

7.1 Linear Regression

In [1]:

```
import pandas as pd
import math
from sklearn import linear_model
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import KFold
```

In [2]:

```
player_time = pd.read_csv('player_time.csv')
player_ratings = pd.read_csv('player_ratings.csv')
```

In [3]:

```
player_time.head()
```

Out[3]:

	match_id	times	gold_t_0	lh_t_0	xp_t_0	gold_t_1	lh_t_1	xp_t_1	gc
0	0	0	0	0	0	0	0	0	0
1	0	60	409	0	63	142	1	186	16
2	0	120	546	0	283	622	4	645	33
3	0	180	683	1	314	927	9	1202	43
4	0	240	956	1	485	1264	11	1583	53

5 rows × 32 columns

In [4]:

```
player_ratings.head()
```

Out[4]:

	account_id	total_wins	total_matches	trueskill_mu	trueskill_sigma
0	236579	14	24	27.868035	5.212361
1	-343	1	1	26.544163	8.065475
2	-1217	1	1	26.521103	8.114989
3	-1227	1	1	27.248025	8.092217
4	-1284	0	1	22.931016	8.092224

In [5]:

```
players = pd.read_csv('players.csv')
```

In [6]:

```
players.head()
```

Out[6]:

	match_id	account_id	hero_id	player_slot	gold	gold_spent	gold_p
0	0	0	86	0	3261	10960	347
1	0	1	51	1	2954	17760	494
2	0	0	83	2	110	12195	350
3	0	2	11	3	1179	22505	599
4	0	3	67	4	3307	23825	613

5 rows × 73 columns

In [7]:

```
teamfights = pd.read_csv('teamfights.csv')
teamfights.head()
```

Out[7]:

	match_id	start	end	last_death	deaths
0	0	220	252	237	3
1	0	429	475	460	3
2	0	900	936	921	3
3	0	1284	1328	1313	3
4	0	1614	1666	1651	5

In [ ]:

```
teamfights_players = pd.read_csv('teamfights_players.csv')
```

To predict the kill-death ratio for a team fight, we will need to first establish the class variable by melting the dataframe so each row represents a teamfight for one team (two rows per fight). Before modifying the teamfight_players dataset to form the class variable we will need to extract some other pieces of information first, specifically xp delta and a column to estimate whether the hero was in the fight or not based on gold and xp delta.

In [ ]:

```
teamfights_players = teamfights_players[0:50000]#cutting data to wor
k on feasible dataset
#remove damage, buyback columns as they are not used
teamfights_players = teamfights_players.drop(['damage', 'buybacks'],
 axis = 1)
```

In [13]:

```
#add 'active' column
def isActive(gold, xp, slot):#A player is 'active' during (not neces
sarily present in) a team fight if the gain gold and xp
    if gold==0 or xp==0:
        return -1
    else:
        return slot

teamfights_players['active'] = teamfights_players.apply(#Add active
 column
    lambda x: isActive(x['xp_end'] - x['xp_start'], x['gold_delta'],
 x['player_slot']), axis = 1)
```

In [14]:

```
#add fight index to use as primary key
teamfights_players['fight_id'] = pd.Series(teamfights_players.index.
values/10).apply(math.floor)
```

In [15]:

```
#remove fights with high death counts (above three per player)
invalid_fights = teamfights_players.loc[(teamfights_players['deaths'
] > 3),('fight_id')].values#list of invalid fights
teamfights_players = teamfights_players.loc[(teamfights_players['fig
ht_id'].isin(invalid_fights))==False]#select fights not in list
```

In [16]:

```
teamfights_players.head()
```

Out[16]:

	match_id	player_slot	deaths	gold_delta	xp_end	xp_start	active	f
0	0	0	0	173	536	314	0	C
1	0	1	1	0	1583	1418	-1	C
2	0	2	0	0	391	391	-1	C
3	0	3	0	123	1775	1419	3	C
4	0	4	0	336	1267	983	4	C

In [17]:

```
def findTeam(player_slot):#Check if team is radiant or not
    if player_slot < 100:
        return True
    else:
        return False

KDR = teamfights_players.loc[:,('match_id', 'player_slot', 'deaths',
 'fight_id')]
KDR['radiant'] = KDR.apply(lambda x: findTeam(x['player_slot']), axi
s = 1)#add team colum
KDR.head()
```

Out[17]:

	match_id	player_slot	deaths	fight_id	radiant
0	0	0	0	0	True
1	0	1	1	0	True
2	0	2	0	0	True
3	0	3	0	0	True
4	0	4	0	0	True

In [18]:

```
deathCount = KDR.groupby(['fight_id', 'radiant'])['deaths'].sum()#Fi
nd sum of deaths on each team for each fight
```

In [19]:

```
deathCount[0:10]
```

Out[19]:

```
fight_id  radiant
0         False      2
          True       1
1         False      3
          True       0
2         False      3
          True       0
3         False      3
          True       0
4         False      3
          True       2
Name: deaths, dtype: int64
```

In [20]:

```
#create table with columns for each player slot
active = teamfights_players.pivot_table(columns = ['player_slot'], i
ndex = ['match_id', 'fight_id'])
active = active['active']#Make a table with each player slot for eac
h fight to find aggregate statistics for each player in the fight
active.head()
```

Out[20]:

	player_slot	0	1	2	3	4	128	129	130	131	132
match_id	fight_id										
0	0	0	-1	-1	3	4	128	129	130	-1	132
	1	0	1	2	3	4	128	129	130	131	132
	2	0	1	2	3	4	128	-1	-1	-1	132
	3	0	1	2	3	4	128	129	130	131	132
	4	0	1	2	3	4	128	129	130	-1	132

In [21]:

```
#make a list of fights to remove if a team is absent, as this is not
 useful data
def findEmptyTeam(x):
    players = [x['P0'], x['P1'],x['P2'], x['P3'], x['P4']]
    if sum([1 for i in range(5) if players[i] < 0]) == 5:
        return True
    players = [x['P128'], x['P129'],x['P130'], x['P131'], x['P132']]
    if sum([1 for i in range(5) if players[i] < 0]) == 5:
        return True
    return False
```

In [22]:

```
#take the players in a fight, the match, time and desired metric
#return the sum of that metric for players active in the fight
def pre_stats(time, match, players, metric):
    #measures = ['gold', 'xp', 'lh']
    time = max(0, time)#index error for negative times
    stats = player_time.loc[((time - 60) < player_time['times']) & (
player_time['times'] <= time) & (player_time['match_id'] == match)]
    #select the row from player_time before the start of the fight f
or current metric
    team = sum([stats[metric + '_t_' + str(players[i])] for i in ran
ge(5) if players[i] >= 0]).iloc[0]/sum(1 for i in range(5) if player
s[i] >= 0)
    #calculate the average of the statistic over all players
    return team
```

In [23]:

```
#Create a dataframe for cumulative team stats
activeF = active.reset_index()#reset index for merge

playerTimes = teamfights.merge(activeF, left_index = True, right_ind
ex = True)
#create dataframe for aggregate statistics for each fight
aggStats = playerTimes[['fight_id']]
playerTimes = playerTimes.rename(index = str, columns = {0:'P0', #it
 is difficult to deal with columns with numeric names
                                                         1:'P1',
                                                         2:'P2',
                                                         3:'P3',
                                                         4:'P4',
                                                         128:'P128',
                                                         129:'P129',
                                                         130:'P130',
                                                         131:'P131',
                                                         132:'P132',
                                                         'match_id_x':'match_id'
})
playerTimes.head()
```

Out[23]:

	match_id	start	end	last_death	deaths	match_id_y	fight_id	P0	I
0	0	220	252	237	3	0	0	0	-
1	0	429	475	460	3	0	1	0	1
2	0	900	936	921	3	0	2	0	1
3	0	1284	1328	1313	3	0	3	0	1
4	0	1614	1666	1651	5	0	4	0	1

In [24]:

```
playerTimes['remove'] = playerTimes.apply(lambda x: findEmptyTeam(x
), axis = 1).values#remove empty teams
invalid_teams = playerTimes['fight_id'].loc[playerTimes['remove']].v
alues
playerTimes = playerTimes.loc[playerTimes['fight_id'].isin(invalid_t
eams)==False]#check if the fight is invalid
aggStats = aggStats.loc[aggStats['fight_id'].isin(invalid_teams)==Fa
lse]#do the same for aggstats
#active = aggStats.loc[acitve['fight_id'].isin(invalid_teams)==Fals
e]
```

In [26]:

```
#Add an aggregate coloumn for each team for each metric
aggStats['radiant_gold'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P0'], x['P1'
], x['P2'], x['P3'], x['P4']], 'gold'), axis = 1).values
aggStats['dire_gold'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P128'], x['P1
29'], x['P130'], x['P131'], x['P132']], 'gold'), axis = 1).values
aggStats['radiant_xp'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P0'], x['P1'
], x['P2'], x['P3'], x['P4']], 'xp'), axis = 1).values
aggStats['dire_xp'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P128'], x['P1
29'], x['P130'], x['P131'], x['P132']], 'xp'), axis = 1).values
aggStats['radiant_lh'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P0'], x['P1'
], x['P2'], x['P3'], x['P4']], 'lh'), axis = 1).values
aggStats['dire_lh'] = playerTimes.apply(
    lambda x: pre_stats(x['start'], x['match_id'], [x['P128'], x['P1
29'], x['P130'], x['P131'], x['P132']], 'lh'), axis = 1).values
```

In [28]:

```
player_matches = players[['match_id', 'account_id', 'player_slot']]
#A table to use to find wins, matches, and tureskill for each player
player_matches.head()
```

Out[28]:

	match_id	account_id	player_slot
0	0	0	0
1	0	1	1
2	0	0	2
3	0	2	3
4	0	3	4

In [29]:

```
player_ratings.set_index('account_id')
player_ratings.head()
```

Out[29]:

	account_id	total_wins	total_matches	trueskill_mu	trueskill_sigma
0	236579	14	24	27.868035	5.212361
1	-343	1	1	26.544163	8.065475
2	-1217	1	1	26.521103	8.114989
3	-1227	1	1	27.248025	8.092217
4	-1284	0	1	22.931016	8.092224

In [30]:

```
match_ratings = pd.merge(player_matches, player_ratings, how='left',
 on='account_id', sort=False)
#Aanonomous players will wash out the wins and matches data.
#These players shoulde not be removed as they still participate in f
ights and a lot of data would be missing without them.
#For now, I will find the average number of wins and matches for ide
ntifiable players,
#and assign this value to all '0' account_ids
#remove player 0
avgGame = match_ratings[['account_id', 'total_wins', 'total_matches'
]]
avgGame = avgGame.loc[avgGame['account_id'] > 0]#find the average nu
mber of games played by non-anonymous users
win = avgGame['total_wins'].mean()
match = avgGame['total_matches'].mean()

match_ratings.loc[match_ratings['account_id'] == 0,'total_wins'] = w
in#set anonymous users win and match numbers to the average
match_ratings.loc[match_ratings['account_id'] == 0,'total_matches']
= match
```

In [31]:

```
match_ratings = match_ratings.loc[match_ratings['trueskill_mu'].notn
ull()]#remove NaN rows from the above join due index mismatch
```

In [32]:

```
def static_stats(match_id, players, metric):#find aggregate wins, ma
tches, or trueskill for a team
    stats = match_ratings[(match_ratings['match_id']==match_id)&(mat
ch_ratings['player_slot'].isin(players))]
    return stats[[metric]].sum()
```

In [33]:

```
#Make a new aggregate column for each team for each statis statistic
aggStats['radiant_wins'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(5)],
 'total_wins'), axis = 1).values
aggStats['dire_wins'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(128,
133)], 'total_wins'), axis = 1).values

aggStats['radiant_matches'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(5)],
 'total_matches'), axis = 1).values
aggStats['dire_matches'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(128,
133)], 'total_matches'), axis = 1).values

aggStats['radiant_trueskill'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(5)],
 'trueskill_mu'), axis = 1).values
aggStats['dire_trueskill'] = playerTimes.apply(lambda x:
    static_stats(x['match_id'], [x['P' + str(i)] for i in range(128,
133)], 'trueskill_mu'), axis = 1).values
```

In [34]:

```
aggStats.head()
```

Out[34]:

	fight_id	radiant_gold	dire_gold	radiant_xp	dire_xp	radiant_lh	
0	0	711.0	939.25	685.333333	833.25	4.666667	
1	1	1708.8	1953.20	1812.000000	1789.00	15.400000	
2	2	4662.2	5283.50	4805.000000	4520.50	48.600000	
3	3	7314.0	6465.20	7821.200000	6307.60	72.800000	
4	4	9525.4	9413.50	10450.600000	9356.00	94.000000	

In [35]:

```
def playerCount(players):
    return sum([1 for i in range(5) if players[i] > 0])
```

In [36]:

```
#count the number of active players on each team in each fight
aggStats['radiant_count'] = playerTimes.apply(lambda x:
    playerCount([x['P' + str(i)] for i in range(5)]), axis = 1).valu
es
aggStats['dire_count'] = playerTimes.apply(lambda x:
    playerCount([x['P' + str(i)] for i in range(128,133)]), axis = 1
).values
```

In [37]:

```
aggStats.head()
```

Out[37]:

	fight_id	radiant_gold	dire_gold	radiant_xp	dire_xp	radiant_lh	
0	0	711.0	939.25	685.333333	833.25	4.666667	
1	1	1708.8	1953.20	1812.000000	1789.00	15.400000	
2	2	4662.2	5283.50	4805.000000	4520.50	48.600000	
3	3	7314.0	6465.20	7821.200000	6307.60	72.800000	
4	4	9525.4	9413.50	10450.600000	9356.00	94.000000	

In [38]:

```
def getDifference(fight_id):#calculate the relative difference betwe
en the number of kills and deaths
    kills = deathCount[fight_id][True] + deathCount[fight_id][False]
    difference = deathCount[fight_id][False] - deathCount[fight_id][
True]
    KDD = difference / kills
    return KDD
```

In [39]:

```
aggStats['KDD'] = playerTimes['fight_id'].apply(getDifference).value
s
```

In [40]:

```
aggStats.head()
```

Out[40]:

	fight_id	radiant_gold	dire_gold	radiant_xp	dire_xp	radiant_lh	
0	0	711.0	939.25	685.333333	833.25	4.666667	
1	1	1708.8	1953.20	1812.000000	1789.00	15.400000	
2	2	4662.2	5283.50	4805.000000	4520.50	48.600000	
3	3	7314.0	6465.20	7821.200000	6307.60	72.800000	
4	4	9525.4	9413.50	10450.600000	9356.00	94.000000	

In [41]:

```
#naive regression, pre variable engineering
X = aggStats[['radiant_gold',
              'dire_gold',
              'radiant_xp',
              'dire_xp',
              'radiant_lh',
              'dire_lh',
              'radiant_wins',
              'dire_wins',
              'radiant_matches',
              'dire_matches',
              'radiant_trueskill',
              'dire_trueskill',
              'radiant_count',
              'dire_count']]
Y = aggStats[['KDD']]
```

In [42]:

```
lm = linear_model.LinearRegression()
model = lm.fit(X, Y)
predictions = lm.predict(X)
predictions[0:5]
print(lm.score(X, Y))
lm.coef_
#lm.intercept_
```

```
0.225307855302
```

Out[42]:

```
array([[ -3.89564582e-06,   4.94446206e-06,   4.12761170
e-06,
         -4.68437584e-06,  -2.78659158e-04,   2.89515953
e-04,
         -6.98116871e-04,  -2.21857446e-03,   3.01567826
e-04,
          1.19427876e-03,   1.69543981e-03,  -8.03938036
e-04,
          1.64666834e-01,  -1.67355848e-01]])
```

In [43]:

```
#variable engineering
#take the difference between each variable for each team, and divide
 by the sum.
deltaStats = pd.DataFrame()
variables = ['gold', 'xp', 'lh', 'wins', 'matches', 'trueskill', 'co
unt']
deltaStats['KDD'] = aggStats['KDD']
for i in range(len(variables)):#calculate the relative differnce of
 each statistic
    delta = (aggStats['radiant_' + variables[i]] - aggStats['dire_'
+ variables[i]])
    total = (aggStats['radiant_' + variables[i]] + aggStats['dire_'
+ variables[i]])
    total = total.apply(lambda x: max(x,1))
    deltaStats[variables[i]] = delta/total
```

In [44]:

```
deltaStats.head()
```

Out[44]:

	KDD	gold	xp	lh	wins	matches	tru
0	0.333333	-0.138312	-0.097404	-0.363636	-0.592669	-0.495237	-0.2
1	1.000000	-0.066739	0.006387	-0.167568	-0.362544	-0.281416	-0.0
2	1.000000	-0.062469	0.030508	-0.083883	0.461478	0.552112	0.33
3	1.000000	0.061600	0.107129	-0.038309	-0.362544	-0.281416	-0.0
4	0.200000	0.005908	0.055264	-0.090689	-0.293217	-0.193915	0.02

In [45]:

```
X1 = deltaStats.loc[:,variables]#Decision variables
Y1 = deltaStats.loc[:,'KDD']#class variables
X1.loc[:,'xp'] = X1.loc[:,'xp']*2#delta xp is small, doubling to bri
ng it to scale with other variables in regression
#deltaStats
```

In [47]:

```
X1 = X1.replace(np.nan, 0)#remove NaN values before regression
```

In [51]:

```
lm = linear_model.LinearRegression()
model = lm.fit(X1, Y1)#fit model to dataset
print(lm.score(X1, Y1))#check score
(pd.DataFrame([lm.coef_], columns = variables))
#lm.intercept_
```

```
0.212299350847
```

Out[51]:

	gold	xp	lh	wins	matches	trueskill	co
0	-0.129747	0.088434	-0.041339	0.001833	-0.032074	0.290963	0.915

In [49]:

```
#Model Evaluation:
score = []
kFold = KFold(10, True, 1)
for test, train in kFold.split(X1['wins']):
    model = lm.fit(X1.iloc[train],Y1.iloc[train])
    score.append(lm.score(X1.iloc[test], Y1.iloc[test]))
```

In [50]:

```
sum(score)/len(score)
```

Out[50]:

```
0.19547169883585364
```

7.2 Apriori Association Code

In [132]:
```
#import libraries
import pandas as pd
import numpy as np
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

#import data
matches = pd.read_csv('test_player.csv')
champions = pd.read_csv('hero_names.csv')

#taking a peek at the matches table, seems like we have 2 sets of player
 slots (0-4, 128-132)
matches.head(n=12)
```

Out[132]:

	match_id	account_id	hero_id	player_slot
0	50000	117784	96	0
1	50000	158361	84	1
2	50000	158362	46	2
3	50000	137970	85	3
4	50000	1090	39	4
5	50000	2391	9	128
6	50000	2393	75	129
7	50000	2394	106	130
8	50000	36737	74	131
9	50000	2392	62	132
10	50001	32932	44	0
11	50001	0	10	1

Transforming test_player.csv so that each row is a team

Step 1: Pivot the data so that each entry is all the heroes in a match

In [133]:
```
#drop the account_id column, it won't be used
matches.drop(['account_id'],axis=1)
#collapse data into columns so that each row becomes all the hero IDs on
 a team
competitor_rows = pd.pivot_table(matches, values='hero_id', index=['matc
h_id'], columns='player_slot')
competitor_rows.head(n=10)
```

Out[133]:

player_slot	0	1	2	3	4	128	129	130	131	132
match_id										
50000	96	84	46	85	39	9	75	106	74	62
50001	44	10	57	2	106	58	61	21	18	14
50002	74	7	42	99	88	69	8	25	26	79
50003	44	15	110	56	94	2	101	32	7	72
50004	98	26	73	51	46	2	106	50	65	21
50005	16	39	36	71	46	27	106	33	21	93
50006	104	44	21	62	73	86	19	74	76	67
50007	68	23	2	76	39	65	73	21	9	30
50008	112	11	39	104	101	55	19	16	79	63
50009	8	5	42	3	92	9	31	12	74	21

Step 2: Split and append the data so that each row is an individual team of 5 players

In [4]:
```
#Here we want to combine the data
#split the data for team 1 (player slot 0-4) and team 2 (player slot 128
-132)
split1,split2 = np.split(competitor_rows,2,axis=1)

#set the columns to be the same so that we can collapse it back into 1 t
able
split2.columns = split1.columns
index0 = split1.index.tolist()
index1 = split2.index.tolist()
```

In [5]:
```
#generate new indexes, so that each pair of even number represents team
 1 and odd number represents team 2
#new indexes are needed because if data is appended with the same index
 (match_id), the duplicate entry will be removed
i = 0
while i <= 99999:
    index0[i] = i*2
    index1[i] = i*2 + 1
    i += 1

print(len(index0))
```

```
100000
```

In [6]:
```
split1.index = index0
split2.index = index1
```

In [9]:
```
#join the two tables back together with their new indexes
join = split1.append(split2)
join = join.sort_index()
join.head(n=20)
```

Out[9]:

player_slot	0	1	2	3	4
0	96	84	46	85	39
1	9	75	106	74	62
2	44	10	57	2	106
3	58	61	21	18	14
4	74	7	42	99	88
5	69	8	25	26	79
6	44	15	110	56	94
7	2	101	32	7	72
8	98	26	73	51	46
9	2	106	50	65	21
10	16	39	36	71	46
11	27	106	33	21	93
12	104	44	21	62	73
13	86	19	74	76	67
14	68	23	2	76	39
15	65	73	21	9	30
16	112	11	39	104	101
17	55	19	16	79	63
18	8	5	42	3	92
19	9	31	12	74	21

Using hero role data from hero_names.csv so that we get a list of hero roles on each team

Step 1: Define a function to identify the hero row from hero_names.csv given a hero_id

In [10]:
```
#create function to determine role of each champion on a team
def find_role(int):
    if (int == 0):
        int = 113
    champ_row = champions.loc[champions["hero_id"] == int]
    role =  champ_row["role"]
    role = role.values[0]
    return role
```

Step 2: Created a nested list of hero roles on each team

In [11]:
```
#create nested list of champion roles present in each team
dataset = []
for row in join.iterrows():
    temp_list = []
    for role in range(0,5):
        temp_list.append(find_role(row[1][role]))
    dataset.append(temp_list)
```

In [19]:
```
#visually inspecting the first few entries of the list to make sure it's
 what we want
dataset[:5]
```

Out[19]:
```
[['durable', 'support', 'carry', 'durable', 'nuker'],
 ['escape', 'disabler', 'escape', 'nuker', 'escape'],
 ['carry', 'escape', 'support', 'initiator', 'escape'],
 ['jungler', 'pusher', 'carry', 'carry', 'disabler'],
 ['nuker', 'initiator', 'durable', 'durable', 'disabler']]
```

Transforming the nested list into a boolean table that indicates the presence of a role

In [20]:
```
#Use TransactionEncoder to generate a boolean table as demonstrated in t
utorial
oht = TransactionEncoder()
oht_ary = oht.fit(dataset).transform(dataset)
df = pd.DataFrame(oht_ary, columns=oht.columns_)

#Visually inspecting the output
df.head()
```

Out[20]:

	carry	disabler	durable	escape	initiator	jungler	nuker	pusher	support
0	True	False	True	False	False	False	True	False	True
1	False	True	False	True	False	False	True	False	False
2	True	False	False	True	True	False	False	False	True
3	True	True	False	False	False	True	False	True	False
4	False	True	True	False	True	False	True	False	False

Association Rule Mining Begins

Iteration 1: min_support = 0.6, min_confidence (threshold): 0.95

In [124]:
```
#find frequent champion role sets that have a threshold
frequent_roleset1 = apriori(df, min_support=0.60, use_colnames=True)
roleset_frame1 = frequent_roleset1.sort_values(['support'], ascending =[
0])
roleset_frame1
```

Out[124]:

	support	itemsets
0	0.874320	(carry)
2	0.707660	(support)
1	0.658295	(nuker)
3	0.615020	(support, carry)

In [125]:
```
rules = association_rules(frequent_roleset1, metric="confidence", min_th
reshold=0.95)
rule_frame1 = pd.DataFrame(rules[['antecedents', 'consequents', 'suppor
t', 'confidence', 'lift']])
rule_frame1 = rule_frame1.sort_values(['confidence'], ascending=[0])
rule_frame1
```

Out[125]:

	antecedents	consequents	support	confidence	lift

Iteration 2: min_support = 0.6, min_confidence (threshold): 0.85

In [126]:
```
#run apriori association rule mining  >> ITERATION 2
frequent_roleset2 = apriori(df, min_support=0.60, use_colnames=True)
pd.DataFrame(frequent_roleset2)
```

Out[126]:

	support	itemsets
0	0.874320	(carry)
1	0.658295	(nuker)
2	0.707660	(support)
3	0.615020	(support, carry)

In [127]:
```
rules = association_rules(frequent_roleset2, metric="confidence", min_th
reshold=0.85)
rule_frame2 = pd.DataFrame(rules[['antecedents', 'consequents', 'suppor
t', 'confidence']])
rule_frame2 = rule_frame2.sort_values(['confidence'], ascending=[0])
rule_frame2
```

Out[127]:

	antecedents	consequents	support	confidence
0	(support)	(carry)	0.61502	0.86909

Iteration 3: min_support = 0.5, min_confidence (threshold): 0.6

In [128]:
```
#run apriori association rule mining  >> ITERATION 3
frequent_roleset3 = apriori(df, min_support=0.5, use_colnames=True)
pd.DataFrame(frequent_roleset3)
```

Out[128]:

	support	itemsets
0	0.874320	(carry)
1	0.544965	(initiator)
2	0.658295	(nuker)
3	0.707660	(support)
4	0.553710	(nuker, carry)
5	0.615020	(support, carry)

In [129]:
```
rules = association_rules(frequent_roleset3, metric="confidence", min_th
reshold=0.6)
rule_frame3 = pd.DataFrame(rules[['antecedents', 'consequents', 'suppor
t', 'confidence']])
rule_frame3 = rule_frame3.sort_values(['confidence'], ascending=[0])
rule_frame3
```

Out[129]:

	antecedents	consequents	support	confidence
2	(support)	(carry)	0.61502	0.869090
0	(nuker)	(carry)	0.55371	0.841127
3	(carry)	(support)	0.61502	0.703427
1	(carry)	(nuker)	0.55371	0.633304

Iteration 4: min_support = 0.05, min_confidence (threshold): 0.75

In [130]:
```
#run apriori association rule mining  >> ITERATION 4
frequent_roleset4 = apriori(df, min_support=0.05, use_colnames=True)
roleset_frame = pd.DataFrame(frequent_roleset4)
```


In [131]:
```
rules = association_rules(frequent_roleset4, metric="confidence", min_th
reshold=0.75)
rule_frame4 = pd.DataFrame(rules[['antecedents', 'consequents', 'suppor
t', 'confidence']])
rule_frame4 = rule_frame4.sort_values(['confidence'], ascending=[0])
rule_frame4
```

Out[131]:

	antecedents	consequents	support	confidence
7	(support)	(carry)	0.615020	0.869090
4	(jungler)	(carry)	0.052315	0.847618
3	(initiator)	(carry)	0.460495	0.844999
5	(nuker)	(carry)	0.553710	0.841127
0	(disabler)	(carry)	0.275445	0.835555
13	(initiator, support)	(carry)	0.296135	0.833784
14	(nuker, support)	(carry)	0.357340	0.827262
10	(support, disabler)	(carry)	0.164195	0.820298
12	(nuker, initiator)	(carry)	0.269640	0.798082
6	(pusher)	(carry)	0.050965	0.787469
8	(initiator, disabler)	(carry)	0.115945	0.786441
9	(nuker, disabler)	(carry)	0.156050	0.784703
1	(durable)	(carry)	0.165765	0.778020
2	(escape)	(carry)	0.273955	0.772825
16	(nuker, initiator, support)	(carry)	0.152680	0.770547
11	(durable, support)	(carry)	0.109315	0.761910
15	(initiator, support, disabler)	(carry)	0.055765	0.750034

7.3 K-Means Code

In [1]:

```
# import required libraries
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib


#import data
# matches = pd.read_csv('test_player.csv')
# champions = pd.read_csv('hero_names_category.csv')
# heroes = pd.read_csv('hero_names.csv')
matches = pd.read_csv('./dota-2-matches/test_player.csv')
champions = pd.read_csv('./dota-2-matches/hero_names_category.csv')

heroes = pd.read_csv('./dota-2-matches/hero_names.csv')
```

In [2]:

```
#remove account_id column
matches = matches.drop("account_id",axis=1)
```

In [3]:

```
#Returns a map of heroId to name
def mapId2Name():
    mapheroes = {}
    for row in heroes.itertuples():
        mapheroes[row.hero_id] = row.localized_name
    return mapheroes

#Returns a map of name to Id
def mapName2Id():
    mapId = {}
    for row in heroes.itertuples():
        mapheroes[row.localized_name] = row.hero_id
    return mapId


def searchHeroName(heroId):
    mapToReturn = mapId2Name()
    return mapToReturn[heroId]


def searchHeroId(heroName):
    mapToReturn = mapName2Id()
    return mapToReturn[heroName]
```

In [4]:

```
#understanding player slot
matches.head(n=5)
```

Out[4]:

	match_id	hero_id	player_slot
0	50000	96	0
1	50000	84	1
2	50000	46	2
3	50000	85	3
4	50000	39	4

In [5]:

```
#collapse data into columns
#The interpretation of this table is that the row represent that par
ticular match ID. The column names are the player
#slots. They are not the individual players but rather the slots tha
t the players play in.
#The value within each cell is the hero id.
competitor_rows = pd.pivot_table(matches, values='hero_id', index=[
'match_id'], columns='player_slot')
competitor_rows.head(n=5)
```

Out[5]:

player_slot	0	1	2	3	4	128	129	130	131	132
match_id										
50000	96	84	46	85	39	9	75	106	74	62
50001	44	10	57	2	106	58	61	21	18	14
50002	74	7	42	99	88	69	8	25	26	79
50003	44	15	110	56	94	2	101	32	7	72
50004	98	26	73	51	46	2	106	50	65	21

In [6]:

```
#Current each row consists of two teams. Therefore, we split datafra
me vertically to separate the two teams.
#The 2nd section of the dataframe now gets appended to the end of th
e 1 section.
#Each row in the resultant dataframe corresponds with only one team
 rather than two.
section = competitor_rows
sectionA,sectionB  = np.split(section, 2, axis=1)
sectionB.columns = sectionA.columns
sectionA.append(sectionB,ignore_index = True)
sectionA.head(n=5)
```

Out[6]:

player_slot	0	1	2	3	4
match_id					
50000	96	84	46	85	39
50001	44	10	57	2	106
50002	74	7	42	99	88
50003	44	15	110	56	94
50004	98	26	73	51	46

In [7]:

```
#The table above now gets transformed into a binary matrix. Each col
umn name corresponds to the hero id.
#Each row corresponds to a particular match id. However, it's worth
 mentioning that the match id values gets reset to
#start at one but the current situation does not consider the match
 id for clustering so the result is not affected.

#The interpretation of the binary matrix starts off with each row re
presenting a match for one team
#Each column in the binary matrix corresponds to a hero Id
#The value of each cell represent whether that particular hero is ch
osen for that particular match
#for that specific team.

#It's worth mentioning that the dataframe starts off at column index
 0. However, hero Ids start off at 1
#This means that every player's index is offset by one. This situati
on has been accounted for.

dataset=[]
for row in sectionA.iterrows():
    index, data = row
    dataset.append(data.tolist())
#it transforms the input dataset (a Python list of lists) into a one
-hot encoded NumPy boolean array
oht = TransactionEncoder()
oht_ary = oht.fit(dataset).transform(dataset)
df = pd.DataFrame(oht_ary, columns=oht.columns_)
df.head(n=5)
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	
0	False	False	False	False	False	False	False	False	False	False	...	Fa
1	False	False	True	False	False	False	False	False	False	False	...	Fa
2	False	False	False	False	False	False	False	True	False	False	...	Fa
3	False	False	False	False	False	False	False	False	False	False	...	Fa
4	False	False	False	False	False	False	False	False	False	False	...	Fa

5 rows × 111 columns

In [8]:

```
#Converting the binary string matrix to a binary integer matrix
# 0: False
# 1: True
df = df*1
df.head(n=5)
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	102	103	104	105	106	107	109	110	1
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

5 rows × 111 columns

In [9]:

```
# create kmeans object and call fit
#Applies Kmeans algorithm
KM = KMeans(n_clusters=4, init='k-means++', random_state=170)
KM = KM.fit(df)
print("The cluster centroids are: \n", KM.cluster_centers_)
print("Cluster_label:\n", KM.labels_)
print("Cluster_inertia:\n", KM.inertia_)
```

```
The cluster centroids are:
 [[  2.10921516e-05   9.41342726e-02   6.49638270e-02
3.55613676e-02
    3.07945414e-02   8.30819852e-02   2.82212989e-02
1.23009428e-01
    1.18242602e-01   7.59317458e-02   1.79283289e-02  -
3.11417558e-14
    4.49262829e-02   2.14929025e-02   1.06916117e-01
2.35177490e-02
    3.06047120e-02   3.57511970e-02   3.61097636e-02
7.61004830e-02
    4.54114024e-02   1.76220150e-13   5.73706524e-02
2.99719474e-02
    1.04638164e-01   7.35694248e-02   3.96321529e-02
1.08371475e-01
    2.63019131e-02   7.72183670e-02   4.89548839e-02
4.49684672e-02
    2.43192508e-02   3.61097636e-02   4.70987745e-02
6.67566598e-02
    2.07757693e-02   1.40051887e-02   1.49859737e-01
3.15960431e-02
    3.09210943e-02   7.61426673e-02   2.11976124e-02
7.53833499e-02
    1.83923562e-02   9.49990509e-02   5.05367953e-02
2.58167936e-02
    2.40661450e-02  -4.78922457e-14   4.38927675e-02
1.48488747e-02
    3.25240978e-02   2.62175445e-02   4.36185695e-02
2.63019131e-02
    5.33420514e-02   1.08413659e-02   4.35342009e-02
3.71643711e-02
    1.44270317e-02   7.42232815e-02   2.85165890e-02
2.83056675e-02
    1.04195229e-02   6.13781612e-03   6.37193900e-02
8.32507224e-02
    7.99392546e-02   4.30279893e-02   7.36959777e-02
7.50247833e-02
    1.39587859e-01  -8.69859740e-14   7.83362511e-02
2.02484655e-02
    1.07991816e-02   1.06304444e-02   1.52074413e-02
1.10733796e-02
    2.31169982e-02   2.17460083e-02   1.79283289e-02
4.17202759e-02
    6.38670351e-02   8.42420535e-02   4.93767269e-02
2.57535171e-02
    9.87112695e-03   2.22522199e-02   2.02062812e-02
1.07780895e-02
```

```
    9.16243066e-02   2.44458037e-02   1.92571344e-02
1.93625952e-02
    4.96509249e-02   3.71010947e-02   4.36818460e-02
1.21554070e-01
    3.13429373e-02   3.73120162e-02   8.56341355e-03
8.93674464e-02
    1.44692160e-02   1.26341988e-01   2.73354285e-02
1.33935163e-02
    3.07523570e-02   1.09468267e-02   1.02086014e-01]
 [  5.70320520e-05   9.14223794e-02   5.71461161e-02
3.07973081e-02
    2.28128208e-02   6.27922893e-02   2.75464811e-02
1.14007072e-01
    1.08360899e-01   5.99977187e-02   1.26611155e-02
4.02178291e-14
    3.92380518e-02   9.69544884e-03   7.48260522e-02
1.36306604e-02
    2.80027375e-02   1.71096156e-02   2.92574427e-02
5.32109045e-02
    4.14052698e-02   1.00000000e+00   3.06832440e-02
1.81361925e-02
    6.25071290e-02   6.35907380e-02   2.95996350e-02
1.06364777e-01
    2.11018592e-02   6.47884111e-02   3.85536672e-02
3.13676286e-02
    1.97330900e-02   1.56267823e-02   2.56644234e-02
4.39146801e-02
    1.46002053e-02   1.10642181e-02   6.12524239e-02
2.24135964e-02
    2.80027375e-02   7.15752253e-02   1.10071860e-02
5.89711418e-02
    1.48283335e-02   3.92950838e-02   2.49230067e-02
2.05885708e-02
    1.36876925e-02   8.55480780e-02   3.03980837e-02
9.46732063e-03
    2.92574427e-02   2.43526862e-02   3.06262119e-02
1.55697502e-02
    4.69944109e-02   8.55480780e-03   3.58731607e-02
3.01699555e-02
    8.78293601e-03   6.63853085e-02   1.76799361e-02
2.33261093e-02
    7.87042318e-03   5.81726931e-03   5.87430136e-02
7.30580586e-02
    7.33432189e-02   3.69567697e-02   6.47884111e-02
5.46937379e-02
    8.54910460e-02   8.37800844e-02   6.15946162e-02
9.23919243e-03
    7.81339113e-03   6.33055777e-03   1.18626668e-02
```

```
8.26964754e-03
    2.02463785e-02   1.30033079e-02   1.52845899e-02
3.86106992e-02
    5.61765712e-02   7.94456485e-02   4.03216608e-02
2.32120452e-02
    6.78681419e-03   1.75088400e-02   1.51134938e-02
1.08931219e-02
    7.87042318e-02   1.60830387e-02   1.39158207e-02
1.51705258e-02
    2.61206798e-02   2.77746093e-02   3.36489107e-02
1.03342078e-01
    2.39534618e-02   3.16527889e-02   6.33055777e-03
7.04345842e-02
    1.22048591e-02   7.81909433e-02   2.29839170e-02
9.23919243e-03
    2.39534618e-02   8.49777575e-03   9.07379948e-02]
 [  1.22978540e-04   9.75834717e-02   5.73694890e-02
2.81620857e-02
    2.50261329e-02   5.43565148e-02   2.12752875e-02
1.03424952e-01
    1.06191970e-01   5.60782144e-02   1.37735965e-02
4.73467380e-03
    4.24275964e-02   1.30357253e-02   7.15120212e-02
1.53723175e-02
    2.40423046e-02   1.71555064e-02   3.38805878e-02
5.56477895e-02
    3.40650556e-02   7.00977679e-03   2.98222960e-02
1.93076308e-02
    6.37028838e-02   5.47254504e-02   2.44112402e-02
1.06868351e-01
    2.24435836e-02   5.06056693e-02   3.66476050e-02
3.65246265e-02
    1.82623132e-02   1.74629527e-02   2.63174076e-02
4.31654676e-02
    1.21133862e-02   1.18674291e-02   8.22111542e-02
2.10293304e-02
    3.06216565e-02   7.63696735e-02   1.11295579e-02
6.58550083e-02
    1.31587038e-02   5.18354547e-02   3.24048454e-02
2.03529484e-02
    1.72784849e-02   4.23169157e-01   4.24890857e-02
7.74764804e-03
    2.61329398e-02   2.53335793e-02   3.89227080e-02
1.83852918e-02
    3.71395192e-02   8.79296563e-03   7.85832872e-02
3.11135707e-02
    1.22978540e-02   6.34569268e-02   2.27510299e-02
2.03529484e-02
```

```
    8.17807293e-03   4.30424891e-03   6.22271414e-02
5.99520384e-02
    7.75994589e-02   3.47414376e-02   6.29650126e-02
6.26575663e-02
    9.81368751e-02   6.52278177e-01   5.50328968e-02
1.11910472e-02
    9.03892271e-03   7.74764804e-03   1.10680686e-02
9.16190125e-03
    2.13982660e-02   1.78933776e-02   1.25438111e-02
2.99452745e-02
    5.24503474e-02   6.65928795e-02   3.67090943e-02
2.20746480e-02
    8.42403001e-03   1.63561459e-02   1.21748755e-02
7.68615877e-03
    8.42403001e-02   1.80163561e-02   1.61101888e-02
1.64176351e-02
    3.39420771e-02   3.20973990e-02   3.70780299e-02
1.08221115e-01
    2.12752875e-02   3.02527209e-02   6.21041628e-03
8.08583902e-02
    1.02072188e-02   8.90364631e-02   2.28740085e-02
1.28512575e-02
    2.84695321e-02   7.62466950e-03   7.82758409e-02]
 [  5.32141337e-05   8.58876117e-02   5.51830566e-02
3.54938272e-02
    1.76670924e-02   6.38037463e-02   1.87845892e-02
1.22552150e-01
    1.11696467e-01   6.31651767e-02   1.04831843e-02
1.00000000e+00
    3.75691784e-02   6.70498084e-03   6.04512559e-02
1.06960409e-02
    2.79906343e-02   5.00212857e-03   2.79374202e-02
4.49127288e-02
    4.03363133e-02   1.09674330e-01   1.80928054e-02
1.85717327e-02
    4.87973606e-02   6.77415922e-02   3.08109834e-02
1.29895700e-01
    2.41060026e-02   6.58790975e-02   4.36355896e-02
3.19816943e-02
    2.21902937e-02   5.74712644e-03   1.88910175e-02
4.31034483e-02
    1.29842486e-02   1.19199659e-02   4.34227331e-02
2.17645807e-02
    2.36802895e-02   7.50851426e-02   6.22605364e-03
5.39591315e-02
    1.29310345e-02   8.51426139e-03   1.78799489e-02
1.63367390e-02
    1.00042571e-02   1.10100043e-01   4.16134525e-02
```

```
8.03533418e-03
    2.68731375e-02   2.07002980e-02   4.53384419e-02
1.66560238e-02
    4.68816518e-02   8.35461899e-03   3.73563218e-02
3.03852703e-02
    1.19731801e-02   7.04555130e-02   1.81992337e-02
2.39463602e-02
    8.88676032e-03   6.01319711e-03   5.47041294e-02
7.87569178e-02
    8.50361856e-02   3.13431247e-02   7.43401447e-02
5.77373350e-02
    4.39548744e-02   3.85802469e-02   6.07173265e-02
4.84248616e-03
    7.66283525e-03   6.01319711e-03   1.25585355e-02
7.98212005e-03
    1.67624521e-02   9.57854406e-03   1.51128140e-02
3.72498936e-02
    5.90144742e-02   8.67922520e-02   4.56577267e-02
2.02213708e-02
    7.98212005e-03   2.01681567e-02   1.23456790e-02
8.62068966e-03
    8.07790549e-02   1.39953172e-02   1.12813963e-02
1.59642401e-02
    3.02788421e-02   3.13431247e-02   3.29395487e-02
1.22818221e-01
    2.19774372e-02   3.37909749e-02   6.27926777e-03
6.54533844e-02
    1.07492550e-02   6.28458919e-02   2.74584930e-02
1.12281822e-02
    2.74052788e-02   9.57854406e-03   1.08131120e-01]]
Cluster_label:
 [0 0 2 ..., 0 0 3]
Cluster_inertia:
 424893.569994
```

In [10]:

```
#Based off the results collected, each index corresponds with a Hero
ID. However, an array starts at 0 which offets
#every monster to the left. Additionally, the dataset not account fo
r hero ID: 24 but there is still a value at index.
#This means that every monster with an original hero id greater than
 23 is offset to the left by two.
#Therefore, the below method brings back the monster to its rightful
 index
def correctCluster(cluster):
    resultArr = []
    for num in cluster:
        resultArr.append(num+2) if num>23 else resultArr.append(num+
1)
    return resultArr
```

In [11]:

```
cluster1 = KM.cluster_centers_[0]
cluster2 = KM.cluster_centers_[1]
cluster3 = KM.cluster_centers_[2]
cluster4 = KM.cluster_centers_[3]

#This map's keys would be the cluster's name and each corresponding
 value would be a tuple of the top 5 hero names
clusterMapName = {}
#This map's keys would be the cluster's name and each corresponding
 value would be a tuple of the top 5 hero Ids
clusterMapId = {}

##Cluster 1

#The function below sorts the array index from lowest to highest bas
ed off the value of each elements
#This means that the first 5 index slots that has highest values wil
l be returned
#For example: given array=[5,1,4,8,0], the result would be [4,1,2,0,
3]
A = sorted(range(len(cluster1)), key=lambda i: cluster1[i])[-5:]
top5HerosId = correctCluster(A)
hero1 = searchHeroName(top5HerosId[0])
hero2 = searchHeroName(top5HerosId[1])
hero3 = searchHeroName(top5HerosId[2])
hero4 = searchHeroName(top5HerosId[3])
hero5 = searchHeroName(top5HerosId[4])
clusterMapId['Cluster1'] = (top5HerosId[0],top5HerosId[1],top5HerosI
d[2],top5HerosId[3],top5HerosId[4])
clusterMapName['Cluster1']=(hero1,hero2,hero3,hero4,hero5)

##Cluster 2
A = sorted(range(len(cluster2)), key=lambda i: cluster2[i])[-5:]
top5HerosId = correctCluster(A)
hero1 = searchHeroName(top5HerosId[0])
hero2 = searchHeroName(top5HerosId[1])
hero3 = searchHeroName(top5HerosId[2])
hero4 = searchHeroName(top5HerosId[3])
hero5 = searchHeroName(top5HerosId[4])
clusterMapId['Cluster2'] = (top5HerosId[0],top5HerosId[1],top5HerosI
d[2],top5HerosId[3],top5HerosId[4])
clusterMapName['Cluster2']=(hero1,hero2,hero3,hero4,hero5)

##Cluster 3
A = sorted(range(len(cluster3)), key=lambda i: cluster3[i])[-5:]
```

```
top5HerosId = correctCluster(A)
hero1 = searchHeroName(top5HerosId[0])
hero2 = searchHeroName(top5HerosId[1])
hero3 = searchHeroName(top5HerosId[2])
hero4 = searchHeroName(top5HerosId[3])
hero5 = searchHeroName(top5HerosId[4])
clusterMapId['Cluster3'] = (top5HerosId[0],top5HerosId[1],top5HerosI
d[2],top5HerosId[3],top5HerosId[4])
clusterMapName['Cluster3']=(hero1,hero2,hero3,hero4,hero5)

##Cluster 4
A = sorted(range(len(cluster4)), key=lambda i: cluster4[i])[-5:]
top5HerosId = correctCluster(A)
hero1 = searchHeroName(top5HerosId[0])
hero2 = searchHeroName(top5HerosId[1])
hero3 = searchHeroName(top5HerosId[2])
hero4 = searchHeroName(top5HerosId[3])
hero5 = searchHeroName(top5HerosId[4])
clusterMapId['Cluster4'] = (top5HerosId[0],top5HerosId[1],top5HerosI
d[2],top5HerosId[3],top5HerosId[4])
clusterMapName['Cluster4']=(hero1,hero2,hero3,hero4,hero5)
```

In [12]:

```
#Table of hero names for a given cluster
pd.DataFrame(clusterMapName)
```

Out[12]:

	Cluster1	Cluster2	Cluster3	Cluster4
0	Skywrath Mage	Skywrath Mage	Mirana	Mirana
1	Juggernaut	Tidehunter	Tidehunter	Juggernaut
2	Earth Spirit	Mirana	Skywrath Mage	Skywrath Mage
3	Invoker	Juggernaut	Clockwerk	Tidehunter
4	Venomancer	Zeus	Silencer	Phantom Lancer

In [13]:

```
#Table of hero ids for a given cluster
resultIds = pd.DataFrame(clusterMapId)
resultIds
```

Out[13]:

	Cluster1	Cluster2	Cluster3	Cluster4
0	101	101	9	9
1	8	29	29	8
2	107	9	101	101
3	74	8	51	29
4	40	22	75	12

In [14]:

```
#Given a hero id, we return the role that hero plays
def find_role(heroId):
    champ_row = champions.loc[champions["hero_id"] == heroId]
    role =  champ_row["role"]
    role = role.values
    return ''.join(role)

#In the table of hero Ids, we apply the above function into each cel
l
resultCategory = resultIds.applymap(lambda x: find_role(x))
#A table of what types of roles each cluster has
resultCategory
```

Out[14]:

	Cluster1	Cluster2	Cluster3	Cluster4
0	nuker	nuker	escape	escape
1	carry	initiator	initiator	carry
2	nuker	escape	nuker	nuker
3	nuker	carry	initiator	initiator
4	support	nuker	disabler	carry

References

1. https://www.polygon.com/2017/8/12/16136964/dota-2-ti7-winners-international
2. T. Nuangjumnonga and H. Mitomo, “Leadership development through online gaming,” in 19th ITS Biennial Conference: : Moving Forward with Future Technologies: Opening a Platform for All, (Bangkok), pp. 1–24, 2012.
3. N. Pobiedina and J. Neidhardt, “On successful team formation,” tech. rep., 2013.
4. F. Rioult, J.-P. M´etivier, B. Helleu, N. Scelles, and C. Durand, “Mining Tracks of Competitive Video Games,” AASRI Procedia, vol. 8, no. Secs, pp. 82–87, 2014.
5. A. Drachen, M. Yancey, J. Maguire, D. Chu, I. Y. Wang, T. Mahlmann, M. Schubert, and D. Klabajan, “Skill-based differences in spatio-temporal team behaviour in defence of the Ancients 2 (DotA 2),” Games Media Entertainment (GEM), 2014 IEEE, vol. 2, no. DotA 2, pp. 1–8, 2014.
6. P. Yang, B. Harrison, and D. L. Roberts, “Identifying Patterns in Combat that are Predictive of Success in MOBA Games,” in Proceedings of Foundations of Digital Games, (Miami, Florida), pp. 1–8, 2014.
7. M. Schubert, A. Drachen, and T. Mahlmann, “Esports Analytics Through Encounter Detection,” in MIT SLOAN Sports Analytics Conference, pp. 1 – 18, 2016.
8. L. Gao, J. Judd, D. Wong, and J. Lowder, “Classifying Dota 2 Hero Characters Based on Play Style and Performance,” 2013.
9. C. Eggert, M. Herrlich, J. Smeddinck, and R. Malaka, “Classification of Player Roles in the Team-Based Multi-player Game Dota 2,” in Entertainment Computing - ICEC 2015 (K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, and R. Malaka, eds.), vol. 9353 of Lecture Notes in Computer Science, (Cham), pp. 112– 125, Springer International Publishing, 2015.
10. K. Conley and D. Perry, “How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2,” tech. rep., 2013.
11. https://ieeexplore.ieee.org/abstract/document/7960061
12. https://ieeexplore.ieee.org/abstract/document/7828640