Analysis of DOTA 2 Game Data

by

Lam Tran ()

David Nguyen (dnguy260@calpoly.edu)

Luke Watts (luwatts@calpoly.edu)

Joey Trevino (jrtrevin@calpoly.edu)

Group Project Report

CSC 466-01 - Knowledge Discovery from Data

California Polytechnic State University – San Luis Obispo

December 10th, 2021

Fall 2021

TABLE OF CONTENTS

INTRODUCTION

DOTA 2 is a popular multiplayer online battle arena (MOBA) game originally launched by developer and distributor Valve on their platform Steam in 2013 with millions of players worldwide to this day. The game boasts the largest prize pool of any esports game with 2019's The International tournament having a total purse of $40,018,195. As a result of the large fanbase, active competitive scene, and rather complicated mechanics of the game, DOTA 2 has developed a community of statistics minded players who have developed open source API tools to retrieve data from matches for the purpose of analysis. Our base dataset was created using these tools, and our team expanded our dataset using these same API tools as well.

Each game of DOTA begins with a drafting stage in which both teams select one playable character (hero) at a time until ten unique heroes, five on each team, have been selected. An average game of DOTA is around 45 minutes and each game can be influenced by a myriad of ingame factors and mechanics. In this report, we are going to explain all steps, from prior, to initial, to final steps that we carried throughout the analytic process and provide supporting visualization to strengthen the claims.

Our team developed three main engaging questions that can be answered through different types of machine learning algorithms that were the subject matter of CSC 466, Knowledge Discovery from Data. The first question we developed was centered around the idea of classification with wondering if a team's victory could be predicted by the heroes selected on a team, abilities selected by players, or by overall player skill. In addition, we wondered if trends of certain heroes being commonly picked together or against one another could be determined through an Apriori algorithm using the concepts of association rules. Moreover, we pondered whether heroes could be grouped into specific clusters using ingame statistical analytics and if so which clustering algorithms would be the most effective at identifying heroes by position or roles. All of this information along with a more detailed analysis will be presented within our report.

ABSTRACT

In this paper, we explore the possible insights for the game DOTA 2. These dataset was obtained after mining in-game match data for over 40,000 matches in combinatinon with other popular DOTA datasets found on Kaggle. The insights we achieved include: determining if we can predict a match's outcome using different classification models, clustering a variety of heroes together using in-game statistics to visualize possible team compositions, and determining association rules for heroes that were commonly picked together inside matches. Overall, we produced results and insights that may help in increasing match outcomes if used in real world settings.

DATASET DESCRIPTION

Our dataset of choice was found via kaggle and contains information regarding 50,000 games of DOTA 2 played in 2018. Although expanded upon using the OpenDota API tools, the original dataset can be found at the following address : https://www.kaggle.com/devinanzelmo/dota-2-matches.

hero_names.csv - contains 3 columns: name, hero_id, localized_names. There are 112 champions in this game in total, thus giving 112 rows. Each hero_id is unique, which also maps to a unique name, as well as localized_names.

match.csv - There are 13 columns in this data set: Match_id, start_time, duration, tower_status_radiant, tower_status_dire, barracks_status_dire, barracks_status_radiant, first_blood_time, game_mode, radiant_win, negative_votes, positive_votes, cluster. However, we will only be using column match_id, and radiant_win to keep track of the Win/Loss for the radiant team for a specific match.

players.csv - 500,000 rows of player data with 73 attributes of varying in game statistics based around a player’s performance and decisions.The 500,000 rows were generated from the 50,000 matches from match.csv with each match having 10 players, and each row of player data can be mapped to match.csv using the match_id attribute.

Player_rating.csv - There are 5 columns to this dataset: account_id, total_wins, total_matches, trueskill_mu, and trueskill_sigma. We will only be looking at account_id and trueskill_mu, in which each account_id represents a unique player. The trueskill_mu is the player’s rating based on his pure skill, and is all collected based off of their past performances.

Analytical Questions

1. Given a team consisting of a selection of playable characters (referred to in game as heroes), can we determine if said team will win or lose using classification models?
2. Can we determine the association rules for heroes picked together and their counterpicks?
3. Using in-game statistics from player data, can heroes be clustered based upon their average performances to provide insight on their playstyles and team composition?

Question 1

Given a team consisting of a selection of playable characters (referred to in game as heroes), can we determine if said team will win or lose using classification models?

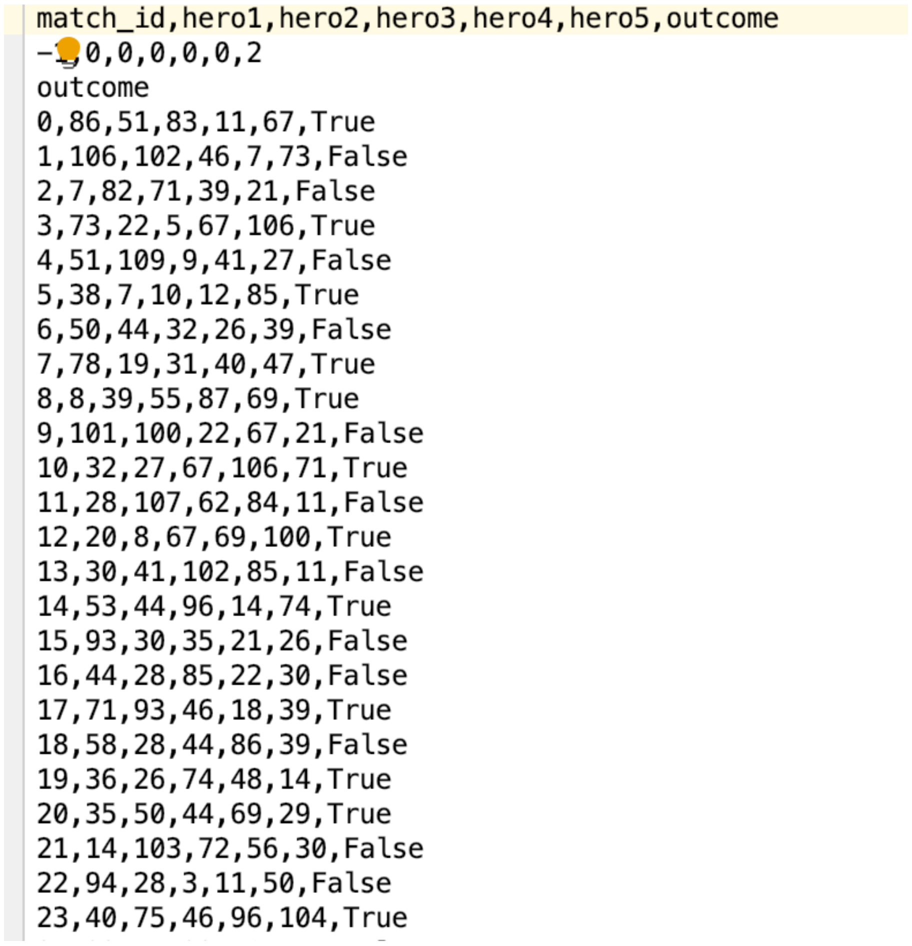
METHODOLOGY

C45 - The following method is using C45 algorithm in order to create a prediction tree for the match's outcome based purely on heroes selection amongst a team. The results were gathered via terminal outputs from the PyCharm console. We first must parse through the given datasets in order to create an input file that follows the format of 3 lines of overhead data containing the names of all columns which represent the given attributes and the class variable. The class variable is the final decision which will be represented as leaf nodes in the C4.5 algorithm, in which it would match the outcome. The second line tells the program some crucial information about each column such as if it represents an ID, if the given attribute is numeric, or the domain of the given categorical attribute. The third line tells you the name of the class variable so it can be isolated from row one. The rest of the file would then just be data points that are used to create the prediction tree, using the concept of entropy splitting. Once we have the C45 prediction tree, we classify them by running it directly back to the training dataset, and determine its accuracy

Analysis

Initial Approach: The initial approach to creating this input file is to have 5 independent variables: hero1, hero2, hero3, hero4, hero5. All of these columns would be the five selected heroes on that team in that game, which can be depicted in figure 1A. Toward the end, we would have 100+ different values for each attribute, which is a terrible idea since the generated tree is humongous. We tried classifying the data point, but classifying 50,000 matches with this tree takes an obnoxious amount of time. Therefore, we decided to redesign our process and take a different approach to predict match outcomes.

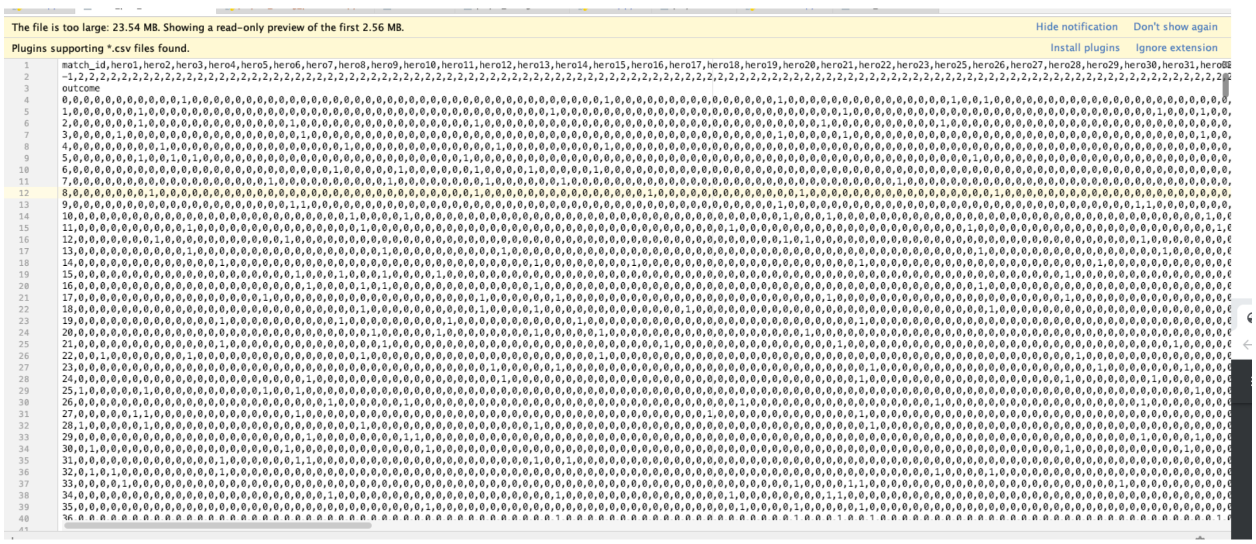
Figure 1A - Input1 for C45



```
match_id,hero1,hero2,hero3,hero4,hero5,outcome
-1,0,0,0,0,0,2
outcome
0,86,51,83,11,67,True
1,106,102,46,7,73,False
2,7,82,71,39,21,False
3,73,22,5,67,106,True
4,51,109,9,41,27,False
5,38,7,10,12,85,True
6,50,44,32,26,39,False
7,78,19,31,40,47,True
8,8,39,55,87,69,True
9,101,100,22,67,21,False
10,32,27,67,106,71,True
11,28,107,62,84,11,False
12,20,8,67,69,100,True
13,30,41,102,85,11,False
14,53,44,96,14,74,True
15,93,30,35,21,26,False
16,44,28,85,22,30,False
17,71,93,46,18,39,True
18,58,28,44,86,39,False
19,36,26,74,48,14,True
20,35,50,44,69,29,True
21,14,103,72,56,30,False
22,94,28,3,11,50,False
23,40,75,46,96,104,True
```

Modified Approach: In order to reduce the size of the prediction tree which will ultimately reduce the classifying run time, we must change the way our attributes are presented. We reapproached this by creating a singular attribute for each hero, thus resulting in 113 independent columns since there are 113 heroes in total. The data points can be seen as a sparse vector of 1s and 0s, in which 1 determines that hero is being picked. So on a team, there would be 5 heroes columns that have 1s and 108 heroes columns that are 0. This allows us to have a much smaller tree since the number of distinct values for each attribute now is just two, being 1 or 0. An example of this input file can be demonstrated in Figure 1B.

Figure 1B - Input2 for C45



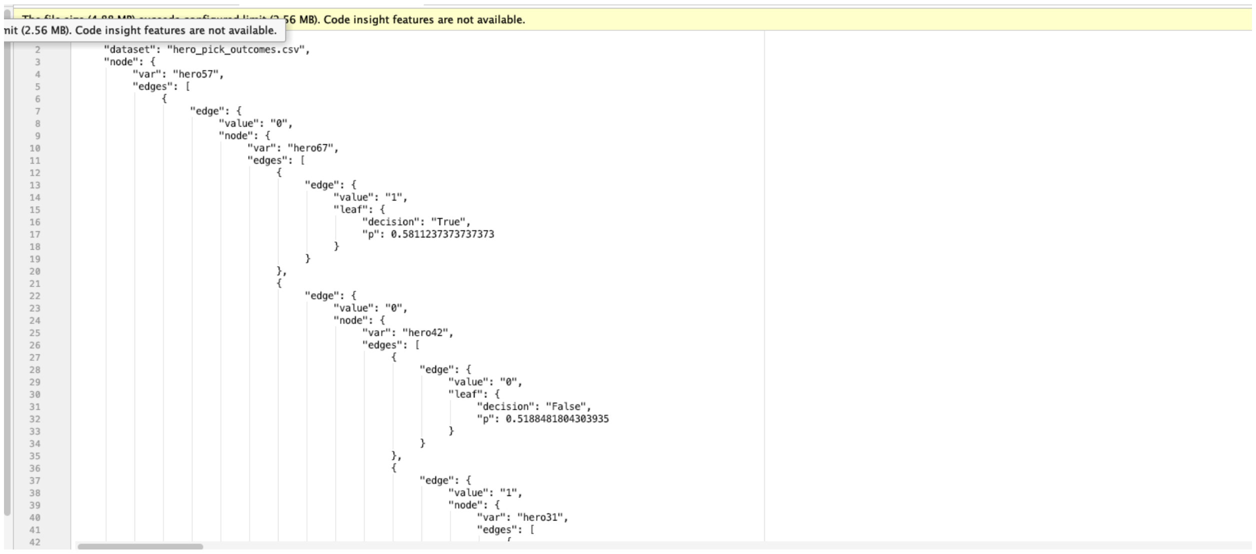
The file is too large: 23.54 MB. Showing a read-only preview of the first 2.56 MB.

Plugins supporting *.csv files found.

```
match_id,hero1,hero2,hero3,hero4,hero5,hero6,hero7,hero8,hero9,hero10,hero11,hero12,hero13,hero14,hero15,hero16,hero17,hero18,hero19,hero20,hero21,hero22,hero23,hero25,hero26,hero27,hero28,hero29,hero30,hero31,hero3
-1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
outcome
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,
2,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
3,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
4,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
5,0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
8,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
11,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0
12,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
13,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
14,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
17,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
19,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
21,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0
22,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
23,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0
24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0
25,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0
26,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
27,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
28,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
29,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0
30,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0
31,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
32,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
33,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
34,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
35,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
36,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
```

With this input file, we must still find a reasonable size tree, and accumulating the correct threshold is very tricky for this particular dataset because there are so many attributes to consider. In fact, the range of cut off for this is very fixed, meaning if we go beyond a value by a small percent, it would generate a null tree. The threshold we picked for this input file is .0054 because anything beyond .0055 creates a null tree. This took a long time to figure out because on previous C45 assignments, our threshold goes from .10 - .50. Part of the resulting tree is depicted in Figure 1C.

Figure 1C - Classification Tree from Input2



```
56 MB). Code insight features are not available.
nit (2.56 MB). Code insight features are not available.
    "dataset": "hero_pick_outcomes.csv",
    "node": {
        "var": "hero57",
        "edges": [
            {
                "edge": {
                    "value": "0",
                    "node": {
                        "var": "hero67",
                        "edges": [
                            {
                                "edge": {
                                    "value": "1",
                                    "leaf": {
                                        "decision": "True",
                                        "p": 0.5811237373737373
                                    }
                                }
                            },
                            {
                                "edge": {
                                    "value": "0",
                                    "node": {
                                        "var": "hero42",
                                        "edges": [
                                            {
                                                "edge": {
                                                    "value": "0",
                                                    "leaf": {
                                                        "decision": "False",
                                                        "p": 0.5188481804303935
                                                    }
                                                }
                                            },
                                            {
                                                "edge": {
                                                    "value": "1",
                                                    "node": {
                                                        "var": "hero31",
                                                        "edges": [
```

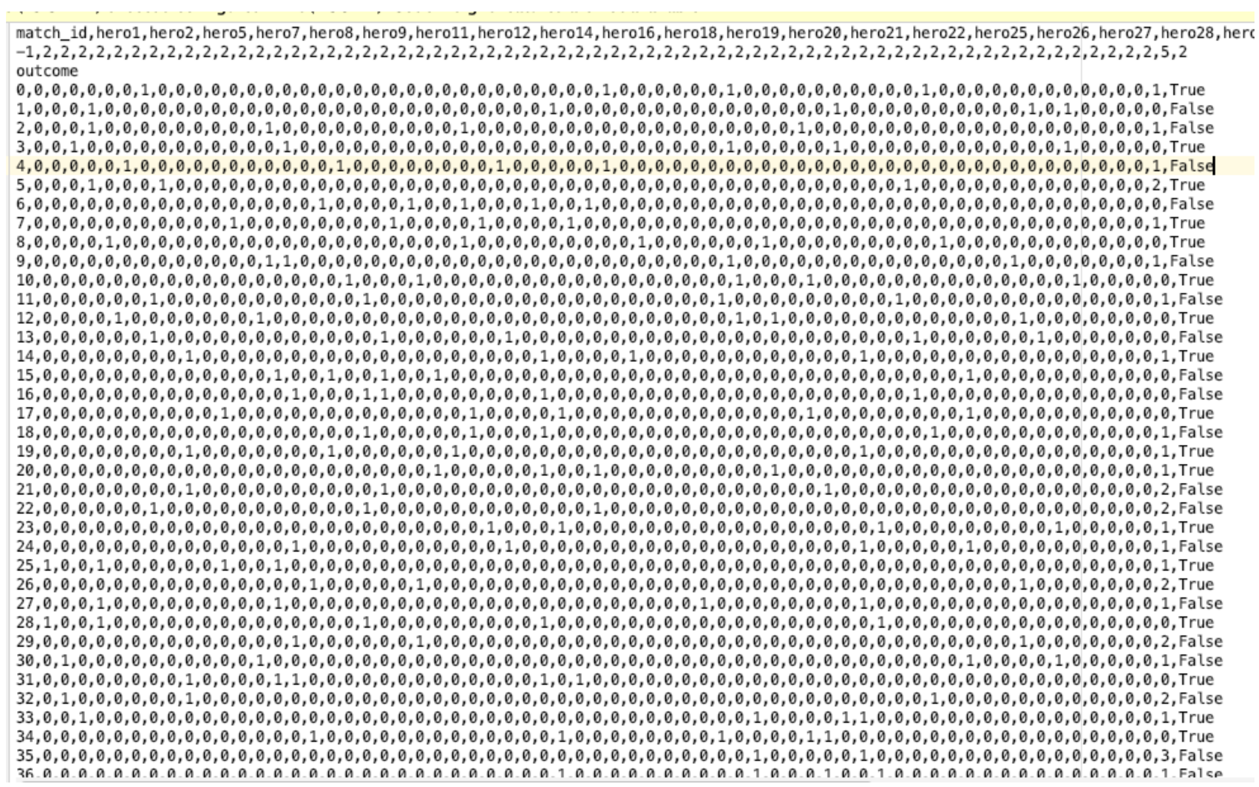
Classify 1C Result - With this tree, it still took a long time to run to predict all 50,000 matches because there were many attributes to consider when splitting. But we managed to get an outcome after about 90 minutes, which is depicted in Figure 1D.

Figure 1D - Classify Result From Classification Tree (1C)

```
Predicted  False   True     All
Actual
False      42366   7634   50000
True       39083  10917   50000
All        81449  18551  100000
Total Number Of Records Classified:  100000
Total Number Of Records Correctly Classified:  53283
Total number of records Incorrectly Classified:  46717
Overall accuracy and error rate of the classifier:  53.283 %
```

Final Approach: The final thought is always to ask how we can bring the runtime down, and of course the overall accuracy up. We also believe that the dataset could reset in overfitting. The solution to this is to reduce the amount of 113 independent variables down by categorizing low pick heroes as 1 category. FIrst, we computed the pick rate for each hero in the game and aggregated the low percentage ones to be in one category after finding what that low percentage cut off point is. In fact, this allows our data input to reduce from 113 independent variables to 64, which is almost half of what we originally have! Then instead of 1s and 0s, we would have 1s and 0s for the non low category ones, and 0-5 for the low_picked ones as in a match there could be up to all 5 picked characters being considered low category, and as low as 0. The input data would be depicted in Figure 1E.

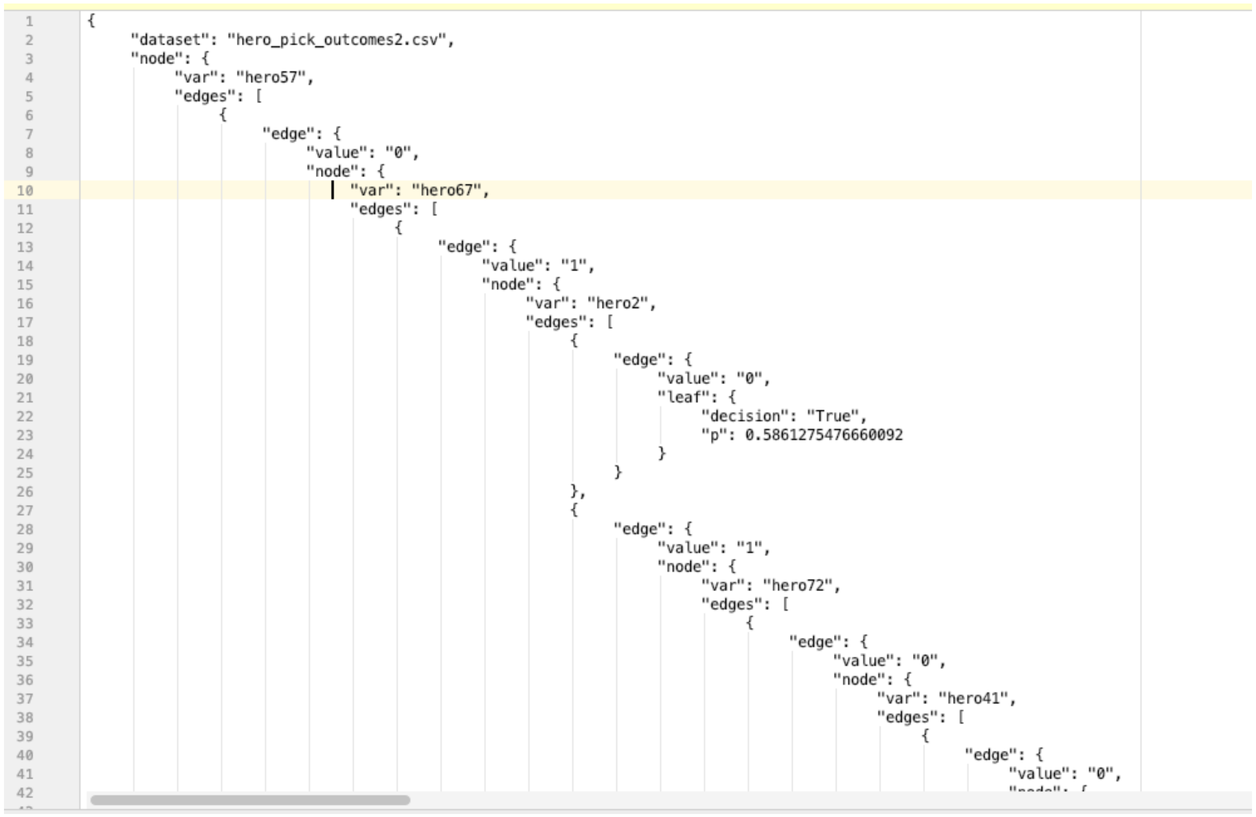
Figure 1E - Input3 for C45



```
match_id,hero1,hero2,hero5,hero7,hero8,hero9,hero11,hero12,hero14,hero16,hero18,hero19,hero20,hero21,hero22,hero25,hero26,hero27,hero28,herc
-1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,5,2
outcome
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,False
2,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
3,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,True
4,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
5,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,True
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,False
7,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
8,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,True
9,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,False
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,True
11,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
12,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,True
13,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,False
14,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,False
16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,False
17,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,True
18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,False
19,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
21,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,False
22,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,False
23,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,True
24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,False
25,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
26,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,2,True
27,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
28,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,True
29,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,2,False
30,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,False
31,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,True
32,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,False
33,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
34,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,True
35,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,False
36,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
```

With this input file, we had to still find the threshold as now it has changed. In fact, the range of this cut off is even smaller than the previous one. The threshold we picked for this input file is .001 because anything beyond .0012 creates a null tree. The final tree is in Figure 1F.

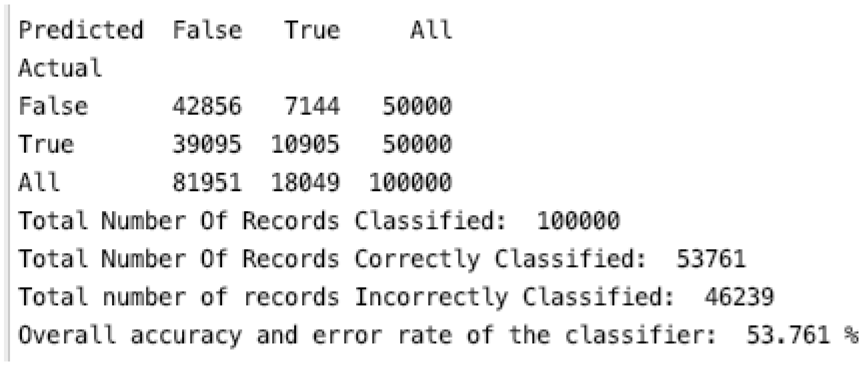
Figure 1F - Classification Tree from Input3



```
{
    "dataset": "hero_pick_outcomes2.csv",
    "node": {
        "var": "hero57",
        "edges": [
            {
                "edge": {
                    "value": "0",
                    "node": {
                        "var": "hero67",
                        "edges": [
                            {
                                "edge": {
                                    "value": "1",
                                    "node": {
                                        "var": "hero2",
                                        "edges": [
                                            {
                                                "edge": {
                                                    "value": "0",
                                                    "leaf": {
                                                        "decision": "True",
                                                        "p": 0.5861275476660092
                                                    }
                                                }
                                            },
                                            {
                                                "edge": {
                                                    "value": "1",
                                                    "node": {
                                                        "var": "hero72",
                                                        "edges": [
                                                            {
                                                                "edge": {
                                                                    "value": "0",
                                                                    "node": {
                                                                        "var": "hero41",
                                                                        "edges": [
                                                                            {
                                                                                "edge": {
                                                                                    "value": "0",
```

Classify 1F Result - With this tree, it took roughly 60 minutes, which is 30 minutes faster than the previous result, but the result only increased by a small factor. The result is depicted in Figure 1G.

Figure 1G - Classify Result From Classification Tree (1C)

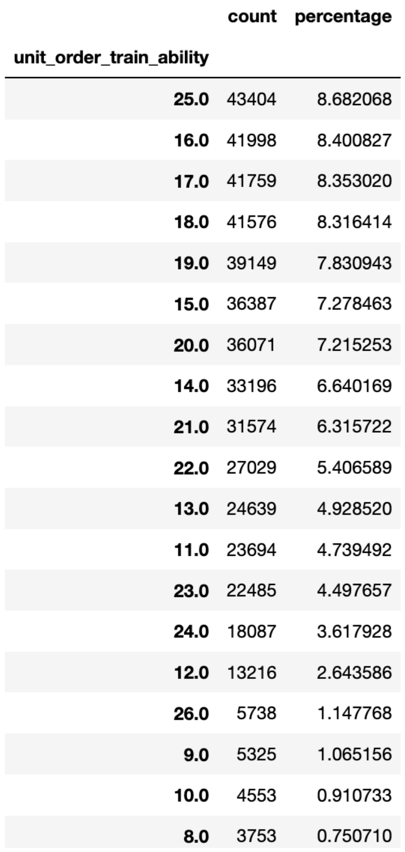

```
Predicted  False   True     All
Actual
False      42856   7144   50000
True       39095  10905   50000
All        81951  18049  100000
Total Number Of Records Classified:  100000
Total Number Of Records Correctly Classified:  53761
Total number of records Incorrectly Classified:  46239
Overall accuracy and error rate of the classifier:  53.761 %
```

Overall Analysis of C45 - The percentage may look pretty low since it's only around 53%, and if you were just to take a blind guess, it would already be 50/50. However, the game is designed to be that way. If you could win solely by picking it'd be a trash game. The highest win rate heroes alone are rarely above 55 percent, in which with this prediction accuracy, it's quite reasonable.

OTHER CLASSIFIERS

After taking the initial approach with C4.5, we explored using two other classifiers: random forest and k-nearest-neighbors. With this approach, however, we tried to determine if a player's skill would help predict match outcome. We used player abilities as a proxy for skillset with the logic that users who have more experience with the game are more likely to pick specific skills for a given hero. This introduced the same issue we experienced with C4.5, however, considering there are over 100 abilities in the game to choose from. To counteract this, we decided to group abilities in a drastic manner to go from 112 abilities to 5 groupings overall, iteratively named.

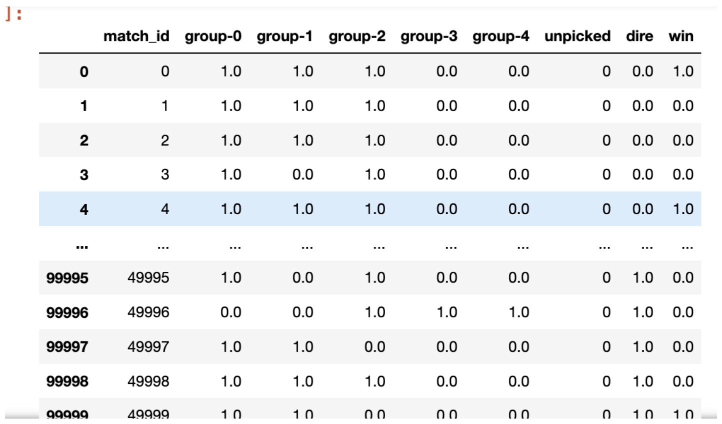
Methods - First, we aggregated the abilities and counted how many players used each ability specifically. From there, we made groupings containing a selection of abilities chosen, by popularity. **Figure 1H** shows a sample of abilities and their counts. For example, group 1 contained abilities with a pick-percentage of over 7%, and group 2 would contain abilities with pick-percentages from 5% to 7% (exclusive), and so forth.



	count	percentage
unit_order_train_ability		
25.0	43404	8.682068
16.0	41998	8.400827
17.0	41759	8.353020
18.0	41576	8.316414
19.0	39149	7.830943
15.0	36387	7.278463
20.0	36071	7.215253
14.0	33196	6.640169
21.0	31574	6.315722
22.0	27029	5.406589
13.0	24639	4.928520
11.0	23694	4.739492
23.0	22485	4.497657
24.0	18087	3.617928
12.0	13216	2.643586
26.0	5738	1.147768
9.0	5325	1.065156
10.0	4553	0.910733
8.0	3753	0.750710

Figure 1H - Abilities and their Pick-Percentages

This allowed us to reduce the ability set drastically to get the dataset in **Figure 1I.** With that dataset, we ran the random forest classifier and k-neighbors classifier using scikit learn's classifier methods.

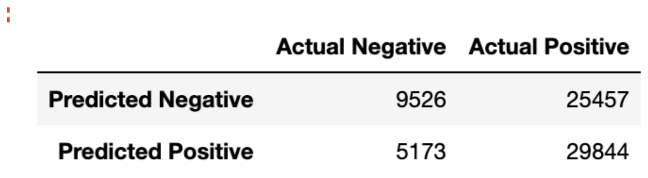


]:

	match_id	group-0	group-1	group-2	group-3	group-4	unpicked	dire	win
0	0	1.0	1.0	1.0	0.0	0.0	0	0.0	1.0
1	1	1.0	1.0	1.0	0.0	0.0	0	0.0	0.0
2	2	1.0	1.0	1.0	0.0	0.0	0	0.0	0.0
3	3	1.0	0.0	1.0	0.0	0.0	0	0.0	0.0
4	4	1.0	1.0	1.0	0.0	0.0	0	0.0	1.0
...	...	...	...	...	...	...	...	...	...
99995	49995	1.0	0.0	1.0	0.0	0.0	0	1.0	0.0
99996	49996	0.0	0.0	1.0	1.0	1.0	0	1.0	0.0
99997	49997	1.0	1.0	0.0	0.0	0.0	0	1.0	0.0
99998	49998	1.0	1.0	1.0	0.0	0.0	0	1.0	0.0
99999	49999	1.0	1.0	0.0	0.0	0.0	0	1.0	1.0

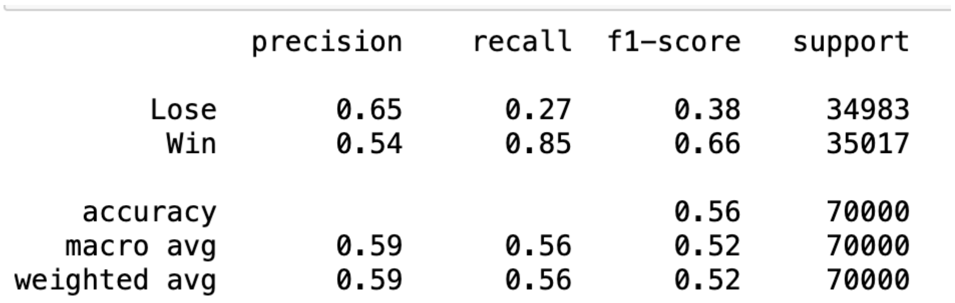
Figure 1I - A densified dataset for classification

Random Forest - With the densified dataset shown in Figure 1I, we set off to generate a random forest with a training set size of 70%. Interestingly, we found the results were somewhat similar to the C4.5 results with hero picks. Metrics are below for our classifier.



:

	Actual Negative	Actual Positive
Predicted Negative	9526	25457
Predicted Positive	5173	29844



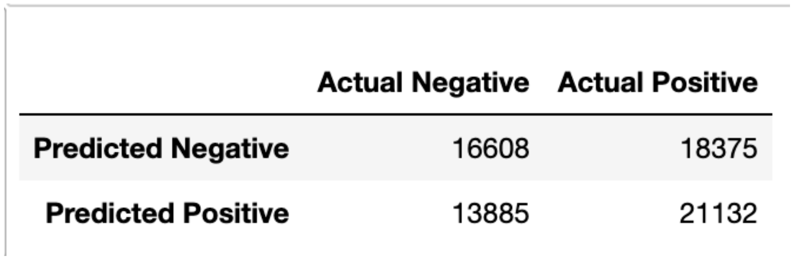
```
              precision    recall  f1-score   support

        Lose       0.65      0.27      0.38     34983
         Win       0.54      0.85      0.66     35017

    accuracy                           0.56     70000
   macro avg       0.59      0.56      0.52     70000
weighted avg       0.59      0.56      0.52     70000
```

One interesting aspect about this classifier's metrics is the recall score of a win. This classifier did extremely well at determining the sensitivity of the dataset. In other words, most of the actual "wins" were correctly predicted. Losses, on the other hand, had a higher precision overall but a much lower recall. It seems that this classifier does poorly when determining if a team will lose simply on abilities alone.

K-Neighbors - We also ran this classifier using a euclidean distance metric on the sparse vector. Results were on par with our decision tree classifier and metrics are below.



	Actual Negative	Actual Positive
Predicted Negative	16608	18375
Predicted Positive	13885	21132

```
              precision    recall  f1-score   support

        Lose       0.54      0.47      0.51     34983
         Win       0.53      0.60      0.57     35017

    accuracy                           0.54     70000
   macro avg       0.54      0.54      0.54     70000
weighted avg       0.54      0.54      0.54     70000
```

While this classifier performed with similar precision, we notice that the recall of a loss classification is much higher than our forest classifier. Unfortunately, this comes with a reduced recall score of a win classification. Overall, it seems that this classifier is much more balanced at predicting a win and a loss, whereas our random forest seems to favor a win prediction on abilities chosen alone.

The f1-scores seem to be balanced out as well, but our training test contained a consistent split of win and loss data. This is simply due to the nature of the dataset: we can generate a line for the team that won, and a line for the team that lost, so we have an even distribution of data. Thus, it is fairly safe to look at our overall precision of the classifiers to make a final judgement call: it seems that the random forest is the better classifier of the two, but both have their strengths and weaknesses.

Going Further - It’s an interesting concept to determine if a team will win a game solely on abilities or heroes chosen. We think that having a precision of roughly 56% in our classifiers is fair due to the nature of the game, however it brings to light the issues of densifying our dataset. A biased representation of the dataset may have been generated by grouping highly chosen attributes together, instead of some other metric. Essentially, most picked abilities will be in group 1 which may skew our classifier’s predictions. Another experiment should carefully consider how to group these abilities together. One may want to determine which attributes are most similar (for instance, group damaging abilities together, and supporting abilities with each other) when densifying the dataset then place these together. This form of dataset may be useful because it would signify if a team is “balanced”, “damage prone”, or “support prone”. If a team consists mainly of damage abilities, that could affect a team’s likelihood to win. Simply grouping these attributes by pick percentages may have influenced our classifier’s results dramatically.

Question 2

Can we determine the association rules for heroes picked together and their counterpicks?

Background & Context

From a high-level overview, the Apriori algorithm works by first finding frequent singleton itemsets and slowly building these singleton itemsets into larger itemsets. These large itemsets are only created if these large itemsets appear often enough in the dataset. The algorithm stops when there are no more itemsets to be created. In order to determine what itemset is considered frequent, we compute statistics such as support, confidence, and in some implementations, the level. When we are generating these large itemsets, we constantly measure their support values and when we input a support threshold, we effectively prune the search space of the algorithm which allows it to eventually end and reduces the need to exhaustly explore all possible itemsets. After exploring all itemsets, we can generate the association rules by computing the confidence of an itemset; given a confidence threshold, we can return a list of association rules that are above some certain threshold and this list of association rules is the output of the apriori algorithm.

When we apply this in context to our project, we specifically focus on the game Dota, a multiplayer game where two teams of size 5 compete against each other. With a total game size of 10 players, we can extract some key insight in the context of Dota. In Dota, there are a set of 113 playable heroes and these heroes have certain skills and abilities that invariably complement other heroes. For example, one hero could be a support hero to a hero that mainly deals damage to other enemy heroes. What we are interested in is whether we can determine association rules for these heroes. Given the entire dataset of matches along with their respective heroes inside these matches, we want to find association rules for team compositions for both of the teams in a match. By generating association rules for team compositions, we can evaluate how often and the probability a certain hero will be picked along with another set of heroes.

Methodology & Implementation

When approaching the implementation to this problem of determining market basket data analysis for heroes in Dota, we relied on using an external Python library, efficient-apriori, that contains the apriori algorithm for market basket data analysis. A significant amount of development time was spent mining in-game data using API calls from the website, OpenDota.com. After mining our data, roughly over 40,000 in-game matches, a majority of development time was spent cleaning, modifying, and coercing the data into a usable Pandas data frame. Once we finished identifying the necessary attributes and modifying the data for the appropriate setup for the apriori algorithm, we ran the data frame under the Apriori library to produce the association rules for heroes and their team compositions. After coercing our data to usable forms for use with Apriori, we ran this algorithm testing a variety of parameters. When changing different parameters, we varied the minimum support and the minimum confidence for each rule produced. After determining the appropriate thresholds and parameters, proceed to translating these results into a human-readable format. With the association rules for heroes (in the form of hero id's), we translated these association rules to their respective in-game hero names.

Our market basket data analysis methodology was fairly straight-forward, although, a different set of issues were encountered in our initial attempt to run our Apriori algorithm. Hardware limitation issues as well as development environment issues were the main culprits to our analysis. When we were mining matches, we mined match data as well as a number of other columns that may be useful for other questions. What we ended up with was a massive sparse matrix with an extensive amount of columns in our dataframe. Reading multiple datasets to produce a united dataset requires massive amount of RAM. The solution was to use online development environments such as Google Colab Pro that had the appropriate hardware necessary to run create the matrix. At the same time we ran into hardware limitations, development environments were artificially slowing down development due to the limitations imposed. Uploading multiple large datasets, disk quota limits, and configuring our notebook to enable zip and unzip files were the main headaches encountered during development. Jupyter Notebooks in combination with Calpoly's unix servers created more issues due to the read and write operations limitations imposed for each user. In addition, there were permission settings that continually kept occurring which led to us moving our development environment to Google Colab Pro.

Analysis of Results

When evaluating which parameters to use, we first start by plotting the minimum support against the number of rules created. Holding minimum confidence level constant at 0.01 to generate an unfiltered number of association rules, the figure below shows how we ranged test multiple support values ranging from [0.01, 0.15). What we see is a dropoff of zero rules generated for a minimum support value of 0.058. Initially, it was confusing to have such a low support value, but since there are a large number of matches in our dataset, it's appropriate to have a low support value since the number of occurrences for a set of 113 heroes would be fairly low. This combined with the fact that there are 6.208856635 E+13 possible hero combinations in a sample of 10 heroes per basket with only 40,000 matches in our dataset means that our apriori analysis can only cover a limited number of combinations leading to a low support value.

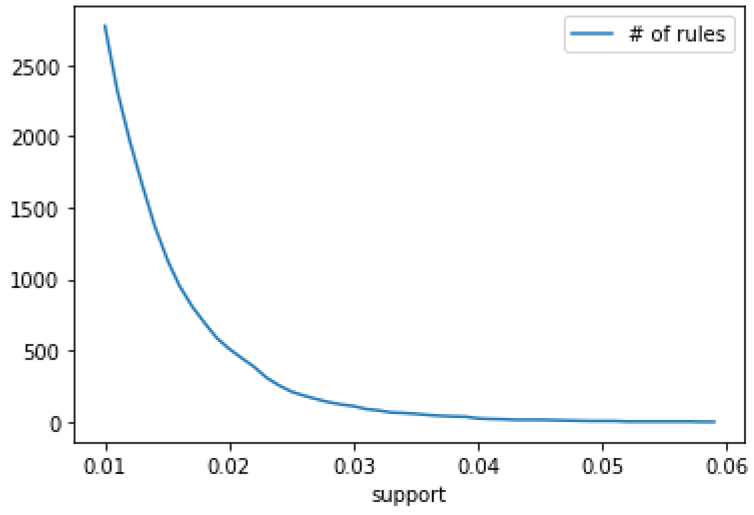


Figure 2A: Minimum Support Value (x) vs Number of Rules Generated (y) on a minimum confidence value of 0.01

Moving forward, evaluating the graph leads us to choosing a minimum support value of 0.02 since this is where the graph appears to start tapering off in the number of rules generated. In the figure below, we see the plot of the number of rules generated against the changing minimum confidence values ranging from [0.05, 0.3). Before a minimum support value of 0.05, there are no changes in the number of rules generated. Choosing a minimum confidence value here is difficult and lies affects reproducibility in the sense that there are no clear steep drop offs that suggest a clear minimum confidence value to choose. Because ther are no changes at ranges [0, 0.05] and [0.3, 1], we can choose the median confidence value between the range [0.05, 0.3]. Therefore, we proceed with a confidence value of 0.15. From our plotting analysis, we choose our parameters to be 0.02 for minimum support value and 0.15 for our minimum confidence value.

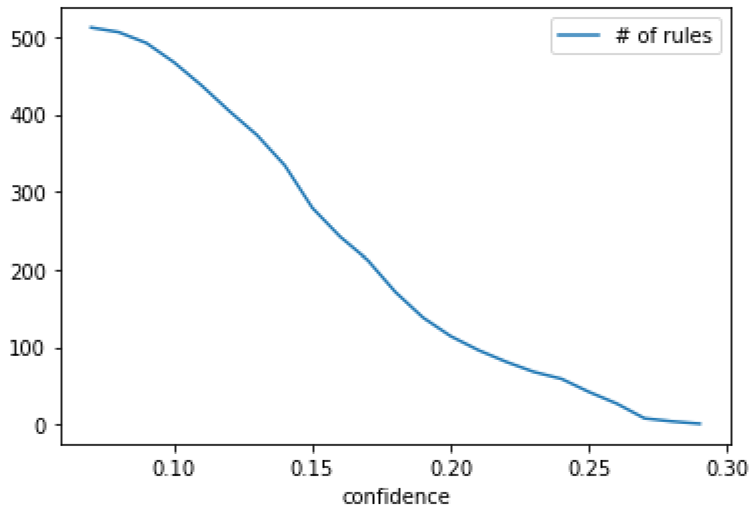


Figure 2B: Minimum Confidence Value(x) vs Number of Rules Generated (y) on a minimum support value of 0.2

Proceeding forward with our support and confidence values that we found to be fairly appropriate, we ran our apriori algorithm to generate the number of rules mapping hero id's to other hero id's. After doing so, we generated mapped these heroes to their respective in-game names. Referring to the appendix at the very bottom, we can clearly observe that with our given threshold parameters, we generate only singleton set associations. This may due to the rather limited number of matches our dataset contains in combination with the large number of possible hero combinations in a set of size 10 causing us to generate association rules of singleton sets.

Evaluating these association rules, we can look at a single concrete example to determine if this association rule is appropriate and makes sense when applied to Dota. Focusing on the hero, Anti-Mage in the figure below, we can observe a few peculiarities that heavily affect the association rules generated. When we apply domain knowledge to these association rules, there are a variety of interesting insights that we can evaluate. Before we dive further into our analysis, the association rules from left-hand side to right-hand side show that hero on the right-hand side synergizes or often is picked alongside the left-hand side hero. We see that the hero Ogre Magi is often picked with the hero Anti-Mage. This makes sense since Ogre Magi's support abilities synergize well with Anti-Mage's abilities and thus they are often picked together. But when we look at the association rule from Anti Mage to Invoker, these champions do not have any outstanding synergies that complement each other. We know this association rule exists because Invoker is popular hero picked for a different position. This shows that our results are heavily skewed by the popularity of the champion in matches and this affects our association rules generated. Realizing this insight, when we apply further domain knowledge, we realize that these are commonly picked heroes in Dota matches. In order to identify heroes that complement a Hero, we would need pre-existing domain knowledge to filter out which association rules occur because of the pure popularity of a hero against the associationrules that occur because two heroes are often picked together due to high synergy between heroes.

Looking at a concrete example, we saw interesting observations that arise from running Apriori on our dataset. This aligns with the nature of Apriori since Apriori looks at the frequency of itemsets within matches. Popular champions that are picked regardless of synergies with other champions appear often in the list of association rules. What's required is to apply domain knowledge to this list of association rules for a particular hero in order to determine which association rules occur because of complementing hero synergy versus the association rules that occur solely due to popularity of a hero for a certain lane or position in the match.

```
npc_dota_hero_antimage -> npc_dota_hero_nevermore
npc_dota_hero_antimage -> npc_dota_hero_pudge
npc_dota_hero_antimage -> npc_dota_hero_windrunner
npc_dota_hero_lion -> npc_dota_hero_antimage
npc_dota_hero_antimage -> npc_dota_hero_lion
npc_dota_hero_antimage -> npc_dota_hero_sniper
npc_dota_hero_antimage -> npc_dota_hero_invoker
npc_dota_hero_antimage -> npc_dota_hero_silencer
npc_dota_hero_antimage -> npc_dota_hero_ogre_magi
npc_dota_hero_antimage -> npc_dota_hero_legion_commander
```

Figure 2C: Association Rules for Anti-Mage

Question 3

Using in-game statistics from player data, can heroes be clustered based upon their average performances to provide insight on their playstyles and team composition?

Methodology

Three different approaches to clustering were used to investigate the above analytical question. These include:

1. K-Means (Using k++ initialization and varying values of k)
2. Density Based (Using DBScan and OPTICS)
3. Hierarchical (Using Agglomerative with Ward Linkage)

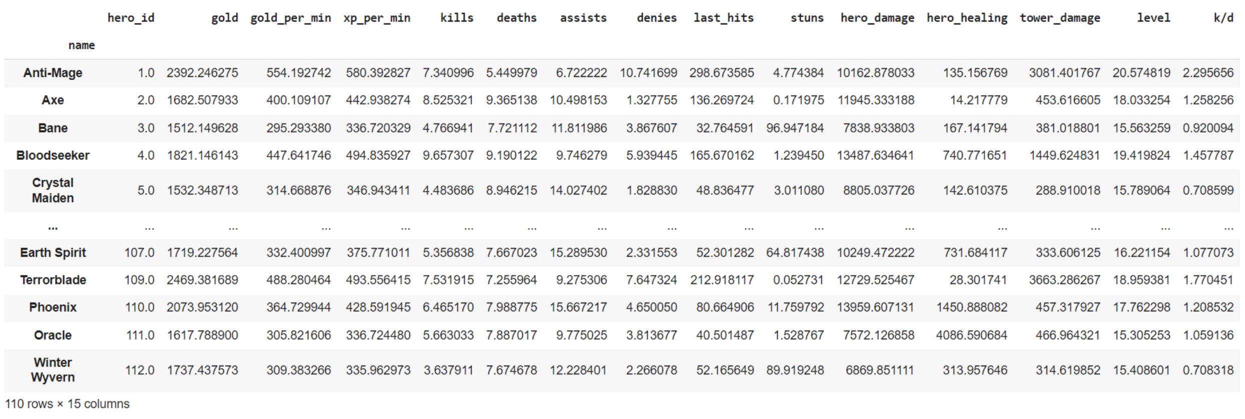
ScikitLearn's clustering library was used to run the aforementioned clustering algorithms. For each attempt at clustering, data from the players.csv file would be trimmed down to 14 attributes. These attributes are as follows:

1. 'hero_id' - the unique ID of a hero which can be mapped to its in-game and localized names using the mapping file hero_names.csv
2. 'gold' - the total gold earned, used for purchasing ingame items, by a player during a match
3. 'gold_per_min' - the gold earned per minute by a player during a match
4. 'xp_per_min' the experience points, used for leveling, earned per minute by a player during a match
5. 'kills' - the number of kills a player achieved in a match
6. 'deaths' - the number of deaths for a player in a match

7. 'assists' - the number of assists, earned by participating in team fights, by a player in a match
8. 'denies' - the number of denies, a method for earning experience points and denying the enemy of gold, achieved by a player during a match
9. 'last_hits' - the number of last hits, a method for obtaining gold, achieved by a player during a match
10. 'stuns' - the total time enemies were stunned, an in game status effect used to disrupt enemy play, by a player during a match
11. 'hero_damage' - the total damage dealt to enemy heroes by a player during a match
12. 'hero_healing' - the total healing dealt to allies by a player during a match
13. 'tower_damage' - the total damage dealt to enemy buildings during the course of a game
14. 'level' - the player level at the end of the game

The 500,000 rows of the players.csv file now with 14 attributes would be grouped by hero_id. Next, a 15th column ‘k/d’ was created to represent the kill to death ratio for a player. To prevent infinite values, games where a player had zero deaths were treated the same as if they had one death, preventing division by zero. Finally, the average performance of each hero would be found by taking the mean of each column in each hero group. Different approaches to clustering would use different combinations of these 14, all columns except hero ID, attributes.

Figure 3A - The DataFrame Containing Mean Hero Performance



name	hero_id	gold	gold_per_min	xp_per_min	kills	deaths	assists	denies	last_hits	stuns	hero_damage	hero_healing	tower_damage	level	k/d
Anti-Mage	1.0	2392.246275	554.192742	580.392827	7.340996	5.449979	6.722222	10.741699	298.673585	4.774384	10162.878033	135.156769	3081.401767	20.574819	2.295656
Axe	2.0	1682.507933	400.109107	442.938274	8.525321	9.365138	10.498153	1.327755	136.269724	0.171975	11945.333188	14.217779	453.616605	18.033254	1.258256
Bane	3.0	1512.149628	295.293380	336.720329	4.766941	7.721112	11.811986	3.867607	32.764591	96.947184	7838.933803	167.141794	381.018801	15.563259	0.920094
Bloodseeker	4.0	1821.146143	447.641746	494.835927	9.657307	9.190122	9.746279	5.939445	165.670162	1.239450	13487.634641	740.771651	1449.624831	19.419824	1.457787
Crystal Maiden	5.0	1532.348713	314.668876	346.943411	4.483686	8.946215	14.027402	1.828830	48.836477	3.011080	8805.037726	142.610375	288.910018	15.789064	0.708599
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Earth Spirit	107.0	1719.227564	332.400997	375.771011	5.356838	7.667023	15.289530	2.331553	52.301282	64.817438	10249.472222	731.684117	333.606125	16.221154	1.077073
Terrorblade	109.0	2469.381689	488.280464	493.556415	7.531915	7.255964	9.275306	7.647324	212.918117	0.052731	12729.525467	28.301741	3663.286267	18.959381	1.770451
Phoenix	110.0	2073.953120	364.729944	428.591945	6.465170	7.988775	15.667217	4.650050	80.664906	11.759792	13959.607131	1450.888082	457.317927	17.762298	1.208532
Oracle	111.0	1617.788900	305.821606	336.724480	5.663033	7.887017	9.775025	3.813677	40.501487	1.528767	7572.126858	4086.590684	466.964321	15.305253	1.059136
Winter Wyvern	112.0	1737.437573	309.383266	335.962973	3.637911	7.674678	12.228401	2.266078	52.165649	89.919248	6869.851111	313.957646	314.619852	15.408601	0.708318

110 rows × 15 columns

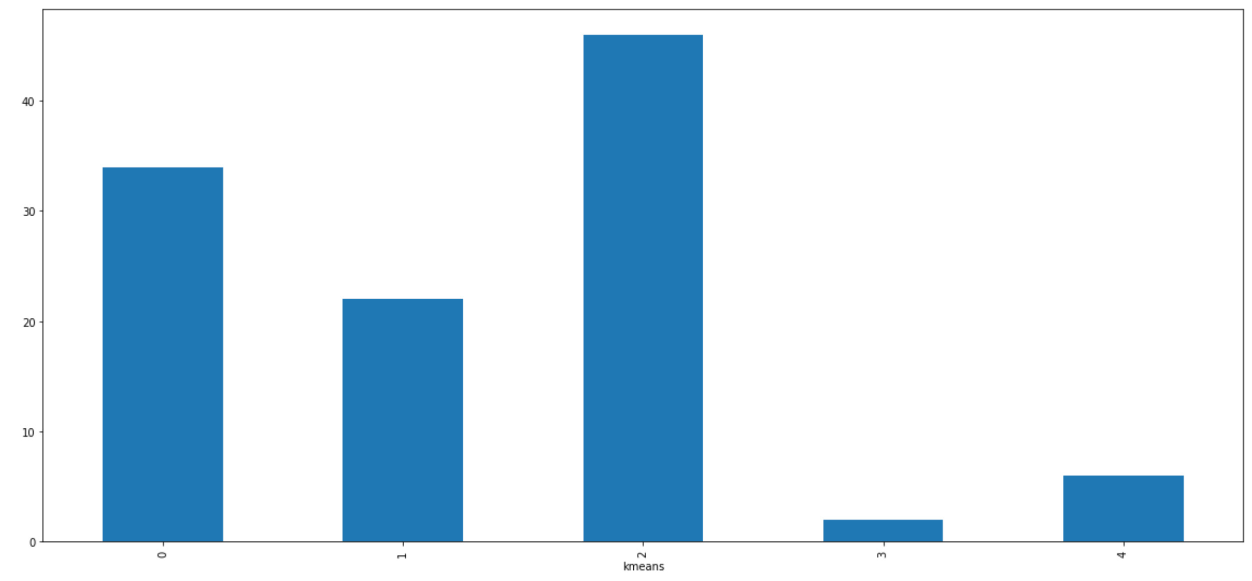
K-Means Clustering - The K-Means clustering approach was chosen as the first method of clustering as our initial belief was that the heroes could be clustered into five groups representing the positions 1 to 5 often referenced in DOTA 2 matches by players. The prediction of these groups are as follows:

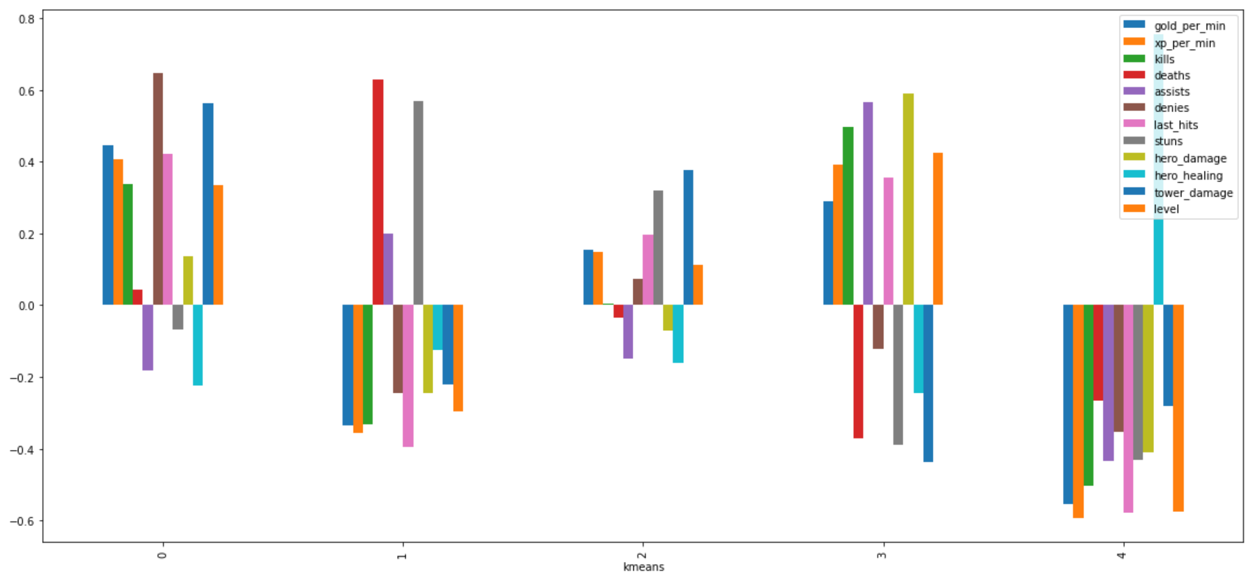
1. Hard Carry/Core - A resource dependent hero that reaches its apex later in games to carry a team to victory.
2. Early Carry/Core - A hero that requires gold and experience early to be effective, but can become strong if these early needs are met.
3. Initiator - A hero who is relatively strong early without resources and can initiate fights with enemies early to disrupt their play.
4. Soft Support - A hero that will travel around the map early to protect its teammates and support in any early team fights.
5. Hard Support - A hero whose main role is to keep their position 1 carry alive at all costs to ensure that said carry can achieve an income of resources at a reliable pace.

It is worth noting that although this scale uses 5 categorical positions, heroes commonly played at one position may play up or down a position depending on a player’s preferences, the composition of the teams, or new “meta” trends popular within the community at the time. It is not uncommon, for example, for a standard position 2 to play the position 1 or a position 4 to play the position 5. In some games, teams may also change their assigned positions at intervals in the game depending on how the early stages of resource acquisition occurred, although in-game changes are more common in public matches (which our players dataset uses) than professional tournament matches.

Using the 14 attribute columns from the hero performance dataframe (Figure 3A) as input, k-means clustering was ran with k++ initialization and k = 5 initially without normalization occurring before the clustering to see the results of the following output.

Figure 3B - Output for Kmeans(k=5) with Normalization Applied After Clustering



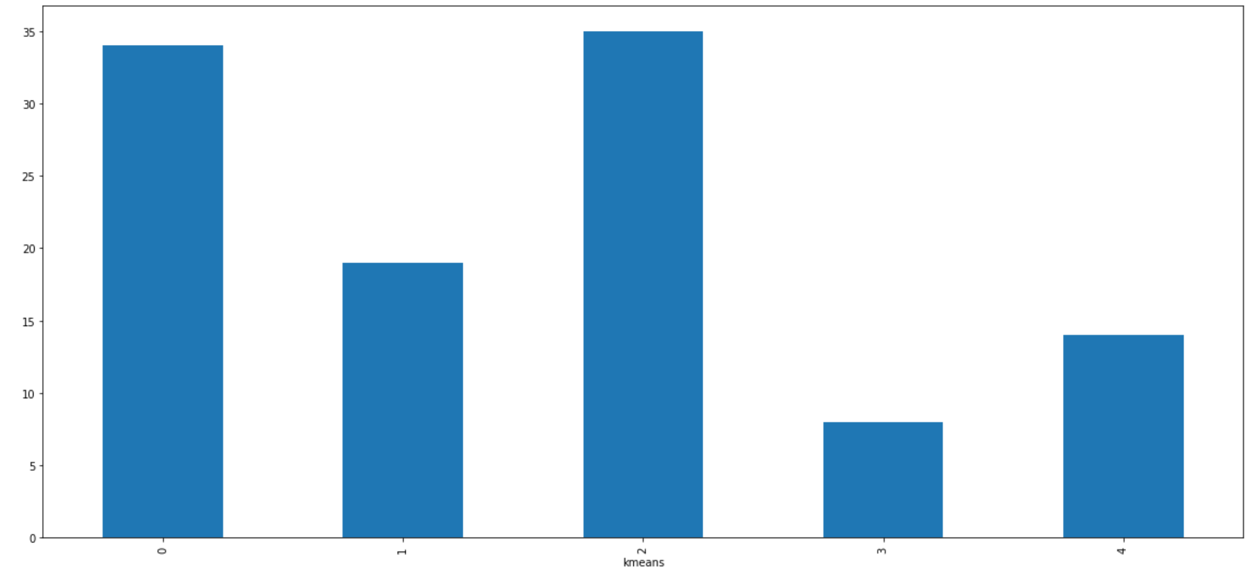


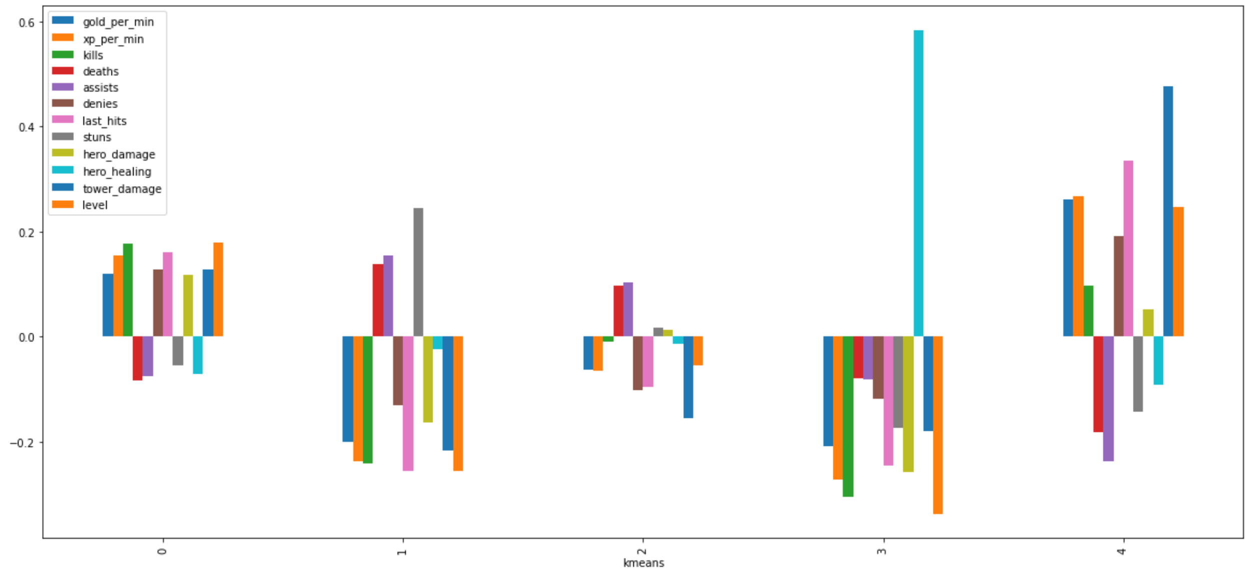
Without normalizing our data, the KMeans algorithm appears to do a fair job of identifying unique clusters based upon gameplay attributes. The algorithm is most successful in identifying position 1 heroes (cluster 0 in Figure 3B) as having high resource intake and output damage against buildings. Cluster 4 of the above figure contains a homogenous group of heroes who have healing output well above the mean of normalized values. Clusters 1 and 3 were inferred to be representative of positions 4 and 2 respectively as the former contained resources selfless heroes who take risks in fights as inferred by the high deaths and stun times and the latter was fairly resource dependant with the highest average player damage, meaning these heroes had access to high damage abilities earlier in the game.

Despite these apparent successes, this clustering representation had clear flaws. First, the distribution of heroes per cluster was skewed heavily with cluster 2 having nearly half of the ingame heroes and cluster 3 only having two heroes, Tinker and Zeus, who although are very similar are not the only representations of a position 2 hero. Additionally, cluster 2 which was most similar to position 3 heroes with mild resource dependence and decent stun duration and tower damage lacked other defining attributes indicative of an active fighting hero such as hero damage or assists. This is likely attributed to the large grouping heroes which formed the cluster. This KMeans based approach also had the inherent flaw of not having a means to detect outliers as not all heroes may fall into the position 1 to 5 scale.

To improve upon this model, our team decided to modify our approach by normalizing our attributes before running our clustering algorithm. For normalization, we used a z-score approach where each value was subtracted by the attribute mean and divided by the attribute standard deviation. The results are as follows:

Figure 3C - Output for Kmeans(k=5) with Normalization Applied Before Clustering





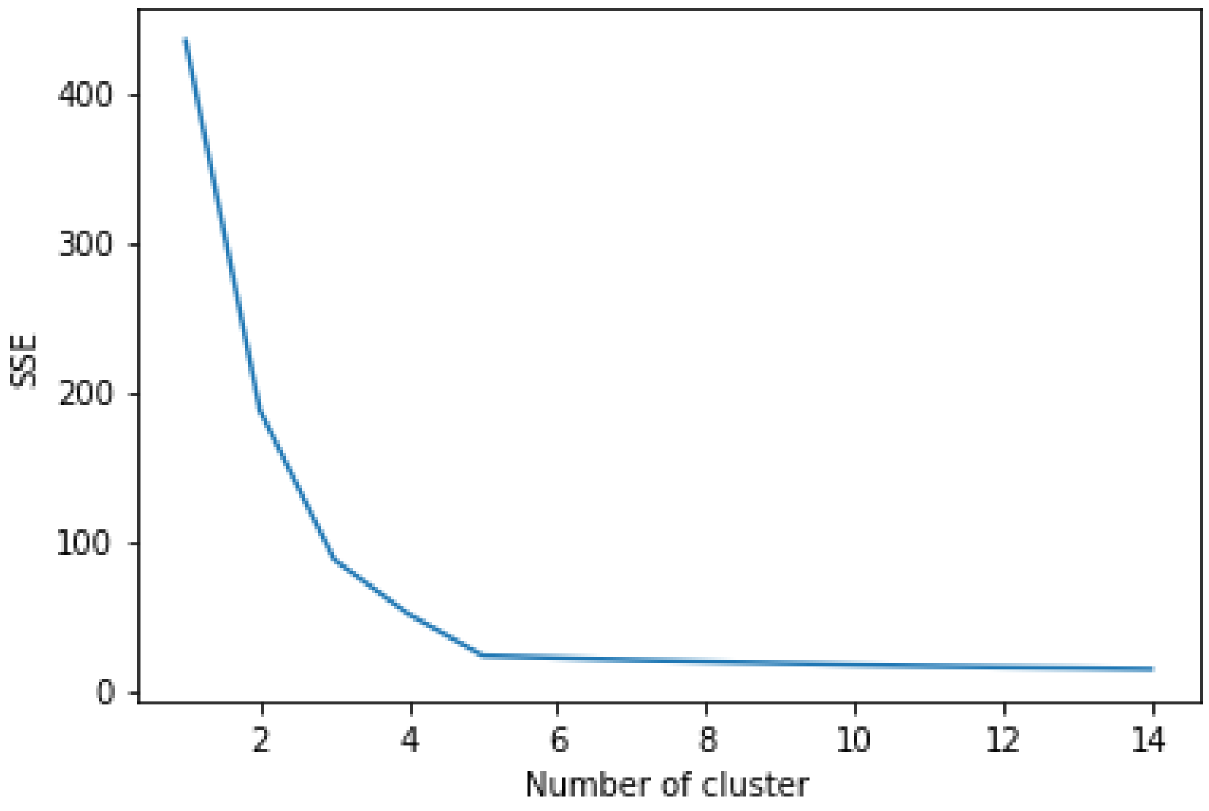
As can be seen from the above figure, the distribution of heroes per cluster improved as not a single cluster contained more than 40 heroes. The smallest cluster, cluster 3 representing likely position 5 heroes, was the smallest in terms of data points at just under 10, and once again we see heroes with high healing output and low resource dependency being clustered together. Position 1 heroes, now at cluster 4, and position

2 heroes, now at cluster 0, both had above average resource dependency with position 1's being higher than position 2's as expected. Once again we see a differentiation in these carries play styles as the former has by far the highest building damage as late game cores will be more effective at destroying the enemy team's base while the latter have the highest hero damage with access to high output damage spells and abilities earlier in the game. The distribution of size between these two clusters was now more inline with our original beliefs as more heroes were placed into the position 2 cluster than position 1 as position 2 heroes may be more flexible to play down a position at 3 or up a position at 1 but will on average play position 2.

Position 4 soft supports, as represented by cluster 1, once again saw above average deaths, assists, and stuns although the deviation from the mean with regards to deaths declined drastically. This is likely a good sign as before heroes who underperformed with high deaths may have been inaccurately clustered with position 4 heroes. Position 3 heroes at cluster 2 once again lack drastically above or below average statistics, meaning our algorithm tended to put middle performing heroes into this cluster. This result is satisfactory as position 3 heroes tend to be in the middle of the bell curve due to their wide range of playstyle and flexibility to change to other positions both higher and lower in game depending on team needs. Position 3 heroes tend to be reliable heroes that will engage in team fights as initiators, and we believe the results of the graph represent this notion well.

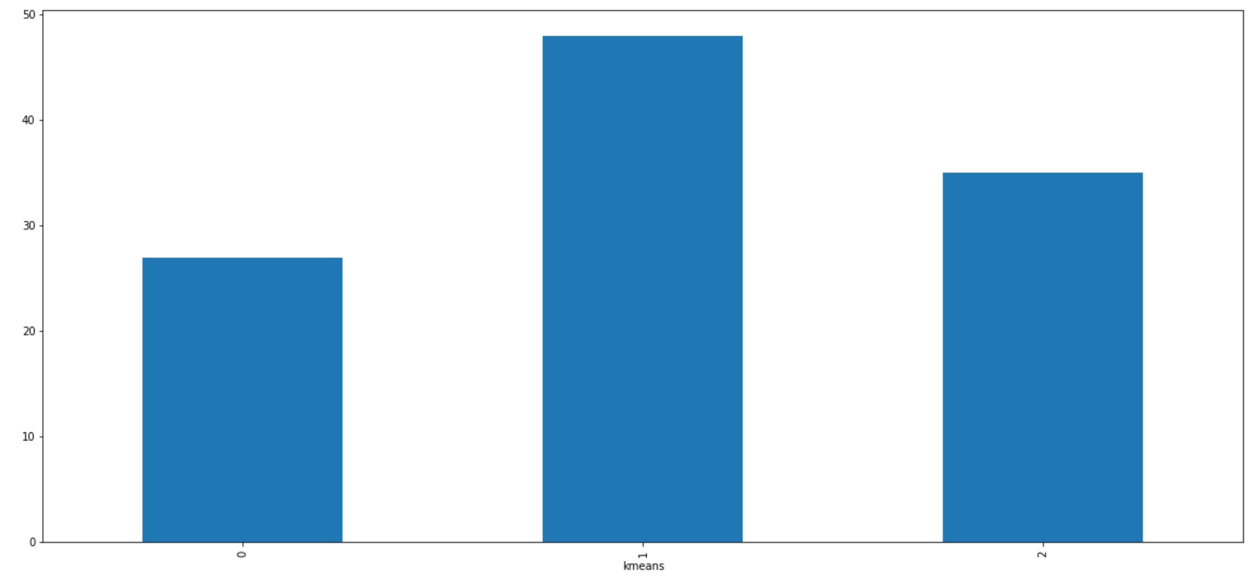
Although normalizing the data saw improvements to our initial clusterings, our group sought out to find what the true clustering representations of the dataset should be as by the inherent values of our data. To do this, we turned to hyperparameter tuning our data using elbow criterion.

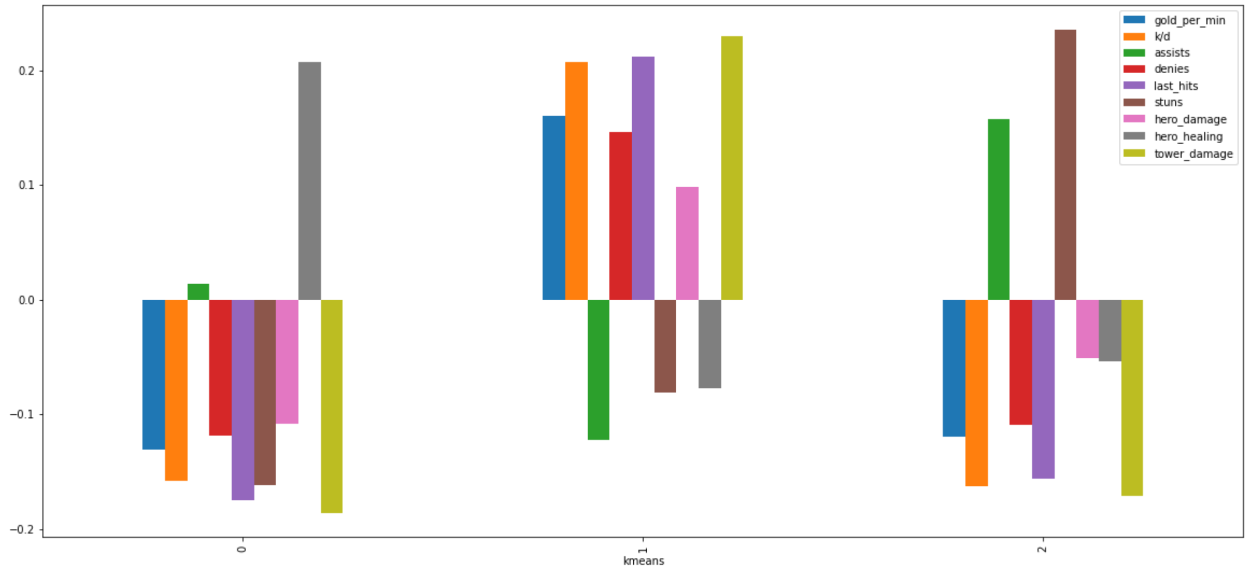
Figure 3D - Elbow Criterion for KMeans Clustering (k vs SSE)



By running our clustering algorithm a range of times incrementing the value of k after each iteration, and tracking the total SSE as calculated by the inertia of the clustering model, using the sum of distances of samples to their closest cluster center, we were able to find an elbow point where the tangential slope from an axis was equivalent to a forty-five degree angle at the k value of 3. With this newly tuned hyperparameter, our group decided to rerun our KMeans algorithm using k = 3 for both normalized and normalized data.

Figure 3E - Output for Kmeans(k=3) with Normalization Applied Before Clustering

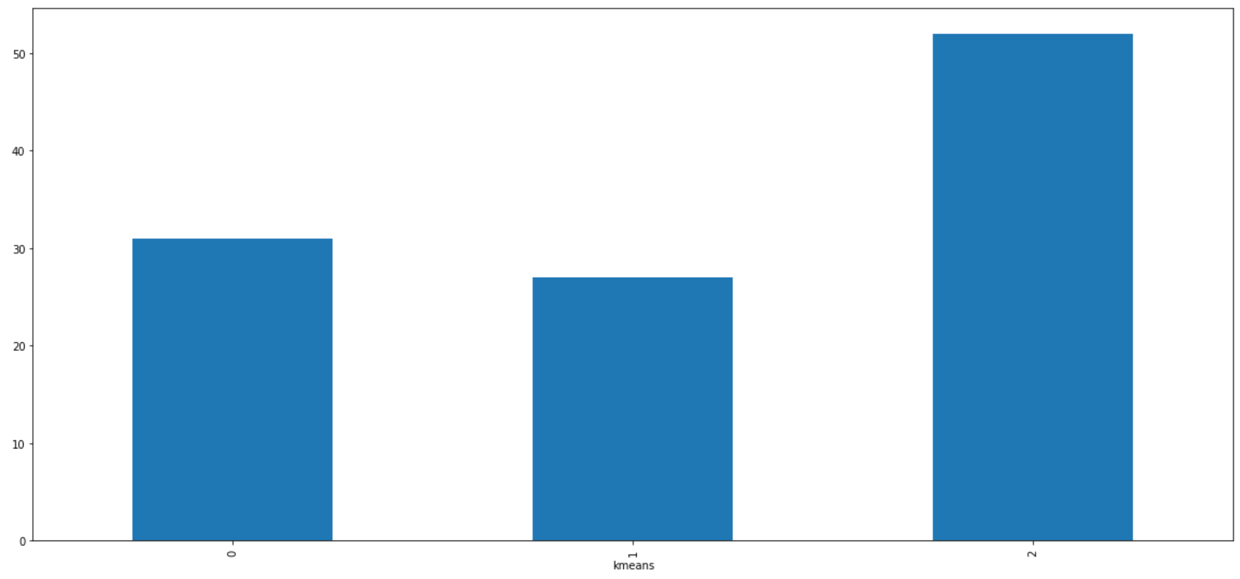


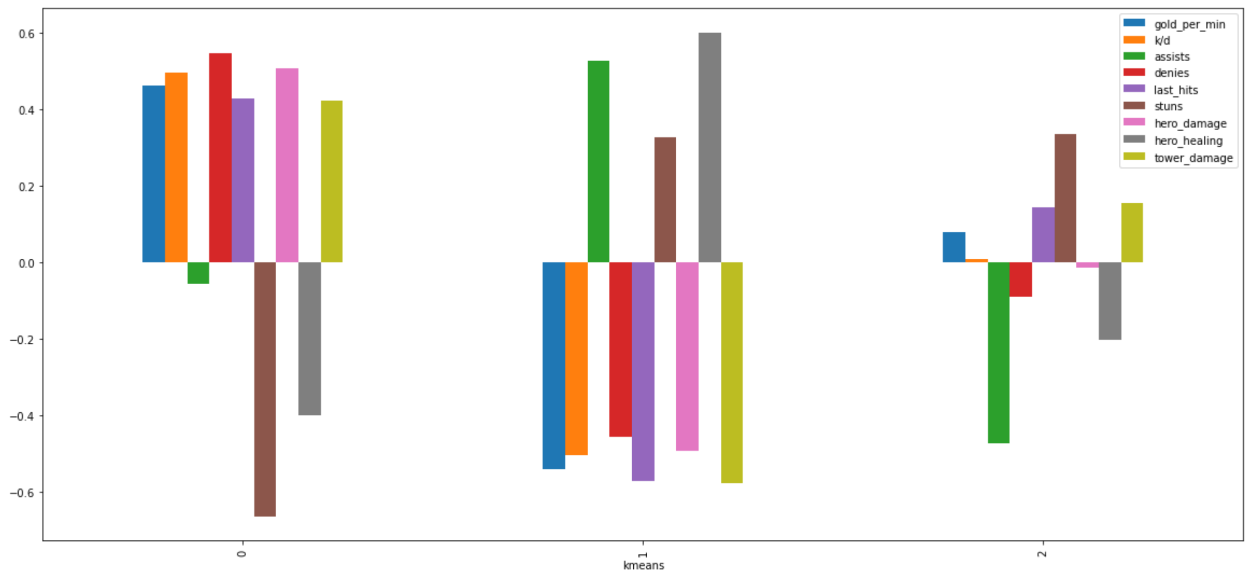


Using a k value of three and a reduced number of attributes, our clusterings as produced from normalized data produced three distinct clusterings. Cluster 0 had significantly higher healing with barely above average assists while being not resource dependent in terms of gold or experience and showing little interest in dealing hero or tower damage. This is inline with the playstyle of a support, such as position 4 or 5.

Cluster 1 shared many qualities of a carry with high resource acquisition and high damage output to both heroes and buildings. The final cluster consisted of high stun durations and assists, representing a hero capable of starting and participating in team fights. With the attribute averages of these three clusters, our group decided to identify them into three categories: supports, cores, and initiators.

Figure 3F - Output for Kmeans(k=3) with Normalization Applied After Clustering

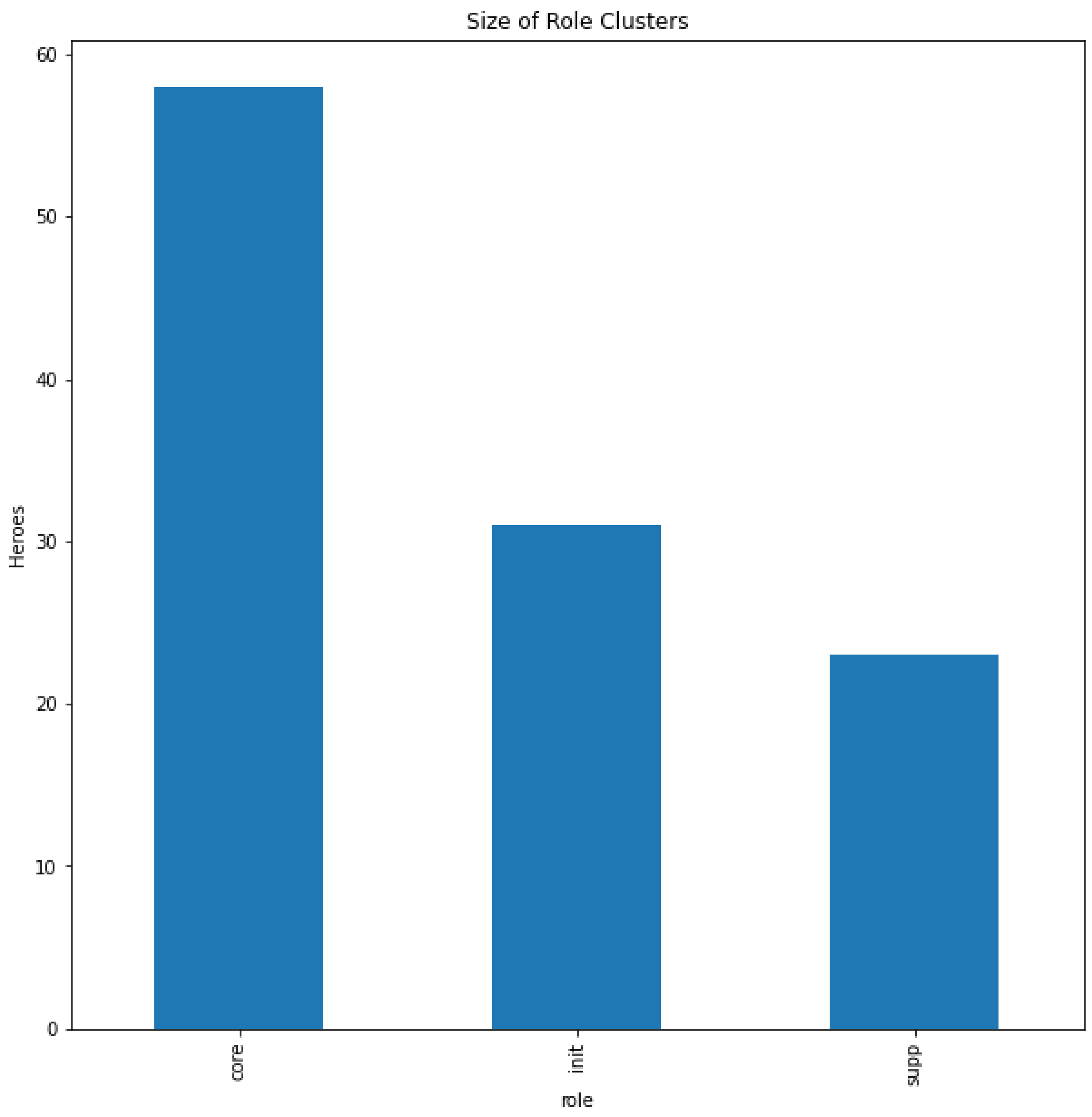




The output for the unnormalized clusterings had some similar trends as the previous clusterings as there appears to be a clear separation of cores, supports, and initiators (as respective to the order of the clusters in the figure above). However, the cluster representing initiators appeared to be composed more of averaged out heroes than those who were specifically fight starters. This is inline with what was seen with unnormalized runs with a k value of 5.

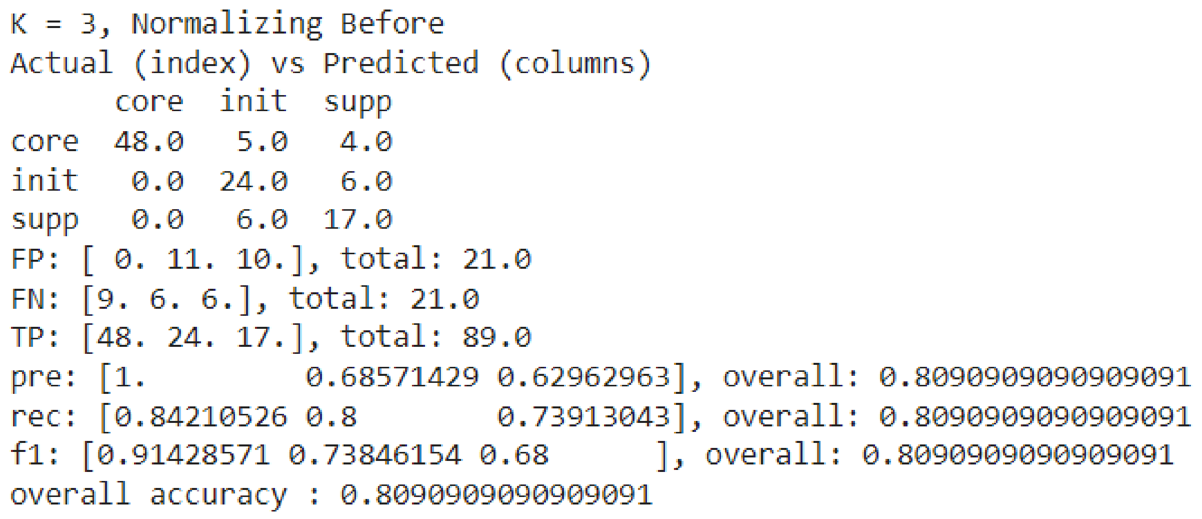
Our group was interested in calculating the accuracy of our clustering with a method other than inertia calculations. One idea we had was to use a ground truth. Unfortunately due to the ever changing updates to DOTA 2 as an online game, there was not a specific ground truth we could use from the source. To make up for this, we decided to poll DOTA players we knew to attempt to create the best attempt at a ground truth. This process involved contacting individuals we knew who played DOTA during the time the games in the players dataset was recorded. We were able to find three volunteers, and although we hoped to find more we still went forward with this process by having the individuals meet in an online call. During this call, a representative from our team would state the name of a hero, give the volunteers time to think over whether the hero was a core, initiator, or support, and then announce their vote by typing their answer into the chat and sending their answers at the same time. Volunteers could not attempt to argue for their answer until after voting concluded. After taking the plurality of the votes, we allowed players to explain their reasoning. The below figure represents the size of the clusters produced using this polling.

Figure 3G: Clusters of k=3 from Polling

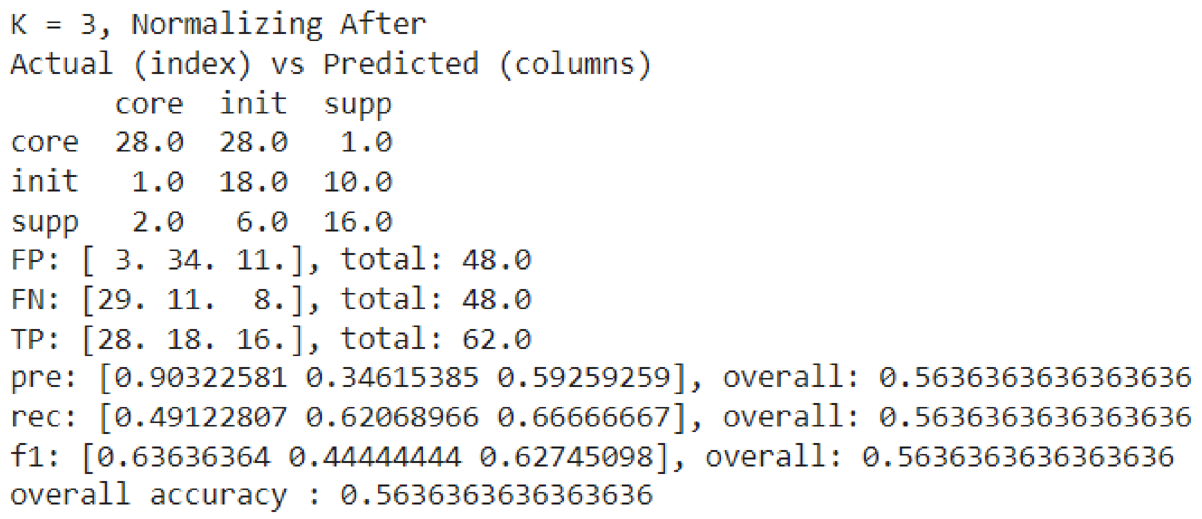


The results of the polling produced a large number of cores, making up roughly half of the heroes. Initiators totaled around 30 and supports made up the smallest group. Using these produced categories, we could now produce multi-categorical confusion matrices to determine the accuracies, precisions, recalls, and f1-scores of each group of heroes.

Figure 3H: Multi-categorical Confusion Matrices for K=3 KMeans



```
K = 3, Normalizing Before
Actual (index) vs Predicted (columns)
      core  init  supp
core  48.0   5.0   4.0
init   0.0  24.0   6.0
supp   0.0   6.0  17.0
FP: [ 0. 11. 10.], total: 21.0
FN: [9. 6. 6.], total: 21.0
TP: [48. 24. 17.], total: 89.0
pre: [1.         0.68571429 0.62962963], overall: 0.8090909090909091
rec: [0.84210526 0.8        0.73913043], overall: 0.8090909090909091
f1: [0.91428571 0.73846154 0.68      ], overall: 0.8090909090909091
overall accuracy : 0.8090909090909091
```

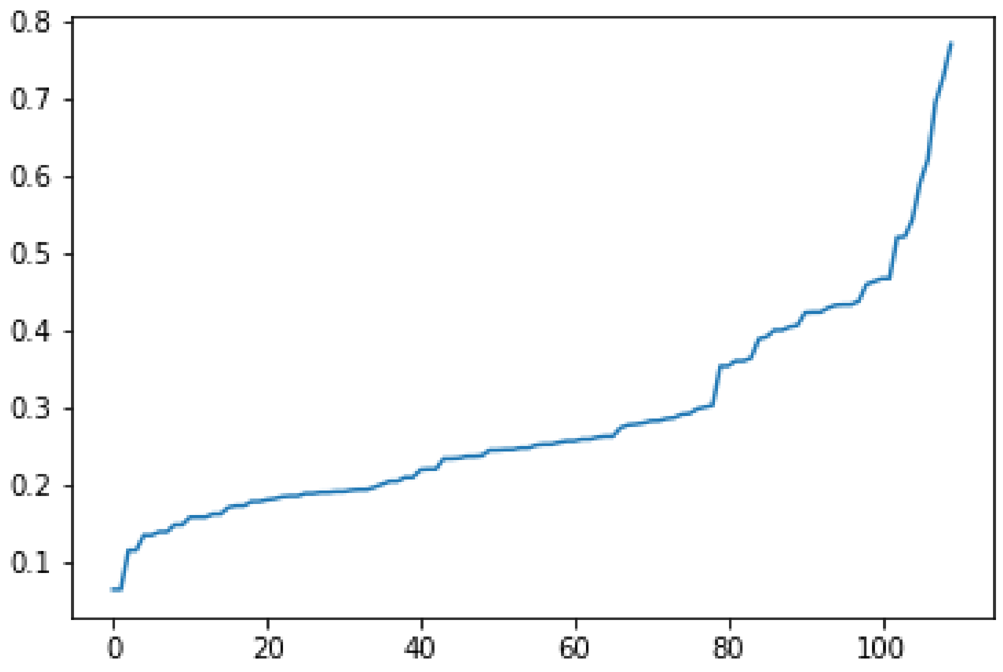


```
K = 3, Normalizing After
Actual (index) vs Predicted (columns)
      core  init  supp
core  28.0  28.0   1.0
init   1.0  18.0  10.0
supp   2.0   6.0  16.0
FP: [ 3. 34. 11.], total: 48.0
FN: [29. 11.  8.], total: 48.0
TP: [28. 18. 16.], total: 62.0
pre: [0.90322581 0.34615385 0.59259259], overall: 0.5636363636363636
rec: [0.49122807 0.62068966 0.66666667], overall: 0.5636363636363636
f1: [0.63636364 0.44444444 0.62745098], overall: 0.5636363636363636
overall accuracy : 0.5636363636363636
```

As can be seen from the above figure, KMeans performed with much higher accuracy when run with z-score normalized data as this approach led to roughly 81 percent accuracy and without producing 56 percent accuracy. This is heavily influenced by the sizes of the clusters formed as the normalized run shows more similar distribution to our polling generated ground truth than the unnormalized variant. Both approaches had high precision scores for cores, 100 percent and 90 percent respectively, as no or few false positives were generated for cores. This means that our clustering representations developed definitions of cores that would not lead to initiators or supports often being mistaken for cores. The unnormalized model often confused low performing or less resource dependent cores for initiators, generating 28 of the 19 false negatives for cores. The unnormalized clusterings did fairly well with supports compared to its performance with cores and initiators as supports had the highest recall and lowest number of false negatives, however the 67 percent recall still was lower than the normalized recall of 73 percent.

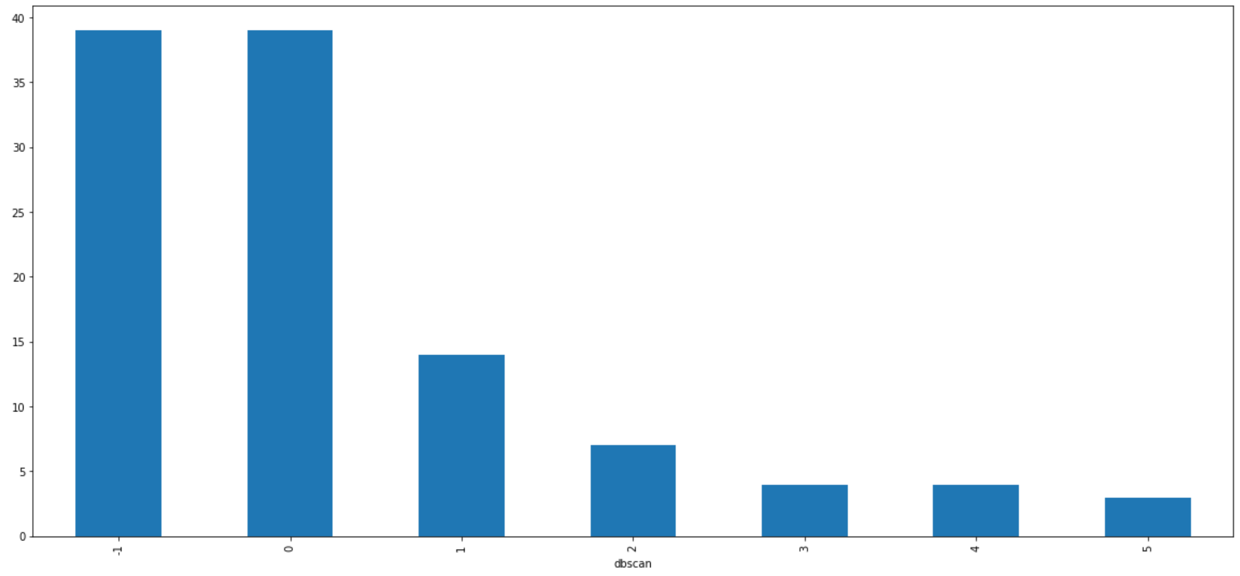
Fairly satisfied with our KMeans results, our group wondered if other clustering models could be used to represent our dataset. For our next approach, we decided to test density based clustering algorithms as we were interested in how their outlier detection would fare against a dataset with relatively sparse clusters of varying radii.

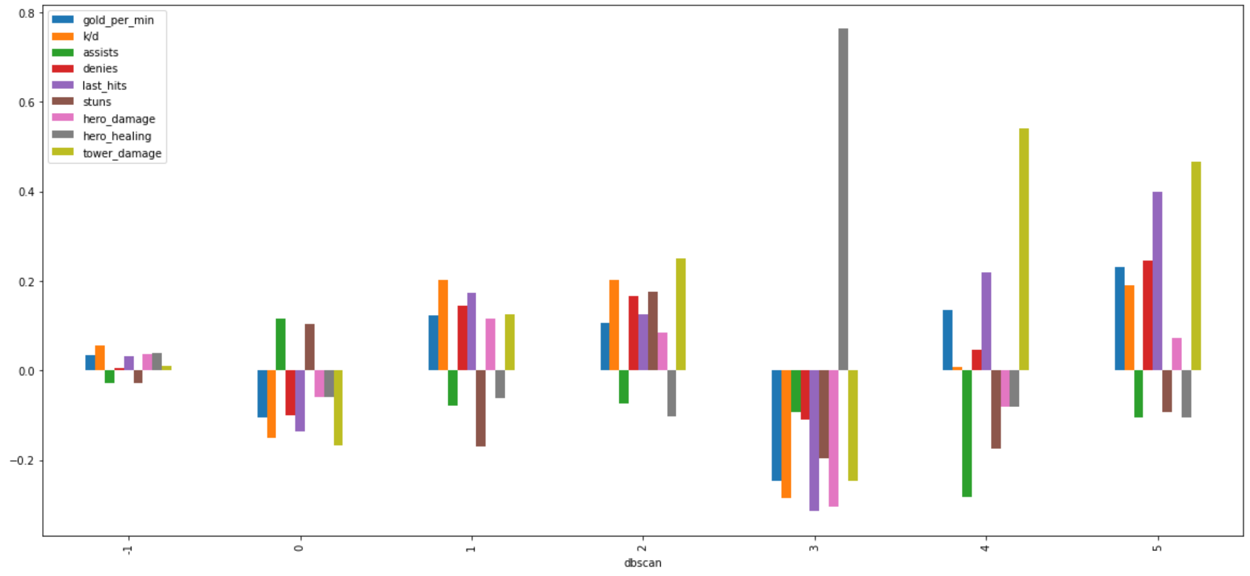
Figure 3I - Elbow Criterion for DBScan (epsilon vs nearest neighbors distance)



To tune our parameter, neighborhood distance measure epsilon, for DBScan, we used a similar method as with KMeans with elbow criterion. Where this approach differed, however, was that our parameter would be along the y-axis and represented by the total distance to the k-nearest-neighbors of a point. The k-nearest-neighbors algorithm would be run with a k value equal to the minpts parameter for our DBscan, then the points would be aligned along the x-axis in ascending order of their distance to k-nearest-neighbors. Using this metric, we decided to use 0.305 as a value of epsilon as that value represented where the change in slope in our graph began to approach the value desired for our elbow criterion.

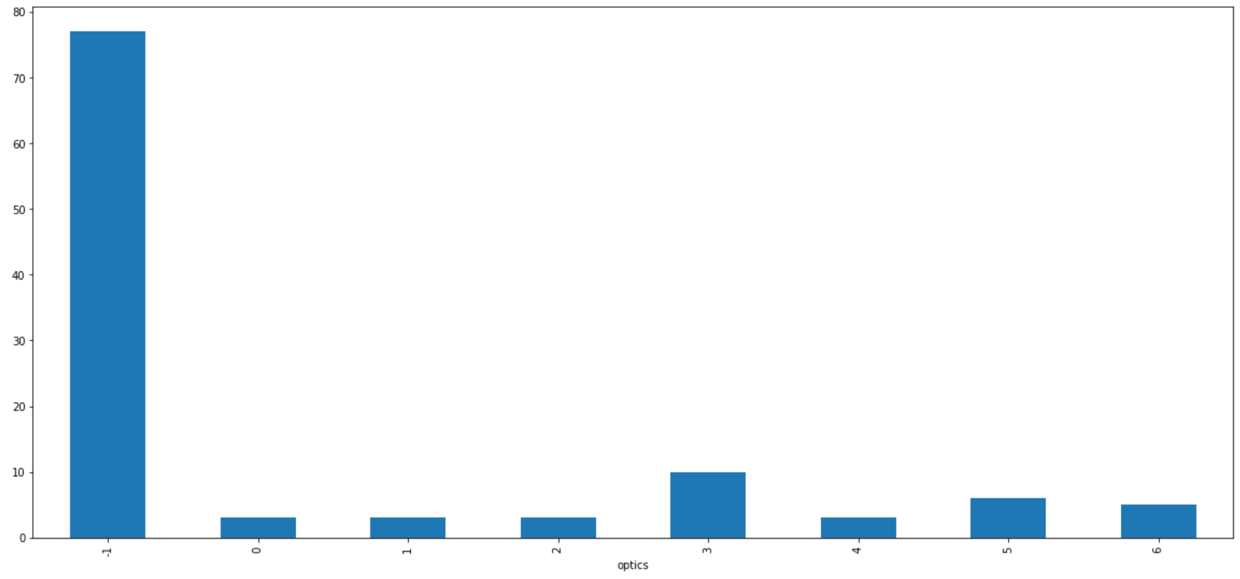
Figure 3J - DBScan with epsilon = 0.305 and minpts = 3

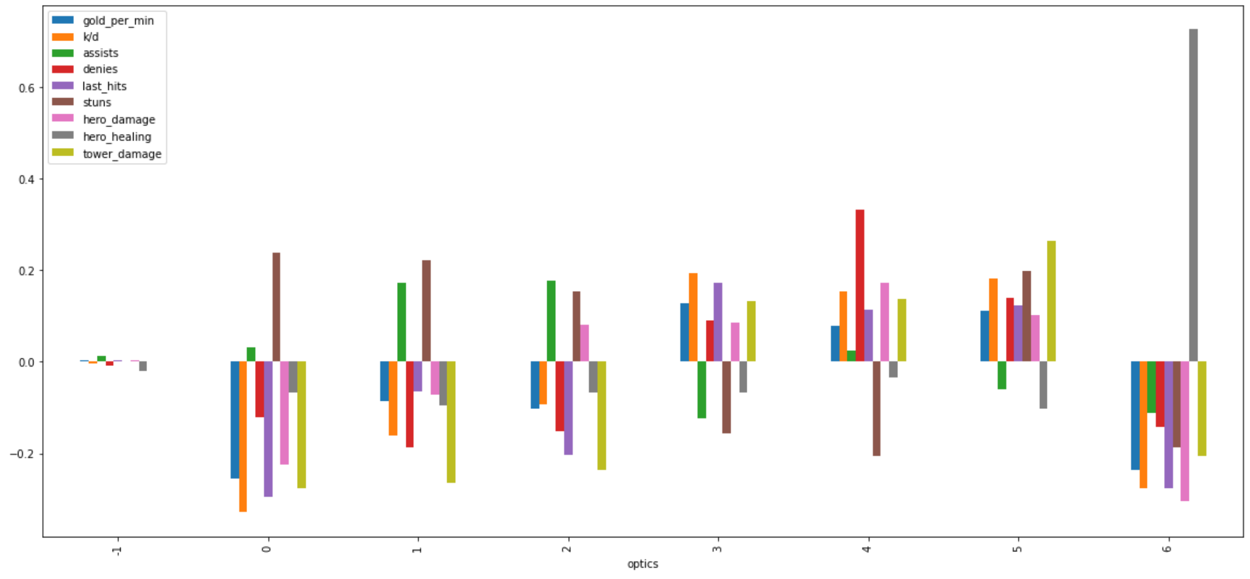




As can be seen from the graph above, DBScan produced a relatively large number of clusters. Distinct groups of healers, building destroyers, fight initiators, and cores of varying playstyles can be seen from the clusters. The clustering algorithm identified just under 40 of the heroes as outliers with these outliers having varied statistics that roughly equal out. Additionally, cluster 0 was much larger than the other non-outliers, roughly the same size as the outlier cluster. Knowing this, we wondered how using a variant of DBScan, OPTICS, which is more forgiving of clusters of varying radii lengths by using a maximum epsilon value would handle our data.

Figure 3K - OPTICS with max epsilon = .5 and minpts = 3

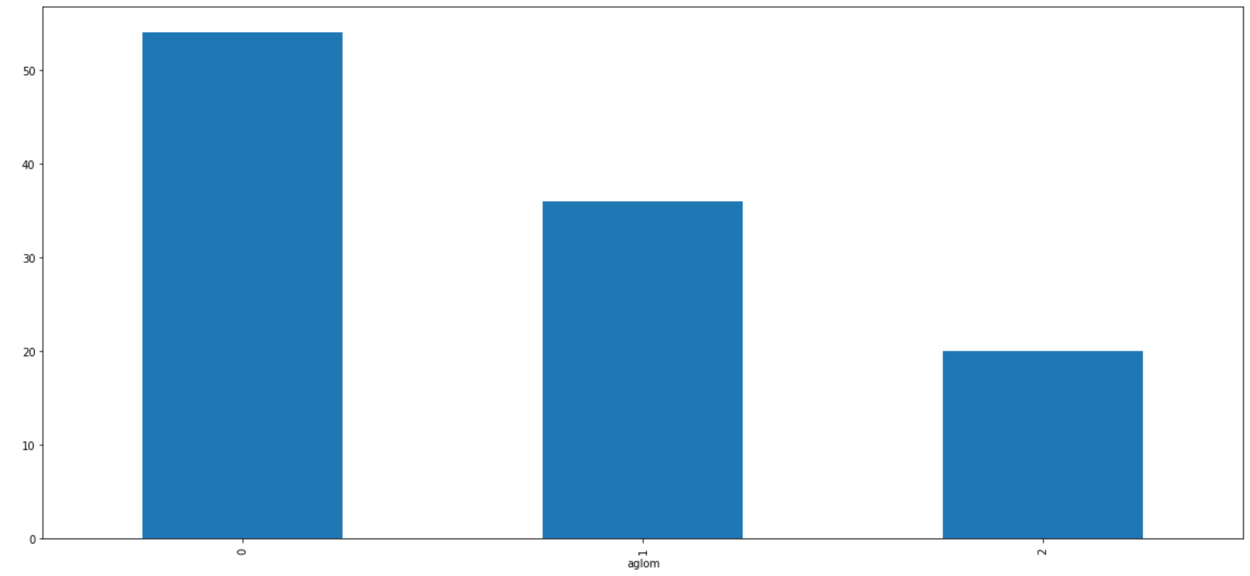


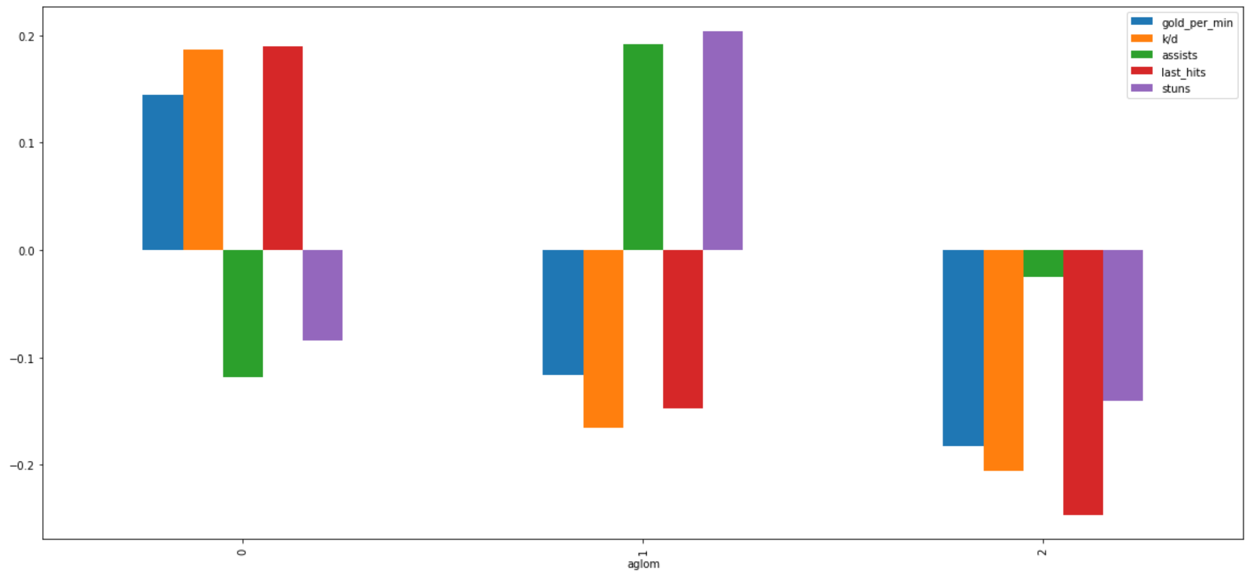


The above figure shows the output for our OPTICS clustering, and once again clusters of specific playstyles can be seen. This method produced three different variants of above average stun duration and assists clusterings all with varying resource dependency. Healers stand out once again as a specific group with the largest relative deviation from the mean of any column of the bar graph. Cluster 4 is comprised of heroes who average many more denies than any other heroes, likely indicating position 2 heroes who play the middle lane of the map early in the game as their playstyle in this area would be heavily dependant on this statistic to prevent the enemy team's position 2 from becoming the dominant hero of the early game. The largest non-outlier cluster was now cluster 3, representing cores who lack the ability to stun enemies, which was the only cluster to be larger than 10 heroes. The number of outliers, however, grew to almost 80. From the output, we can tell that this model is useful for identifying heroes of specific playstyles in very pure clusters, but density based clustering is not the best approach to classify the majority of heroes into a small number of overarching categories.

The final clustering technique our team thought to try was hierarchical based clustering. Using agglomerative clustering, we could set the number of clusters desired, similar to KMeans, and attempt to use different linkage methods to see what would produce the highest accuracy. This linkage metric ended up being Ward's method using the objective function of Error Sum of Squares (or ESS) as the linkage method would attempt to determine which clusters should be merged depending upon the change of the ESS by assigning a new centroid to the points of the merged clusters.

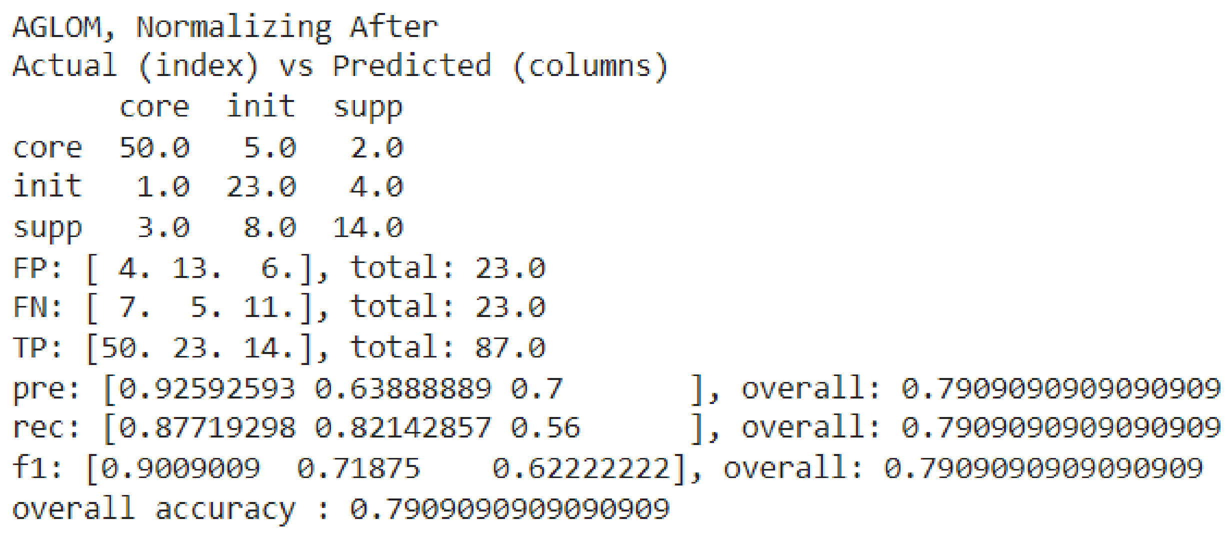
Figure 3L - Output for Agglomerative Clustering k = 3 and linkage = ‘ward’





The output for agglomerative clustering that we achieved using a reduced number of attributes (gold per minute, kill death ratio, assists, last hits, and stuns) did a fair job at identifying three distinct clusters similarly to our KMeans approach. The clusters in order do a fair job of identifying cores, intiatiors, and supports by their attributes. It is worth noting that total healing was removed as when included the supports cluster would solely consist of healers leading supports of varying playstyles split amongst the other two clusters. With this output, we decided to test our hierarchical results against our polling ground truth.

Figure 3M - Confusion Matrix for Agglomerative Clustering



```
AGLOM, Normalizing After
Actual (index) vs Predicted (columns)
      core  init  supp
core  50.0   5.0   2.0
init   1.0  23.0   4.0
supp   3.0   8.0  14.0
FP: [ 4. 13.  6.], total: 23.0
FN: [ 7.  5. 11.], total: 23.0
TP: [50. 23. 14.], total: 87.0
pre: [0.92592593 0.63888889 0.7       ], overall: 0.7909090909090909
rec: [0.87719298 0.82142857 0.56      ], overall: 0.7909090909090909
f1: [0.9009009  0.71875    0.62222222], overall: 0.7909090909090909
overall accuracy : 0.7909090909090909
```

As can be seen from the confusion matrix above, hierarchical clustering did a fair job of representing the heroes as cores, initiators, or supports with 79% overall accuracy. Cores once again had the highest precision, and this clustering algorithm struggled with supports classifying 3 as core and 8 as initiators leading to the lowest recall at 56%.

Going Further - It is clear from our best runs of KMeans and Agglomerative clustering that it is much easier to identify cores than supports or initiators using in game metrics. Part of this is due to the statistics provided as cores are dependent upon gold and experience and output high levels of damage to varying opponents which could be found in our dataset easily, while our dataset lacked information on damage a player received which could identify tankier initiators or information on gold spent on supporting items or the number of camps stacked (a technique in which supports increase the total potential of gold a core can obtain from that team's jungle area of the map) which would have been strong identifiers for supports other than the given information of healing output. In addition, DBScan and OPTICS did a fair job identifying small, pure clusters of heroes with similar specific play styles but were overzealous to label heroes as outliers despite tuning efforts. It would be interesting to find a clustering algorithm which handles outliers in a different manner and see if that could produce differing numbers or sizes of clusters.

CONCLUSION

Lam - The goal of this project for me is to create a prediction tree for the match outcome based solely on heroes picked on a team. I had a good time looking at all these dataset and finding a way to generate an input file for my personal C45 algorithm from lab3. At first, there were so many different factors that I was trying to consider when deciding on this input file. For instance, my initial thought was quite complex, accounting for the playstyle of the hero, and how often that hero is picked and etc. However, I came to the conclusion of just creating a sparse vector of 1s and 0s (1 as picked) for all heroes, aggregating low picked percentage hero into 1 category. Of course there were obstacles in the beginning as described in the method analysis but I managed to get through it by discussing amongst our group and the professor. The experience of this project was great as it allowed me to apply what I have learned from the class this quarter to put it into practice on a game that I am actually quite familiar with too. In terms of our team, the work is divided equally, with each person assigned a very specific task, in order to create a final version of all our data analysis.

Luke - This project involved a myriad of unique and interesting challenges. Establishing which classifiers should be used and how our data should be formatted for each to prevent overfit led to us attempting many methods from expanding to densifying our dataset. It was interesting to see the variation in results produced by attempting to classify by strategy, whether that be by hero or ability selection, or by skill using a player's calculated ranking. I would have liked to see higher accuracies for these predictions, but above fifty percent is still a satisfying result knowing that individual hero win rates rarely exceed fifty-five percent in DOTA. It would be interesting to see if player statistics relative to heroes could have been used for predicting wins by team hero composition, although that would have created a much more complicated model. For clustering, it was intriguing to see how the different algorithms handled the data in different ways and how changes to parameters and selected attributes could modify the results. I was personally satisfied to see our best ground truth comparisons reach around eighty percent accuracy, although I wish our process of ground truth construction could have involved more volunteers or direct source data. After this project, I am curious to know if there is a clustering algorithm that can better handle outliers in our dataset, although DBScan and OPTICS did produce engaging results relative to unique play styles with relatively dense and pure clusters.

Joey - This project, at its core, brought together a myriad of topics that this course taught in an applicable manner. Extending the classification of this dataset to random forests and k-neighbors brought an interesting comparison of how the two classifiers compared on a relatively dense dataset. We saw that while k-neighbors was generally more precise, random forest did exceptionally well at determining if a team would win solely on attributes chosen (based off of the recall value). On the other hand, k-neighbors was a more ‘balanced’ classifier regarding the classification accuracy (a win or a loss roughly had an equal chance at being poorly predicted). I think more consideration can be given towards densifying this dataset based solely on heroes and abilities. Grouping heroes and abilities together based on ‘similarity’ may have differing classification results instead of the method we chose (grouping highly picked heroes/abilities together). In certain cases, we may be able to group heroes together based on their clustering results. Perhaps clustering heroes based on their perceived role and using that to determine a team's balance may have influenced the classification process (for better or worse). For abilities, we could choose to group them based on their ability types (active, passive, targeting, etc.) to determine a team's skill balance. This could also enhance or worsen the classification results. Overall, this project brought exciting algorithms and an interesting dataset together, and I would be interested in seeing further work done with these classifiers and dataset.

David- We find that the insights produced from this market basket data analysis can certainly provide a great deal of help when teams are choosing their heroes. A teammate in a team of size 5 can use this insight of what their teammates picked to influence their own hero selection. By doing so, they can choose the most popular hero picked and increase their chances of winning the match. Likewise, this information could be useful when teams have a chance to ban heroes from being selected during a match. By banning certain popular heroes, a team can prevent the enemy team from choosing a hero for the match. Preventing the enemy team from picking certain popular heroes can help increase the chances of winning for the team that chose the ban. Overall, the insight produced from this market basket analysis can help influence a match's outcome before the match started. Although the market basket data analysis was useful for analyzing team compositions, we saw its flaw where it favored heavily-picked, popular characters for certain heroes since these popular heroes are picked irregardless of its synergy with a given hero. What's required is manual attention and input using domain knowledge to filter these association rules to the core association rules that describe hero synergy as well as common hero counter picks.

Appendix

Association Rules Generated for Dota Heroes

```
npc_dota_hero_antimage -> npc_dota_hero_nevermore
npc_dota_hero_antimage -> npc_dota_hero_pudge
npc_dota_hero_antimage -> npc_dota_hero_windrunner
npc_dota_hero_lion -> npc_dota_hero_antimage
npc_dota_hero_antimage -> npc_dota_hero_lion
npc_dota_hero_antimage -> npc_dota_hero_sniper
npc_dota_hero_antimage -> npc_dota_hero_invoker
npc_dota_hero_antimage -> npc_dota_hero_silencer
npc_dota_hero_antimage -> npc_dota_hero_ogre_magi
npc_dota_hero_antimage -> npc_dota_hero_legion_commander
npc_dota_hero_axe -> npc_dota_hero_pudge
npc_dota_hero_axe -> npc_dota_hero_lion
npc_dota_hero_axe -> npc_dota_hero_phantom_assassin
npc_dota_hero_dazzle -> npc_dota_hero_axe
npc_dota_hero_axe -> npc_dota_hero_dazzle
npc_dota_hero_axe -> npc_dota_hero_ogre_magi
npc_dota_hero_crystal_maiden -> npc_dota_hero_pudge
npc_dota_hero_crystal_maiden -> npc_dota_hero_phantom_assassin
npc_dota_hero_drow_ranger -> npc_dota_hero_pudge
npc_dota_hero_drow_ranger -> npc_dota_hero_lion
npc_dota_hero_witch_doctor -> npc_dota_hero_drow_ranger
npc_dota_hero_drow_ranger -> npc_dota_hero_sniper
npc_dota_hero_drow_ranger -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_drow_ranger
npc_dota_hero_drow_ranger -> npc_dota_hero_silencer
npc_dota_hero_ogre_magi -> npc_dota_hero_drow_ranger
npc_dota_hero_drow_ranger -> npc_dota_hero_ogre_magi
npc_dota_hero_earthshaker -> npc_dota_hero_juggernaut
npc_dota_hero_earthshaker -> npc_dota_hero_pudge
npc_dota_hero_earthshaker -> npc_dota_hero_lion
npc_dota_hero_earthshaker -> npc_dota_hero_sniper
npc_dota_hero_earthshaker -> npc_dota_hero_phantom_assassin
npc_dota_hero_earthshaker -> npc_dota_hero_invoker
npc_dota_hero_earthshaker -> npc_dota_hero_ogre_magi
npc_dota_hero_nevermore -> npc_dota_hero_juggernaut
npc_dota_hero_pudge -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_pudge
npc_dota_hero_windrunner -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_windrunner
npc_dota_hero_lina -> npc_dota_hero_juggernaut
```

```
npc_dota_hero_lion -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_lion
npc_dota_hero_shadow_shaman -> npc_dota_hero_juggernaut
npc_dota_hero_witch_doctor -> npc_dota_hero_juggernaut
npc_dota_hero_sniper -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_sniper
npc_dota_hero_dazzle -> npc_dota_hero_juggernaut
npc_dota_hero_jakiro -> npc_dota_hero_juggernaut
npc_dota_hero_spirit_breaker -> npc_dota_hero_juggernaut
npc_dota_hero_invoker -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_silencer
npc_dota_hero_ogre_magi -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_juggernaut
npc_dota_hero_magnataur -> npc_dota_hero_juggernaut
npc_dota_hero_skywrath_mage -> npc_dota_hero_juggernaut
npc_dota_hero_legion_commander -> npc_dota_hero_juggernaut
npc_dota_hero_juggernaut -> npc_dota_hero_legion_commander
npc_dota_hero_marci -> npc_dota_hero_juggernaut
npc_dota_hero_mirana -> npc_dota_hero_pudge
npc_dota_hero_mirana -> npc_dota_hero_lion
npc_dota_hero_mirana -> npc_dota_hero_phantom_assassin
npc_dota_hero_mirana -> npc_dota_hero_ogre_magi
npc_dota_hero_morphling -> npc_dota_hero_pudge
npc_dota_hero_morphling -> npc_dota_hero_lion
npc_dota_hero_nevermore -> npc_dota_hero_pudge
npc_dota_hero_lion -> npc_dota_hero_nevermore
npc_dota_hero_nevermore -> npc_dota_hero_lion
npc_dota_hero_skeleton_king -> npc_dota_hero_nevermore
npc_dota_hero_phantom_assassin -> npc_dota_hero_nevermore
npc_dota_hero_nevermore -> npc_dota_hero_phantom_assassin
npc_dota_hero_nevermore -> npc_dota_hero_ogre_magi
npc_dota_hero_slark -> npc_dota_hero_nevermore
npc_dota_hero_nevermore -> npc_dota_hero_legion_commander
npc_dota_hero_phantom_lancer -> npc_dota_hero_pudge
npc_dota_hero_storm_spirit -> npc_dota_hero_pudge
npc_dota_hero_windrunner -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_windrunner
npc_dota_hero_zuus -> npc_dota_hero_pudge
npc_dota_hero_kunkka -> npc_dota_hero_pudge
npc_dota_hero_lina -> npc_dota_hero_pudge
npc_dota_hero_lion -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_lion
npc_dota_hero_shadow_shaman -> npc_dota_hero_pudge
```

```
npc_dota_hero_tidehunter -> npc_dota_hero_pudge
npc_dota_hero_witch_doctor -> npc_dota_hero_pudge
npc_dota_hero_lich -> npc_dota_hero_pudge
npc_dota_hero_tinker -> npc_dota_hero_pudge
npc_dota_hero_sniper -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_sniper
npc_dota_hero_necrolyte -> npc_dota_hero_pudge
npc_dota_hero_queenofpain -> npc_dota_hero_pudge
npc_dota_hero_faceless_void -> npc_dota_hero_pudge
npc_dota_hero_skeleton_king -> npc_dota_hero_pudge
npc_dota_hero_phantom_assassin -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_phantom_assassin
npc_dota_hero_viper -> npc_dota_hero_pudge
npc_dota_hero_luna -> npc_dota_hero_pudge
npc_dota_hero_dazzle -> npc_dota_hero_pudge
npc_dota_hero_life_stealer -> npc_dota_hero_pudge
npc_dota_hero_weaver -> npc_dota_hero_pudge
npc_dota_hero_jakiro -> npc_dota_hero_pudge
npc_dota_hero_spectre -> npc_dota_hero_pudge
npc_dota_hero_ancient_apparition -> npc_dota_hero_pudge
npc_dota_hero_ursa -> npc_dota_hero_pudge
npc_dota_hero_spirit_breaker -> npc_dota_hero_pudge
npc_dota_hero_invoker -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_pudge
npc_dota_hero_obsidian_destroyer -> npc_dota_hero_pudge
npc_dota_hero_ogre_magi -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_pudge
npc_dota_hero_disruptor -> npc_dota_hero_pudge
npc_dota_hero_slark -> npc_dota_hero_pudge
npc_dota_hero_medusa -> npc_dota_hero_pudge
npc_dota_hero_magnataur -> npc_dota_hero_pudge
npc_dota_hero_bristleback -> npc_dota_hero_pudge
npc_dota_hero_skywrath_mage -> npc_dota_hero_pudge
npc_dota_hero_legion_commander -> npc_dota_hero_pudge
npc_dota_hero_pudge -> npc_dota_hero_legion_commander
npc_dota_hero_techies -> npc_dota_hero_pudge
npc_dota_hero_ember_spirit -> npc_dota_hero_pudge
npc_dota_hero_monkey_king -> npc_dota_hero_pudge
npc_dota_hero_dark_willow -> npc_dota_hero_pudge
npc_dota_hero_grimstroke -> npc_dota_hero_pudge
npc_dota_hero_hoodwink -> npc_dota_hero_pudge
npc_dota_hero_void_spirit -> npc_dota_hero_pudge
npc_dota_hero_snapfire -> npc_dota_hero_pudge
npc_dota_hero_mars -> npc_dota_hero_pudge
```

```
npc_dota_hero_marci -> npc_dota_hero_pudge
npc_dota_hero_storm_spirit -> npc_dota_hero_lion
npc_dota_hero_storm_spirit -> npc_dota_hero_ogre_magi
npc_dota_hero_windrunner -> npc_dota_hero_lion
npc_dota_hero_windrunner -> npc_dota_hero_sniper
npc_dota_hero_skeleton_king -> npc_dota_hero_windrunner
npc_dota_hero_phantom_assassin -> npc_dota_hero_windrunner
npc_dota_hero_windrunner -> npc_dota_hero_phantom_assassin
npc_dota_hero_spirit_breaker -> npc_dota_hero_windrunner
npc_dota_hero_windrunner -> npc_dota_hero_ogre_magi
npc_dota_hero_slark -> npc_dota_hero_windrunner
npc_dota_hero_windrunner -> npc_dota_hero_slark
npc_dota_hero_zuus -> npc_dota_hero_lion
npc_dota_hero_zuus -> npc_dota_hero_phantom_assassin
npc_dota_hero_zuus -> npc_dota_hero_ogre_magi
npc_dota_hero_kunkka -> npc_dota_hero_lion
npc_dota_hero_lina -> npc_dota_hero_lion
npc_dota_hero_lina -> npc_dota_hero_phantom_assassin
npc_dota_hero_lina -> npc_dota_hero_ogre_magi
npc_dota_hero_shadow_shaman -> npc_dota_hero_lion
npc_dota_hero_witch_doctor -> npc_dota_hero_lion
npc_dota_hero_tinker -> npc_dota_hero_lion
npc_dota_hero_sniper -> npc_dota_hero_lion
npc_dota_hero_lion -> npc_dota_hero_sniper
npc_dota_hero_necrolyte -> npc_dota_hero_lion
npc_dota_hero_queenofpain -> npc_dota_hero_lion
npc_dota_hero_faceless_void -> npc_dota_hero_lion
npc_dota_hero_skeleton_king -> npc_dota_hero_lion
npc_dota_hero_phantom_assassin -> npc_dota_hero_lion
npc_dota_hero_lion -> npc_dota_hero_phantom_assassin
npc_dota_hero_luna -> npc_dota_hero_lion
npc_dota_hero_dazzle -> npc_dota_hero_lion
npc_dota_hero_life_stealer -> npc_dota_hero_lion
npc_dota_hero_weaver -> npc_dota_hero_lion
npc_dota_hero_jakiro -> npc_dota_hero_lion
npc_dota_hero_spectre -> npc_dota_hero_lion
npc_dota_hero_ursa -> npc_dota_hero_lion
npc_dota_hero_spirit_breaker -> npc_dota_hero_lion
npc_dota_hero_invoker -> npc_dota_hero_lion
npc_dota_hero_lion -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_lion
npc_dota_hero_obsidian_destroyer -> npc_dota_hero_lion
npc_dota_hero_ogre_magi -> npc_dota_hero_lion
npc_dota_hero_lion -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_lion
npc_dota_hero_slark -> npc_dota_hero_lion
```

```
npc_dota_hero_lion -> npc_dota_hero_slark
npc_dota_hero_medusa -> npc_dota_hero_lion
npc_dota_hero_magnataur -> npc_dota_hero_lion
npc_dota_hero_bristleback -> npc_dota_hero_lion
npc_dota_hero_skywrath_mage -> npc_dota_hero_lion
npc_dota_hero_legion_commander -> npc_dota_hero_lion
npc_dota_hero_lion -> npc_dota_hero_legion_commander
npc_dota_hero_monkey_king -> npc_dota_hero_lion
npc_dota_hero_void_spirit -> npc_dota_hero_lion
npc_dota_hero_mars -> npc_dota_hero_lion
npc_dota_hero_marci -> npc_dota_hero_lion
npc_dota_hero_shadow_shaman -> npc_dota_hero_sniper
npc_dota_hero_shadow_shaman -> npc_dota_hero_phantom_assassin
npc_dota_hero_shadow_shaman -> npc_dota_hero_invoker
npc_dota_hero_shadow_shaman -> npc_dota_hero_ogre_magi
npc_dota_hero_shadow_shaman -> npc_dota_hero_legion_commander
npc_dota_hero_witch_doctor -> npc_dota_hero_sniper
npc_dota_hero_witch_doctor -> npc_dota_hero_phantom_assassin
npc_dota_hero_witch_doctor -> npc_dota_hero_invoker
npc_dota_hero_witch_doctor -> npc_dota_hero_ogre_magi
npc_dota_hero_witch_doctor -> npc_dota_hero_legion_commander
npc_dota_hero_tinker -> npc_dota_hero_phantom_assassin
npc_dota_hero_tinker -> npc_dota_hero_ogre_magi
npc_dota_hero_faceless_void -> npc_dota_hero_sniper
npc_dota_hero_skeleton_king -> npc_dota_hero_sniper
npc_dota_hero_phantom_assassin -> npc_dota_hero_sniper
npc_dota_hero_sniper -> npc_dota_hero_phantom_assassin
npc_dota_hero_dazzle -> npc_dota_hero_sniper
npc_dota_hero_jakiro -> npc_dota_hero_sniper
npc_dota_hero_spirit_breaker -> npc_dota_hero_sniper
npc_dota_hero_invoker -> npc_dota_hero_sniper
npc_dota_hero_sniper -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_sniper
npc_dota_hero_sniper -> npc_dota_hero_silencer
npc_dota_hero_ogre_magi -> npc_dota_hero_sniper
npc_dota_hero_sniper -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_sniper
npc_dota_hero_slark -> npc_dota_hero_sniper
npc_dota_hero_bristleback -> npc_dota_hero_sniper
npc_dota_hero_skywrath_mage -> npc_dota_hero_sniper
npc_dota_hero_legion_commander -> npc_dota_hero_sniper
npc_dota_hero_sniper -> npc_dota_hero_legion_commander
npc_dota_hero_monkey_king -> npc_dota_hero_sniper
npc_dota_hero_marci -> npc_dota_hero_sniper
npc_dota_hero_faceless_void -> npc_dota_hero_invoker
npc_dota_hero_faceless_void -> npc_dota_hero_ogre_magi
```

```
npc_dota_hero_skeleton_king -> npc_dota_hero_invoker
npc_dota_hero_skeleton_king -> npc_dota_hero_silencer
npc_dota_hero_skeleton_king -> npc_dota_hero_ogre_magi
npc_dota_hero_dazzle -> npc_dota_hero_phantom_assassin
npc_dota_hero_jakiro -> npc_dota_hero_phantom_assassin
npc_dota_hero_spirit_breaker -> npc_dota_hero_phantom_assassin
npc_dota_hero_invoker -> npc_dota_hero_phantom_assassin
npc_dota_hero_phantom_assassin -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_phantom_assassin
npc_dota_hero_phantom_assassin -> npc_dota_hero_silencer
npc_dota_hero_ogre_magi -> npc_dota_hero_phantom_assassin
npc_dota_hero_phantom_assassin -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_phantom_assassin
npc_dota_hero_magnataur -> npc_dota_hero_phantom_assassin
npc_dota_hero_bristleback -> npc_dota_hero_phantom_assassin
npc_dota_hero_skywrath_mage -> npc_dota_hero_phantom_assassin
npc_dota_hero_legion_commander -> npc_dota_hero_phantom_assassin
npc_dota_hero_phantom_assassin -> npc_dota_hero_legion_commander
npc_dota_hero_marci -> npc_dota_hero_phantom_assassin
npc_dota_hero_luna -> npc_dota_hero_ogre_magi
npc_dota_hero_dazzle -> npc_dota_hero_ogre_magi
npc_dota_hero_weaver -> npc_dota_hero_ogre_magi
npc_dota_hero_jakiro -> npc_dota_hero_invoker
npc_dota_hero_jakiro -> npc_dota_hero_ogre_magi
npc_dota_hero_jakiro -> npc_dota_hero_legion_commander
npc_dota_hero_spirit_breaker -> npc_dota_hero_invoker
npc_dota_hero_spirit_breaker -> npc_dota_hero_ogre_magi
npc_dota_hero_silencer -> npc_dota_hero_invoker
npc_dota_hero_ogre_magi -> npc_dota_hero_invoker
npc_dota_hero_invoker -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_invoker
npc_dota_hero_slark -> npc_dota_hero_invoker
npc_dota_hero_magnataur -> npc_dota_hero_invoker
npc_dota_hero_legion_commander -> npc_dota_hero_invoker
npc_dota_hero_invoker -> npc_dota_hero_legion_commander
npc_dota_hero_marci -> npc_dota_hero_invoker
npc_dota_hero_silencer -> npc_dota_hero_ogre_magi
npc_dota_hero_slark -> npc_dota_hero_silencer
npc_dota_hero_silencer -> npc_dota_hero_slark
npc_dota_hero_legion_commander -> npc_dota_hero_silencer
npc_dota_hero_silencer -> npc_dota_hero_legion_commander
npc_dota_hero_rubick -> npc_dota_hero_ogre_magi
npc_dota_hero_slark -> npc_dota_hero_ogre_magi
npc_dota_hero_ogre_magi -> npc_dota_hero_slark
npc_dota_hero_magnataur -> npc_dota_hero_ogre_magi
npc_dota_hero_bristleback -> npc_dota_hero_ogre_magi
```

```
npc_dota_hero_skywrath_mage -> npc_dota_hero_ogre_magi
npc_dota_hero_legion_commander -> npc_dota_hero_ogre_magi
npc_dota_hero_ogre_magi -> npc_dota_hero_legion_commander
npc_dota_hero_monkey_king -> npc_dota_hero_ogre_magi
npc_dota_hero_marci -> npc_dota_hero_ogre_magi
npc_dota_hero_rubick -> npc_dota_hero_legion_commander
npc_dota_hero_legion_commander -> npc_dota_hero_slark
npc_dota_hero_slark -> npc_dota_hero_legion_commander
npc_dota_hero_skywrath_mage -> npc_dota_hero_legion_commander
```