Playing to Win: Machine Learning for Drafting in Games

Elizabeth Reiland
ereilan1
ereiland@jhu.edu

Tristan Orton-Urbina
tortonu1
tortonu1@jhu.edu

Abstract

Heroes of the Storm is a Multiplayer Online Battle Arena (MOBA) game in the same genre as the more well known *League of Legends* or *DOTA 2*. Each game consists of two teams of five players each working together to outplay their opponents by means of fights and objectives, ultimately destroying the opposing team's core structure. This paper documents our attempt to predict which team won a particular game using several machine-learning algorithms based on various features, with the primary focus being on which heroes were drafted by each team.

1 Introduction

Heroes of the Storm (HotS) by Blizzard Entertainment is a popular computer game with a player base of over 9 million. The game encourages highly competitive play, including professional teams that compete in global tournaments with multi-million dollar prize pools.

Each *Heroes of the Storm* match consists of two teams composed of five players each pitted against each other. Before the game starts, the players are told the map they will play one. There are currently 11 unique battlegrounds, each with different objectives for players to complete to help them win the game. Knowing the map, each player selects a "hero" to play out of a pool of 59 distinct heroes. Once a player has chosen a hero, that character cannot be picked by another player in the same match.

Which heroes are chosen can have a huge impact on how well a team performs, as each has a unique set of characteristics and abilities, bringing different strengths to the team. To be effective, a hero must synergize with other heroes on the same team while simultaneously taking advantage of the weaknesses of the enemy team and the demands of the chosen battleground. As a result, the process of drafting heroes is critical to the outcome of a match.

Assuming equally skilled players on each team, it is not uncommon for teams to be heavily favored before play even begins simply based off of the hero picks and battleground selection. The goal of our project is to identify which aspects of teams contribute the most to success, and attempt to use machine learning techniques to isolate these features and use them to predict the outcome of matches.

Previously, others have looked at win/loss predictors for games of the genre as HotS, namely *League of Legends* and *DOTA 2*. Both of these games follow the same format as HotS: two teams of five facing off with players drafting their characters from the pool of all heroes.

Our primary reference was (**?**). The focus in this paper was predicting wins and losses based on player statistics in *League of Legends*; the authors first used clustering models to try to group players based off of their average in game stats, then applied the cluster labels to predict the winner of a match based off the "player composition" of the two teams. The authors opted to use logistic regression, Gaussian discriminant analysis, and support vector machines as their classification models. They compared their results with these "player type" features to a baseline predictor which used only the official in-game classes.

In (**?**), the authors take a more statistical approach to attempt to answer whether hero selection (both individual and combinations of heroes) influenced the outcome of games in *DOTA 2*, using "hero scores" as the main features with a logistic regression model. These hero scores were created from the base attributes of the characters, though the article mentions the possible approach of using classes of heroes as features. Their results showed that hero selection did play a significant role in the outcome of games at all skill levels.

For our work, we hybridize these two ideas. With the format of our data, extracting player information to do player type clustering was infeasible. However, knowing that most players have a handful of heroes that they primarily play, and knowing that these heroes tend to have similar characteristics or class, we opted to attempt applying the methods in (**?**) to "hero types" rather than "player types."

2 Techniques Used

For this project we applied several supervised learning algorithms to predict which team won a game. Specifically, we ran Neural Networks, Support Vector Machines, k-Nearest Neighbor, and Logistic Regression to classify our data. We also used the k-means clustering algorithm in an attempt to group heroes into subclasses.

2.1 Neural Networks

In this project, we chose to focus specifically on the **Multi-Layer Perceptron (MLP)** implementation of a neural network. A MLP is a supervised learning algorithm that learns a function that maps from the dimension of the input to a single output, in our case a binary classification (0 or 1). We chose to use a MLP with 1 hidden layer with approximately 100 hidden nodes because our data hovered around 160 dimensions (though not all features were used for some variants) and because we found this number of hidden nodes to be the best based on experimental data. We wanted to use Neural Networks because of their ability to approximate non-linear functions with high levels of detail, and we hypothesized that our problem might be non-linear.

2.2 SVMs

Support Vector Machines (SVMs) are a class of supervised learning methods that attempt to maximize the margin between data in the training set and the decision boundary. They can be used to classify on a linear boundary or on non-linear boundaries with the kernel trick. For this project, we restricted ourselves to using linear SVMs. We chose to use SVMs because they're a fairly basic algorithm, and if we saw promise we could extend this algorithm to using more complicated kernels, such as RBF or Poly. This was also one of the algorithms utilized in (**?**).

2.3 k-NN

The k-**Nearest Neighbors** algorithm takes in a large set of labeled training data; to predict novel examples, the algorithm finds the k (where k is a positive integer fixed in advance) nearest training examples by Euclidian distance. The algorithm then classifies the new example based on the most common label among these k nearest neighbors. We chose k-NN because assuming hero selection plays a significant role in game outcome (as it does in DOTA 2 based on (**?**)), we would expect games with similar hero compositions to end in the same way, and thus be close in Euclidian space.

2.4 Logistic Regression

Logistic Regression is a linear model for classification despite its name. Logistic regression is used to find a strictly linear separator between datasets using the logistic error function. It then classifies future examples based on this found separator. We chose this method due to its use in (**?**) with a large degree of success.

2.5 k-Means Clustering

k-**Means Clustering** is an unsupervised learning algorithm that groups data points into k "clusters." Initially k cluster centers are set to random values. For each iteration, each data point is assigned to a cluster based on which cluster center it is closest to (using Euclidean distance). Each cluster center is then reset to the average of all points assigned to that cluster.

We used this algorithm to attempt to group heroes into subclasses. In game, heroes are grouped into

8 types based on whether they use melee or ranged attacks along with a designation as warrior, assassin, support, or specialist. However not all heroes in one of these categories perform the same role in game. Using features such as takedowns, siege damage, healing, and more we sought to find more accurate classifications of the heroes with the hypothesis that these subtypes might give a way to predict which team won a game with fewer features.

This approach was inspired by (**?**) though in their work they clustered players based on their playstyles (a large part of which had to do with which characters they played).

2.6 Cross Validation

Cross Validation is a method used to assess the accuracy of a machine learning algorithm when a limited amount of data is available. The data is split into n "folds." We run our algorithm n times, using each fold as the test data exactly once with the remainder of the data used to train. We evaluate the error for each set of test data.

Since there are only 59 heroes in the game (a relatively small number of data points), we used cross validation to find an appropriate value of k for k-means clustering.

3 Work Done

For our project, there were two main facets that we spent the most time on, preparing the data and engineering features, and writing code to actually train and test the algorithms we use.

3.1 Preparing Data

As we stated in our project proposal, we obtained all of our data from HotsLogs, an enthusiast-run data collection website for *Heroes of the Storm*.

When we examined professional games, we realized that we only had 326 games to work with, and these games were played over a long period of time. Heroes will sometimes be adjusted to try to make them more "balanced" with respect to the other characters, so too much time passing could cause game results to become less relevant as heroes are changed.

Keeping this in mind, we chose to invest more time into looking at games from the top 100 players in the last month. To do so, we used the *BeautifulSoup* python package from *bs4*, along with *requests* and *lxml* to scrape HotsLogs and convert the data into vectors that we could consume. We chose the top 100 players so we could make the assumption that the players in each game were of relatively consistent skill level. We believe there was some sort of flaw in how we scraped or vectorized this data, as we ended up with basically 100% accuracy in our predictions.

A week after we started our project, the lead developer of HotsLogs released an API endpoint that would allow us to download all games that had been uploaded to the site in the last 30 days in csv format. We took advantage of this dataset, containing over 2 million games, and attempted to parse it and create vectors of relevant features from it. We encountered a lot of problems here on a computational level. There was enough data that all computers available to us took an unreasonable amount of time to process the data. *scikit-learn*'s implementation of SVM took over 12 hours to train on part of the dataset.

Because of this, we decided to use a strict subset of these 2 million games. Because there are various modes that you can play in *Heroes of the Storm* we stuck to "Hero League", which is the most popular competitive mode. We further restricted the data by removing all games from non-high-skill players, players whose matchmaking ranking (MMR) was below 3200. This left us with approximately 9000 games, a dataset of a more reasonable size. Since we now had plenty of games, we discarded those that we had attempted to scrape directly from the website.

In the replay data, the winning team was always noted first. To avoid having "team 1" always win, we decided which team would be team 1 based on the parity of the unique replay ID in the HotsLogs data. This should not have added any bias to the data, as this ID is completely independent from any data in the replay.

We were concerned that this might cause issues with predictions in that we could have identical team compositions and map (one in train data, one in test data) but with the teams swapped the algorithms might not pick up on the useful information of the "identical" point. To account for this, we tried adding both "versions" of each game (i.e. the version where the winning team was team 1 and the version where the winning team was team 2). However this did not seem to have any meaningful effect on our prediction accuracy.

We were able to repurpose the test/training harness from class and modify it to process our feature vectors and run various scikit-learn algorithms. We also used a shell script to run the algorithms and check them on various datasets.

Our pipeline for processing the professional game data was as follows: First, we used a script to pull JSON data from the HotsLogs API, and then directly parsed that data into usable vectors. For our scraped data, we first needed to scrape the data from the website, then process the scraped data into a machine-readable format, then turn the transformed data into vectors.

For the large CSV formatted data, we attempted to parse it all at once and turn the data into something we could work with, but the script that we tried to use for this ran all of the computers we had access to out of memory. To address this we selected only games in the CSV that were "hero league" games. This reduced the size of the dataset to approximately 600,000 games. We then scraped this dataset to pick out only high-level games which left us with a final size of approximately 9000 games.

3.2 Algorithms

We had originally planned to use a mix of *scikit-learn* algorithms and our own algorithms that we developed for class, but in the end, we decided to stick solely with *scikit-learn* for efficiency and consistency. We used *sklearn.neural_network.MLPClassifier, sklearn.linear_model.LogisticRegression, sklearn.neighbors* and *sklearn.svm.SVC* for directly processing the data, and *sklearn.cluster.KMeans* for creating our clusters. We ran all of these on vectors that we created using other Python scripts. All of these algorithms were encased in a test harness, and we used an automated bash script to run each algorithm on each different feature.

3.3 Feature Engineering

Feature engineering took a significant amount of time and work in our project. We used the map, average MMR for each team, the heroes themselves, the in-game classifications of heroes, and subclass labels for heroes.

In HotS there are 59 different heroes and 11 different battlegrounds. As mentioned above, each of these heroes is divided into a "class" based on what role they are supposed to fill in a game. These classes, however, by no means tell the full story of how each hero performs in game. As a result we opted to look at more detailed classifications. We had three different sources of subclass labels we applied: one set created by HotsLogs, one set created by us by hand based on our knowledge of the game, and one set generated from hero statistics using k-means clustering (discussed below).

We had one feature for each map and used a binary encoding: 1 for the map the match was played on and 0 for all others. Similarly, we had two features for each hero corresponding to "team 1" and "team 2." The appropriate feature was 1 if the hero was on the team in question and 0 otherwise. We also had two features for each in-game classification and each subtype (one per team) and set the value to how many heroes of that type were on the corresponding team. This resulted in very large feature vector, though not all features were always in use as we looked at different combinations of features: using and not using individual heroes, using and not using in-game classifications, and using one (or none) of the subtype classifications.

The size of the feature vectors when using individual hero identifiers (accounting for 118 features in total plus any additional features we applied) gave us another motivation behind investigating subtype classification of heroes. Using in-game classification or one of the three subtype classifications without individual hero identifiers, we could reduce the number of features needed to communicate the hero choices to as few as 16, a much more manageable number. We hoped that the information lost by not indicating individual heroes would not make a significant impact on our prediction accuracy.

3.4 Hero Clustering

To sort heroes into clusters, we created a feature vector for each hero. We used 9 features pulled from the average hero statistics on HotsLogs (Takedowns, Kills, Deaths, Hero Damage, Siege Damage, Healing, Self Heal, Damage Taken, and XP) along with a feature meant to model mercenary camps taken by each hero. This final feature was not available directly from replay data or the statistics page; instead we manually entered values as found on the match awards page. We used a percent corresponding to how often a hero received an award for most mercenary camps taken in games where the award was given (different awards are given each game).

First, we wrote code to normalize these features so they all fell in the range $0 - 1000$. This ensured that the features were weighted equally when looking at Euclidean distance from cluster centers. This was necessary since values for Kills and Deaths are between 0 and 10, whereas Siege or Hero Damage numbers tend to be in the tens of thousands, meaning without normalization differences in damage would be considered much more significant than differences in kills or deaths.

We also wrote code to perform a split of the data into train/test partitions for cross validation, along with code to test for an appropriate k-value using cross validation. The actual k-means algorithm we used was pulled from the scikit-learn package.

When determining k, we looked at potential values in increasing order. For each possible k value we performed n-fold cross validation, taking the average error across all n test sets as our overall error value. We only switched to a new k value if the amount of error was at least ten percent less than our current k value. Additionally, if we ever saw an error value at least five percent more than our current k value, we assumed that overfitting had begun to take place and stopped our search. We added these restrictions due to the fact that strictly taking the k-value with minimum error would almost always result in using the maximum k-value tested.

One issue we ran into was that Damage Taken stats were only recorded for heroes classified as Warriors. As a result, there was an unintended restrictions that warriors were segregated from other classes in clusters containing only other warriors. For the most part, this still seemed to result in reasonable clusters (based on hotslogs subclasses and our own game knowledge). The main question is whether some clusters might have been combined were this stat available for all heroes.

Another issue had to do with determining k. The random seeding of the initial cluster centers meant that not only were we getting different clusters each time we ran the algorithm, we would sometimes get different k values entirely. The majority of runs would result in $k = 9$, but unusual runs had as few as 6 or as many as 14 clusters (where the k values tested ranged from 1 to 15).

We selected an output we felt was fairly representative of the typical clusters we were getting with $k = 9$ and used this for our subtype classification when running our win/loss predictor. This particular set of clusters is included in the appendix.

3.5 Draft Predictor

One of our stretch goals was being able to offer advice to a user who is trying to determine the best hero to choose at some point in the drafting process. Essentially, this requires us to answer the question: what choice of hero(es) to complete the team will give the best chance of victory? We ended up implementing this for a small subset of cases, in that the map has been chosen, 4 heroes on team 1 have been chosen, and 5 heroes on team 2 have been chosen. The predictor then exhaustively searches for the HotsLogs subtype with the highest probability of winning and returns it.

4 Results

Our main goal in this project was to be able to predict the outcome of games based on features that could be determined before the game began.

Our best accuracy in this came from using Logistic Regression with the heroes, battleground, and our k-means generated clusters as features, at 55.63%. Other notable results were Logistic Regression with heroes, in-game roles, battleground, and either our handmade groupings or the k-means generated clusters, both at 55.52%. Overall, we ran 60 different tests, each test with a different algorithm or different combination of features on the same high-MMR dataset. The results from all of these is listed in the appendix.

Our best results were significantly lower than expected, we had hoped to reach 60% or higher correct prediction rate. If we had access to data it would be interesting to run a “player type” analysis as seen in (**?**). If we achieve similar accuracy to their results, this would imply that players and their preferences in heroes play a more significant role in the outcome of games than the heroes chose. If, on the other hand, we achieved similar accuracy to what we found here, it might tell us that hero selection may just be less influential in the outcome of games in HotS relative to League of Legends or DOTA 2. If this is the case, it could explain why k-NN did not perform particularly well.

Additionally, we assumed in advance that the decision boundary for this problem would be non-linear. However, logistic regression and the linear SVM were the best performers, with k-NN behind them and MLP behind k-NN. The algorithms that best predicted the data both did so on a linear separator, and the worst predictor of games turned out to be our universal function approximator, MLP.

Despite this, we still hypothesize that the decision boundary is non-linear, though there may be a significant part of the data that can be sectioned off by a linear boundary. This could explain why the linear algorithms performed well, while the actual boundary may be sufficiently complicated (perhaps non-convex) that the MLP was unable to accurately approximate it.

In general, logistic regression was the best predictor of games. We theorize that logistic regression performed well due to so-called "Over-Powered" heroes, heroes that have an abnormally high win rate (regardless of the rest of the heroes in the game). It is possible that these heroes being on one team could significantly increase that team's chance of winning, which would potentially mean the decision boundary was essentially fixed (and thus in a sense "linear") with respect that that specific feature. We think that the linear SVM could have worked well over all datasets because of this.

What features were used to describe the data also had a large impact of the accuracy of the predictions. We found that the most important feature was the heroes themselves as well as the battleground. We presume that this is due to the unique vulnerabilities and strengths of each hero, and the importance of specific battlegrounds to team composition.

The least important feature was the in-game classification of heroes. We hypothesize that this is due to the fact that these categories are very broad. This is expected, as the primary goal of the in-game classification system is to allow newer players to understand the basic roles of heroes in a very general sense (eg. this hero has more hit points, this hero heals teammates, etc.).

Overall, our results seem to show that while hero choices play a non-trivial role in the outcome of a game, there are many other factors in play affecting the game in potentially more meaningful ways.

5 Comparison to Proposal

- We successfully used logistic regression, SVM, k-nearest neighbors, and a neural network to attempt to predict which team won a game based on draft.

- We experimented with different combinations of features in addition to (or in place of) the heroes on each team, including three different types of "subrole" labels for heroes in addition to in-game labels.

- We successfully applied k-means clustering to find groups of heroes filling a similar role, though we did not always get the same results due to random seeding. We used the results from one run (that seemed relatively representative of the typical output) as additional features for our win/loss predictor.

- While we were able to pull replay data for non-professional games, we opted to focus primarily on high level games (average MMR 3200+ on both teams). This still gave us around 9000 games to work with and avoided concerns of different meta-games at different skill levels.

- We have a preliminary algorithm for recommending what hero could best fill out a draft. At this point though our algorithm is very weak, only being able to offer a subrole suggestion (based on hotslogs classifications) if the enemy team has picked all 5 of their heroes and your team has already chosen 4 heroes.

References

[Ong, Deolalikar, & Peng 2015] Hao Yi Ong, Sunil Deolalikar, and Mark Peng 2015. "Player Behavior and Optimal Team Composition for Online Multiplayer Games." http://arxiv.org/abs/1503.02230

[Pobiedina et al. 2013] N. Pobiedina, J. Neidhardt, M. d. C. Calatrava Moreno , L. Grad-Gyenge, and H. Werthner 2013. “On Successful Team Formation: Statistical Analysis of a Multiplayer Online Game.” *2013 IEEE 15th Conference on Business Informatics*.