How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2

Kevin Conley
Stanford University
Email: kcon@stanford.edu

Daniel Perry
Stanford University
Email: djperry@stanford.edu

***Abstract*—In this paper, we present a hero recommendation engine for the popular computer game Dota 2. We detail previous efforts at hero recommendation, building on these to provide a more complete exploration of machine learning algorithms applied to this problem.**

In addition to discussing the details behind the machine learning algorithms we use, we also provide insight into our method of data collection and feature selection. In doing so, the outcome of our efforts is two-fold. First, we provide the first public survey of machine learning algorithms applied to Dota 2–in the process gaining further insight into the algorithms we explore. Second, we provide a tool that we believe will be useful for both the casual and competitive sides of Dota 2's 6 million unique player base.

I. Introduction

Dota 2 by Valve is a popular computer game with a player base of 6 million unique users. While many of these users play casually, there are also professional players that participate in Dota 2 tournaments with massive monetary payouts. For example, Valve's recent Dota 2 tournament, "The International 3," had a prize pool of over $2.7 million [3].

Each Dota 2 match consists of two teams of five players pitted against each other. Before a match begins, each player selects a character to play as, known as a "hero," from a pool of 106 different heroes. Once a player chooses a hero, no other player can select that hero for the same match. Heroes have a wide-range of characteristics and abilities that, combined with the massive hero pool, make each match unique.

An interesting aspect of the game is that in choosing a hero, players must keep in mind not only the individual strengths and weaknesses of each hero, but also how the strengths and weaknesses of that hero interact with the heroes already chosen by other players. An effective hero pick is one that synergizes with the heroes chosen by teammates, and both exploits the weaknesses and minimizes the strengths of the heroes chosen by the opposing team. Assuming equally skilled teams, the ramifications of hero selection can be so staggering that well devised hero choices can implicitly give a team a large advantage before the match even begins. The goal of our project is to recommend heroes that will perform well against an opposing team of heroes.

This fits as a classic machine learning recommendation problem, but what piques our interest is the sheer volume of the solution space. With 106 heroes to choose from and five heroes per team, we are attempting to find the best five heroes for any given matchup, which results in over eight quadrillion possible team combinations. On a deeper level, recommending heroes using machine learning is challenging because it tries to capture via raw data what professional players have developed a gut instinct for through hundreds of thousands of hours of play time.

II. Related Work

Dota2cp [1] is a web application developed with a similar aim of recommending heroes for Dota 2 matches. Given two teams of five heroes, the author reports a 63% accuracy of predicting the winning team. Although the author does not release source code for his algorithm, he states that Dota2cp models hero selection as a zero-sum-game for which it learns the game matrix by logistic regression. When suggesting hero picks and bans it assumes that teams are min-max agents that take turns picking one hero at a time. While Dota2cp is a great first effort at a hero recommendation engine, we believe we can improve on its results by exploring other machine learning algorithms.

Dotabuff [2] is a website that provides detailed Dota 2 statistics. Dotabuff uses the same web API that we do to collect data, and it allows the user to visualize statistics in an organized manner. While the website currently makes no effort to provide hero recommendations, we found it helpful in verifying the data we collect.

III. Dataset

We used Valve's Steam web API [4] to pull data for 56691 matches between November 5 and December 7. Our data satisfies the following requirements:

- The game mode is either all pick, single draft, all random, random draft, captain's draft, captain's mode, or least played. These game modes are the closest to the true vision of Dota 2, and every hero has the potential to show up in a match.

"How does he saw me?" is an infamous quote from Dota 2 sportscaster David 'Luminous' Zhang, who abandoned English during a particularly exciting Dota 2 match: http://www.youtube.com/watch?v=BAC9B9z_M0s&t=52m6s

- The skill level of the players is “very-high,” which corresponds to roughly the top 8% of players. We believe utilizing only very-high skill level matches allows us to best represent heroes at their full potential.
- No players leave the match before the game is completed. Such matches do not capture how the absent players’ heroes affect the outcome of the match.

The data for each match is structured as JSON and includes which heroes were chosen for each team, how those heroes performed over the course of the game, and which team ultimately won the game. We stored the JSON for each match in a MongoDB database during data collection.

We exported 90% of the matches from our database to form a training set of 51,022 matches. We exported the remaining 10% of our database to form a test set of 5,669 matches.

IV. Methodology

In this section we describe the algorithms we used in developing our recommendation engine. We used the scikit-learn Python library to implement our algorithms.

A. Feature Vector

In Dota 2 matches, one team is called the ”radiant” and the other team is called the ”dire.” These terms are roughly analogous to ”home” and ”away”, as they only determine the starting point of each team on the game world map, which is roughly symmetric.

There are 106 heroes in Dota 2 (as of writing), but the web API uses hero ID numbers that range from 1 to 108 (two hero IDs are not used), so for our algorithms we used a feature vector $x \in \mathbb{R}^{216}$ such that:

$$x_i = \begin{cases} 1 \text{ if a radiant player played as the hero with id } i \\ 0, \text{ otherwise} \end{cases}$$

$$x_{108+i} = \begin{cases} 1 \text{ if a dire player played as the hero with id } i \\ 0, \text{ otherwise} \end{cases}$$

We also defined our label $y \in \mathbb{R}$ to be:

$$y = \begin{cases} 1 \text{ if the radiant team won} \\ 0, \text{ otherwise} \end{cases}$$

B. Making Predictions

Since our dataset contains information about heroes on teams in specific radiant vs. dire configurations, simply running our algorithms on each match in our dataset does not fully utilize all of the data.

Instead, we make predictions using the following procedure. Given a match feature vector, which we call `radiant_query`:

1) Run the algorithm on `radiant_query` to get `radiant_prob`, the probability that the radiant team in `radiant_query` wins the match.
2) Construct `dire_query` by swapping the radiant and dire teams in `radiant_query` so that the radiant team is now the bottom half of the feature vector and the dire team is now the top half of the feature vector.
3) Run the algorithm on `dire_query` to get `dire_prob`, the probability that the radiant team in `radiant_query` loses the match if it was actually the dire team instead.
4) Calculate the overall probability `overall_prob` as the average of `rad_prob` and $(1 - \texttt{dire\_prob})$: $\texttt{overall\_prob} = \frac{\texttt{rad\_prob} + (1 - \texttt{dire\_prob})}{2}$.
5) Predict the outcome of the match specified by `radiant_query` as the radiant team winning if $\texttt{overall\_prob} > 0.5$ and as the dire team winning otherwise.

This procedure accounts for matches in our dataset that might not have the team configuration of a given query in one direction (e.g. radiant vs. dire) but may have the configuration in the other direction (e.g. dire vs. radiant).

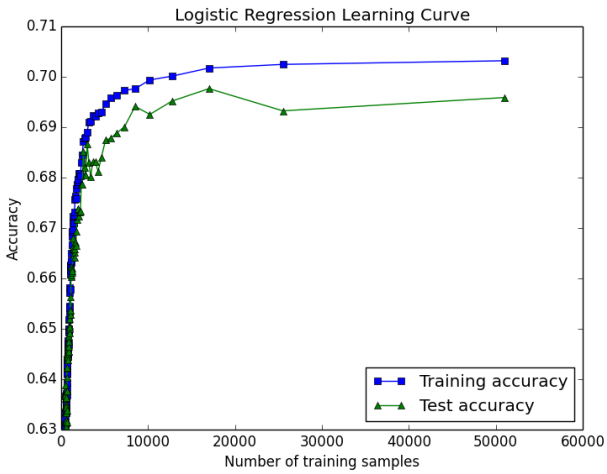

Fig. 1. Logistic regression learning curve

C. Logistic Regression

Logistic regression is a model that predicts a binary output using a weighted sum of predictor variables. We first trained a simple logistic regression model with an intercept term to predict the outcome of a match. The feature vector and label we used is decribed in part A of this section.

1) Learning Curve: A plot of our learning curve for logistic regression is shown in Figure 1. The training and test accuracies are quite close together, indicating that our model does not overfit the data. The test accuracy of our model asympotitcally approaches 69.8% at about an 18,000 training set size, indicating that 18,000 matches is an optimal training set size for our logistic regresion model.

2) Analysis: We believe that this logistic regression model shows that hero selection alone is an important indicator of the outcome of a Dota 2 match. However, since logistic regression is purely a weighted sum of our feature vector (which only indicates which heroes are on either team), logistic regression fails to capture the synergistic and antagonistic relationships between heroes.

D. K-Nearest Neighbors

K-nearest neighbors is a non-parametric method for classification and regression that predicts objects' class memberships based on the k-closest training examples in the feature space. We used a custom weight and distance function and chose to utilize all training examples as "nearest neighbors." Our polynomial weight function described below aggressively gives less weight to dissimilar training examples. The feature vector and label we used is described in part A of this section.

We chose to implement K-nearest neighbors in order to better model the relationships between heroes instead of simply taking into account wins when a hero is present. At a high level, we continue to focus on wins and the hero composition of teams, however with K-nearest neighbors, we have an avenue to weigh matches according to how similar they are to a query match we are interested in. For example, if we are interested in projecting who will win a specific five on five matchup (our query match), a match with nine of the heroes from the query match present will give us more information on who will win the query match than a match with only one hero from the query match present.

1) Calculating Weights: The following equation represents a combined distance and weighting function used in our K-nearest neighbors simulations:

$$w_i = \left(\frac{\sum_{j=1}^{216} AND(q_j, x_j^{(i)})}{NUM\_IN\_QUERY}\right)^d$$

The d parameter represents the polynomial scaling of the weight function. x represents the feature vector for training match i and is compared by the logical *AND* operator to query vector q. Index j represents the hero ID index of each respective vector. *NUM_IN_QUERY* represents the number of heroes present in the query vector.

The function is normalized to be between 0 and 1, and it gives more weight to matches that more closely resemble the query match. To do this, the function compares the query match vector to the training match vector and counts every instance where a hero is present in both vectors.

A larger d will result in similar matches getting much more weight than dissimilar matches. Alternatively, a low d, for example $d = 1$, will result in each match being weighted solely by how many heroes in common the match has with the query match. Stated another way, a high d will choose to put more emphasis on the synergistic and antagonistic relationships between heroes, while a lower d will put more emphasis on the independent ability of a hero.

2) Choosing an Optimal Weight Dimension: To choose the optimal d dimension parameter described above, we used k-fold cross validation with $k = 2$ on 20,000 matches from our training set and varied d across otherwise identical K-nearest neighbors models.

Since K-nearest neighbors must compare the query match to every match in the training set and compute weights and probabilities, this process was quite slow and took about ten hours. Due to time constraints, using more folds or more matches would have taken too long to finish.

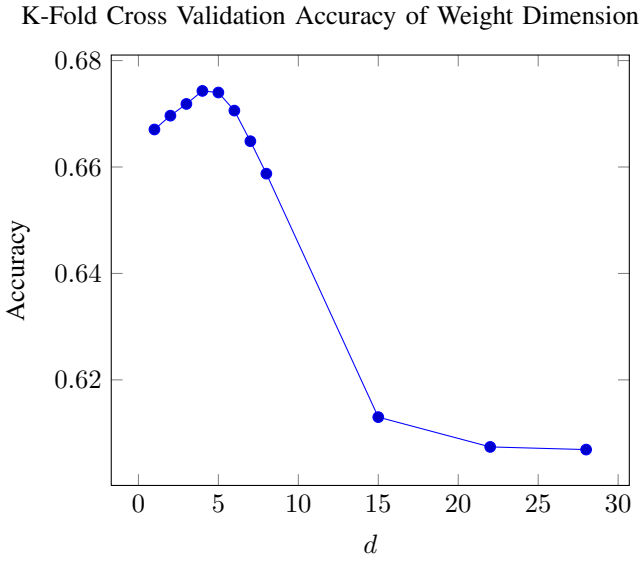

Fig. 2. Choosing an optimal d dimension for our weight function

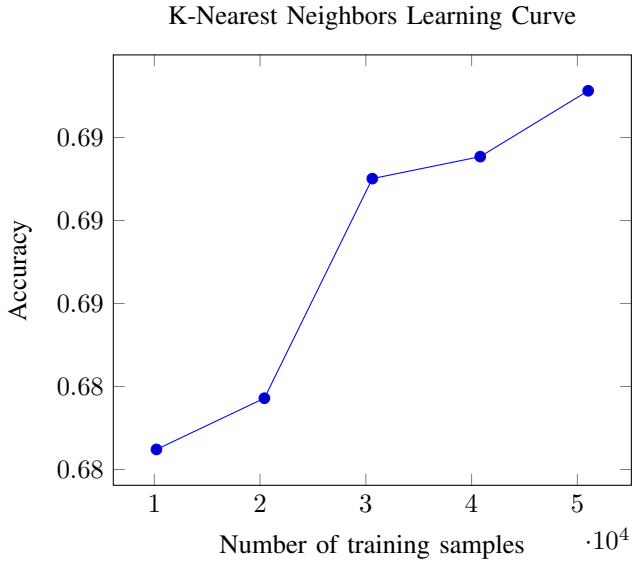

Fig. 3. K-nearest neighbors learning curve

A graph of the accuracies achieved when varying the weight dimension for a K-nearest neighbors model trained on our training set and evaluated on our test set is shown in Figure 2. We found the optimal weight dimension to be $d = 4$, which achieved a mean accuracy of 67.43% during the k-fold cross validation.

3) Learning Curve: A plot of our learning curve for K-nearest neighbors is shown in Figure 3. The test accuracy of our model monotonically increases with training set size up to nearly 70% for around 50,000 training matches. Because we do not see the learning curve level off, this may imply that more data could further improve our accuracy.

E. Recommendation Engine Architecture

We recommend heroes for a team using a greedy search that considers every possible hero that could be added to the

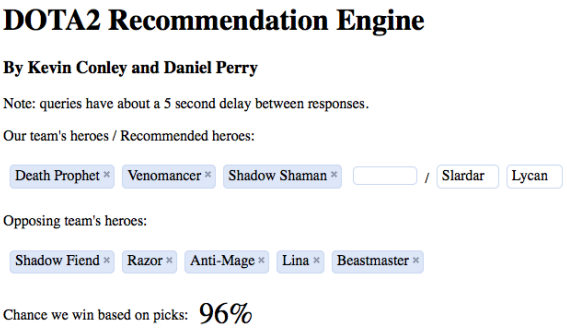

Fig. 4. Screenshot of web interface for recommendation engine

team and ranks the candidates by probability of the team winning against the opposing team if the candidate was added to the team.

Our recommendation engine is modular so that either of our algorithms could be used to recommend heroes. Given the ID numbers of the heroes on both teams, the recommendation engine proceeds as follows:

1) Create a feature vector for the match as described in part A of this section.
2) Create a set of new feature vectors, each with a different candidate hero added to the original feature vector.
3) Run the algorithm to compute the probability of victory with each feature vector from step 2, averaging the probabilities of the team winning in both radiant and dire configurations as described in part B of this section.
4) Sort the candidates by probability of victory to give ranked recommendations.

F. Web Interface

To facilitate the use of our recommendation engine for users, we adapted the data entry interface from the Dota2cp website so that the user can easily input heroes by typing the first few letters of a hero's name, choosing the desired hero from a list of auto-complete suggestions, and pressing enter. We connected this web interface to our recommendation engine using the Flask Python web application library. A screenshot of our web interface is shown in Figure 4.

V. Conclusion

In this paper we present a survey of machine learning algorithms applied to the challenge of building a hero recommendation engine as well as predicting match outcomes for the game Dota 2. We apply logistic regression and K-nearest neighbor models to these challenges and achieve promising results.

Given our logistic regression results, we conclude that solely looking at the hero composition of a team and whether or not that team was victorious can provide a useful model for match outcome prediction and as a result, hero recommendation. Additionally, utilizing a K-nearest neighbors approach with custom weight and distance functions opens up another path towards achieving successful match predictions and hero recommendations. We see that by choosing a moderate d value we can achieve an appropriate balance between recognizing the individual skills of heroes as well as factoring in the synergistic and antagonistic relationships between them. Our results also suggest that using more data could improve our K-nearest neighbors results further.

VI. Future Work

Despite the high accuracy of our K-nearest neighbor model, the performance was quite slow. The five data points we used in our learning curve took about four hours total to calculate, and the k-fold cross validation used to find the optimal weight dimension took over 12 hours.

We believe that the performance of our K-nearest neighbor model could be improved in a number of ways. First, the binary feature vector used could be stored as an integer in binary representation to improve memory usage. Second, the calculation of weighted distances could be parallelized across multiple CPU cores. Third, a GPU could be utilized to vectorize the weighted distance function calculations.

Although the version of the game did not change during the time we collected match data, a new patch is released for Dota 2 every few months that dramatically changes the balance of the game. Therefore, we believe that a sliding window of match history data that resets when a new patch is released could help maintain data relevancy. For this reason, it would be useful to find if there is a training match size for which performance levels off so that we could collect only as much data as is needed.

Finally, we could experiement with a different search algorithm for our recommendation engine such as A* which would account for the opposing team picking heroes that could counter our initial recommendations. However, due to time constraints we were not able to implement A* search for our recommendation engine.

Ultimately, we believe there are many promising directions which future explorations in this area could take.

References

[1] "Dota2 Counter-Pick." Dota2 Counter-Pick. N.p., n.d. Web. 14 Nov. 2013. http://dota2cp.com/.

[2] "DOTABUFF - Dota 2 Statistics." DOTABUFF - Dota 2 Statistics. Elo Entertainment LLC, 2013. Web. 14 Nov. 2013. http://dotabuff.com/.

[3] "The International Interactive Compendium." Dota 2 Official Blog. Valve. 2013. Web. http://www.dota2.com/international/compendium.

[4] "WebAPI." Official TF2 Wiki. Valve, 16 Sept. 2013. Web. 14 Nov. 2013. http://wiki.teamfortress.com/wiki/WebAPI.