Predicting the winning side of DotA2

Kuangyan Song, Tianyi Zhang, Chao Ma

ABSTRACT

In this paper, we tried using logistic regression to predict the winning side of DotA2 games based on hero lineups. We collected data using API provided by the game developer. We find out that only based on hero lineup to predict the game result is not good enough. We also tried to select feature using stepwise regression and the result is better than using all the heroes and hero combos as features.

1 Introduction

DotA 2 is a very popular online game. A regular DotA2 match has ten players with five players on each side. Each player will choose a hero and their goal is to defeat their opponents. In order to win the game, there are two main aspects need to be considered. The first aspect is the competitive level of the players. The competitive level of a player here means not only the overall level, but also the level of playing a particular hero. The other aspect is the lineup of both sides. DotA 2 is such a game that players need to play as a team in order to win, which means that some heroes are designed to support others and some are designed to attack or defense. For example, if one team chooses five supporting hero, it is hard for them to win the game since no one is specialized to attack. Moreover, there are some combinations of two or three heroes whose spells can complement with each other so as to make it easier to defeat their opponents. Also there are restraints between two heroes.

2 Related Work

Dota2, due to its popularity, has drawn fair amount of attention in machine learning projects. In 2014, Atish and Michael did an project on DotA 2 team composition. They predicted winning side by looking at how both sides pick their heros including interactions between heros. However, they found out that after incorporating hero interaction, the prediction rate is lower than the model with out hero interaction. Their insight was that either the model needs to be fined or the data they collected were not good enough. Back to 2013, there was anothor project presented by Kevin and Daniel where they did an survey on the machine learning algorithm that had been applied to DotA 2 prodeiction and then made a recommendation engine that helps the player pick a hero to maximize winning rate.

3 Data Collection

Valve, the company who developed DotA2, provides Web API for developers to retrieve match and player data. We use a python library dota2api which is built on the Web API to collect the data we need.

3.1 Match Data

Since there are different mode of games, such as ”Ranked All Pick”, ”All Pick” and ”Solo Mid”, some of which are not regular DotA modes, we need to filter out those games that are not useful in our prediction. In our case, we only consider those games that is ”All Pick”, ”Ranked All Pick” and ”Random Draft”. Moreover, those games that has absent players or has zero kills through out the game are considered to be invalid. We first tried to filter out the invalid games by setting the arguments of API calls, but the API seems to be broken so that it ignores those filter arguments since Dota2 has changed its engine. We then collected all the match detail data and filter out the invalid match by ourselves. These match data are stored as json files for future use.

3.2 Player Data

The player data are more complex and more difficult to collect. For each single game, there are ten players involved so that we need to use the API to retrieve all the players’ data. In our case, we just request the ”Match History” information for a particular player instead of requesting the ”Match Detail”. We collect about one hundred matches’ history for a single player. In addition, some of the players might choose to make the match history private so that we cannot collect every player’s match history. The match history will contain the information of the hero that the player used and the match id of that game. These data are also stored as json files for future use.

4 Methodology

4.1 Baseline

4.1.1 Feature Selection and Extraction

DotA2 has a player matching system that will automatically match the players that have similar competition level (also called "matchmaking ranking") together to have one game. Thus, we first make an assumption that all the players in a single game should have similar competition level. By assuming this, we are also assuming that a player can play every hero equally well. This is very strong assumption since there are now 110 playable heroes in DotA2, hence it is highly possible that a player is not familiar with the hero that he used in a particular game. Moreover, the players' competition level might be different. This assumption, however, allow us to simply the problem to only consider the hero selection in a game. We could just use the hero selection data to predict the game result and see how hero selection contributes to the final result.
Under such assumption, we define the feature vector as

$$\phi(X) = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_{112} \\ X_{113} \\ X_{114} \\ \vdots \\ X_{224} \end{pmatrix}$$

Since each DotA2 game has two sides, one is "Radiant" and the other is "Dire", the first half of the feature vector $(X_1 \dots X_{112})$ represents the "Radient" side while the second half of the feature vector $(X_{113} \dots X_{224})$ represents the "Dire" side. For the first half of the feature vector:

$$X_i = \begin{cases} 1, & a\ player\ of\ radiant\ side\ plays\ hero\ i \\ 0, & no\ player\ of\ radiant\ side\ plays\ hero\ i \end{cases}$$

And for the second half of the feature vector:

$$X_i = \begin{cases} 1, & a\ player\ of\ dire\ side\ plays\ hero\ i-112 \\ 0, & no\ player\ of\ dire\ side\ plays\ hero\ i-112 \end{cases}$$

We label each match with

$$Y_j = \begin{cases} 1, & Radiant\ side\ won \\ 0, & Dire\ side\ won \end{cases}$$

4.1.2 Logistic Regression

We used logistic regression on the data set and did K-fold cross validation. The logistic regression minimizes the following cost function:

$$min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{m} log(exp(-Y_i(\phi(x)_i^T w + c)) + 1)$$

For the first attempt, we collected in total 3000 matches and divided them to ten folds. So for each fold, we trained the model using 2700 matches and validated using the other 300 matches.
When predicting the winner of the game, we compute the probability of the each label ("Radiant Win" or "Dire Win") for a given match, and then simply choosing the one with larger probability as its prediction.
We used the python module sci-kit to do the logistic regression. After running logistic regression, we got the following results:

Table 1. Training error and testing error of Logistic Regression (3000 matches, 10 folds)

Fold	1	2	3	4	5	6	7	8	9	10
ε_{train}	0.28	0.29	0.30	0.27	0.29	0.25	0.23	0.29	0.33	0.31
ε_{test}	0.47	0.43	0.49	0.42	0.48	0.47	0.42	0.46	0.43	0.46

According to the above result, the training error is nearly 28% while the testing error is much higher (nearly 45%) which is not acceptable to predict the final result of a given match.

4.1.3 Problems for baseline

In this simplified model, we made a strong assumption for the players and we mainly focused on the hero selection. Since the testing error is high, it might be not enough just using hero information of a given match to predict the final result of the match. In other words, we need the player information to better predict the match result.

Moreover, by looking into the details of prediction process under the model of logistic regression, we found out that there are lots of matches that has nearly equal probabilities for both side (e.g. a match might have a probability to be classified as "Radiant Win" with a probability of 53%, and as "Dire Win" with a probability of 47%). In other words, these data points are too close to the decision boundary so that it is hard to classify such matches. In our case, we just simply choose the label that with higher probability, which might lead to prediction error. Thus, we need more information to better classify the match.
Another problem for the first attempt is the feature selection. We just considered the heroes that are used in a game. An intuitive idea that logistic regression gives us is whether a hero is strong or weak. It is true that certain heroes are stronger than the others, but the interaction between heroes are also very important for the game result. Thus, we need to add features that represent the interaction between two or three heroes. Moreover, certain heroes have significant effect on the results of matches, so we could select the heroes that have more effect.

4.2 Adding features for hero combo

Since heroes have interaction between each other, we decided to add features for interactions between two heroes. In particular, we will add features such as two heroes are both in radiant side or dire side. Since there are too many heroes and too many possible hero combos, we use our knowledge to specify some most powerful hero combos. We added fifty powerful two-hero combos to the feature vector and run logistic regression and got the following results:

Table 2. Training error and testing error of Logistic Regression (6000 matches, 10 folds)

Fold	1	2	3	4	5	6	7	8	9	10
ε_{train}	0.25	0.26	0.23	0.28	0.25	0.25	0.24	0.27	0.29	0.28
ε_{test}	0.40	0.41	0.45	0.43	0.41	0.39	0.41	0.43	0.40	0.42

According to the results, we can see that the training error and testing error is smaller than baseline. But the effect is not that much. Also the decreasing of error might because of the increasing number of match data.

4.3 Feature selection

Since there are too many heroes, some of them might not have much effect on the result of matches. We need to select features that has significant effect on the results. In order to do that, we use stepwise regression. Basically, the idea is to start with no feature, and add one feature that increase the model most at each step.

The algorithm is shown below:

Algorithm 1 Stepwise regression

RequiredNumberOfFeatures $\leftarrow$ some number
FeatureSet $\leftarrow \{1\}$ ▷ Initialize the feature set with just interception
while NumOfFeatures $\neq$ RequiredNumberOfFeatures **do** ▷ End algorithm when reaching the required features
    Fit a model using FeatureSet
    Calculate $\frac{\partial L(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i(y_i - P(x_i;\beta))$
    $\beta_{new} \leftarrow \arg\max(|\frac{\partial L(\beta)}{\partial \beta}_i|)$
    Add the feature that corresponding to β_{new} to feature set
    NumOfFeatures $\leftarrow$ NumOfFeatures + 1
return FeatureSet

Using this algorithm, we can select those principal features (heroes and hero combos). In the meantime, the total number features will decrease, hence it might help to decrease error.

Another problem is how to choose the number of features that should be added to feature set. In order to figure out this, we

enumerate the number of features and run the above algorithm for each of the enumerated numbers. The results for the each trial are shown in the figure 1.

According to figure 1, we can see that the training error and testing error are large at first since there are not enough features. As the number of features grows, the error rate goes down. When the number of features is around 60 to 70, the error rate is the lowest. It shows that some heroes do have such principal effect on the result of matches while adding some heroes to the features might cause the prediction go wrong. Also, the result might relate to the data we collected.

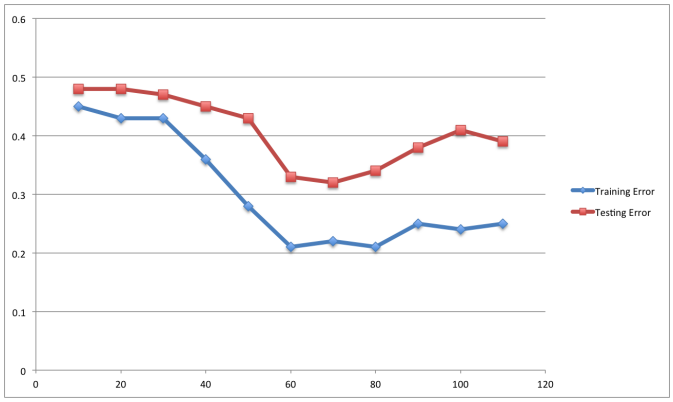

Figure 1. Error v.s Number Of Features

5 Future work

According to our work, we can see that predicting the winning side based on hero selection only based on hero lineups does not seem to have a good resul. Future work should take the player into account to make the prediction more accurate. Moreover, the model might need to be changed to get a better result.

References

1. Kevin Conley, Daniel Perry. *How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2*, 2013

2. Atish Agarwala, Michael Pearce. *Learning Dota 2 Team Compositions*, 2014