Recommendation Engine for selecting heroes in Heroes of Newerth - a multiplayer online battle game.

Jai asher
asher.j@husky.neu.edu
Northeastern University

Akshaya Khare
khare.ak@husky.neu.edu
Northeastern University

Abstract

We will create a recommendation engine for selecting characters in MOBA games specifically Heroes of Newerth using the logs generated for games played between January 2015 to November 2016. We address the need of this particular solution and the complexity of the problem we are trying to solve.

We then proceed to discuss the method of extracting, storing and utilizing the data. Then we discuss the steps involved in preprocessing the data - splitting the data into training and testing, and selecting the feature vectors for applying them on the three major machine learning algorithms used. We then discuss the three algorithms, their results, and analyse all the approaches. We end with possible improvements and different directions for future work.

1. Introduction

Multiplayer online battle arena (a.k.a. MOBA) originated as a subgenre of real-time strategy, in which a player controls a single character (Hero) in one of two teams. The objective is to destroy the opposing team's main structure with the assistance of a few periodically spawned computer-controlled units that march forward along set paths. Heroes of Newerth (a.k.a. HON) is one of the leading games of this genre, the game consists of around 249 characters, each with its own unique abilities and each match consists of two teams of five players each controlling a unique hero.

1.1. Related Work

There has been some work done previously on MOBA game but they were restricted to just Defense of the Ancients (a.k.a. DOTA). Algorithms like Logistic Regression, K Nearest Neighbors, Principle Component Analysis and Random Forest were used to get the best results for the DOTA games. None of these previous work addressed other MOBA games like HON or League of Legends. We decided to tackle the recommendation for HON and provide a summary of our results.

1.2 Background

Heroes of Newerth is one of the fastest growing games, generating a revenue of more than 100 Million dollars. Every year there is a tournament with prize pool of over 3 Million dollars, we would like to create a recommendation engine which would help the players choose a good mix of characters, which complement each other and increase the team's chances of winning.

2. Problem Addressed

Each player needs to choose from 110 characters, each with:-

a. 4 unique abilities
b. Different Strength, Agility and Magic points
c. Different Max Health and Mana
d. Different Damage, Shield and Attack Speed
e. Different Movement speeds
f. Different attack ranges – Melee vs Ranged
g. Different level up abilities – Carry vs Support vs Tank

Hence for each selection there are at least 47520 things to consider, which is a very difficult task for a human being to perform in real time. In reality, the player most of the time goes with his previous experience or his gut feeling.

3. Dataset

We used *HON Statistics API* to pull the data of over forty thousand matches played between January, 2015 and November, 2016. We filtered our data to satisfy the following criteria:

a. The game mode is *Forest of Caldavar*, which is the most popular game mode.
b. No player fled the game before completion.
c. The game is ranked, hence poor performance leads to penalty.

Match details were fetched from the Web API using *Nodejs*. The data was filtered and normalized before persisting in the Database. MongoDB was the obvious choice for database since the data was in JSON format. Below is the Mongoose schema used for storing the data.

```
{
        matchId: String,
        teamOneHeroes: [{type: String}],
        teamTwoHeroes: [{type: String}],
        winner: Number
}
```

Once the data was retrieved from the Web API we divided it into two parts:

i. 90% - Training Dataset
ii. 10% - Test Dataset

4. Algorithms

This section describes the feature vector selection and various algorithms used to make the recommendation.

4.1 Feature Vector Selection

In Heroes Of Newerth there are two teams, Legion and Hellbourne, each consisting of five players. The team determines on which side of the map you will start your game. The player gets to choose a hero from two hundred and forty nine heroes, each hero has it's own id ranging from one to two hundred and forty nine. For Logistic regression our feature matrix consisted of four hundred and ninety eight features:-

$$x_i = \begin{cases} 1 \text{ if a Legion player has chosen the } i^{th} \text{ hero} \\ 0 \text{ if a Legion player hasn't chosen the } i^{th} \text{ hero} \end{cases}$$

$$x_{i+249} = \begin{cases} 1 \text{ if a Hellbourne player has chosen the } i^{th} \text{ hero} \\ 0 \text{ if a Hellbourne player hasn't chosen the } i^{th} \text{ hero} \end{cases}$$

Our output label is defined as-

$$y = \begin{cases} 0 \text{ if the Hellbourne team won the game} \\ 1 \text{ if the Legion team won the game} \end{cases}$$

4.2 Algorithms

In this section we discuss the different algorithms used and analyse their outcomes.

4.2.1 Logistic Regression

Given a set of heroes in the opposite team and an incomplete set of heroes from your team, we use logistic regression to recommend heroes which will go well with the given combination and increase the player's chance of winning the game.

4.2.1.1 Creating Prediction Model

We don't recommend a single hero, instead we recommend a set which is best suited to complete the input combination, hence we need to do the go through a series of steps to get the desired result:-

a. We create a set of possible teams by adding one hero at a time to our team.
b. For every possibility:

- ❖ We create a Legion Team query to check the probability (legion_win_prob) of winning if we played as a Legion.
- ❖ We create a Hellbourne Team query to check the probability (hellbourne_win_prob) of winning if we exchanged our heroes with the Hellbourne team.
- ❖ We calculate the probability of winning as

$$prob = \frac{\text{legion\_win\_prob} + (1 - \text{hellbourne\_win\_prob})}{2}$$

c. Once we have a set of these prob mentioned above we just return the list of N hero ids where N is the number of heroes needed to complete the set.

4.2.1.2 Creating Prediction Model

4.2.1.2.1 Accuracy

Using our model we were able to predict the outcome of the training data set with an accuracy of 72% and the outcome of our test data with an accuracy of 69%.

4.2.1.2.2 Optimal size of training set

A plot of learning curve for logistic regression, shown in *Fig. 1*, indicates that we are not overfitting the data since the difference between the training accuracies and test accuracies is minimal. Beyond ten thousand records the training and test accuracy are almost constant, indicating that ten thousand number of records is optimal for creating the model.

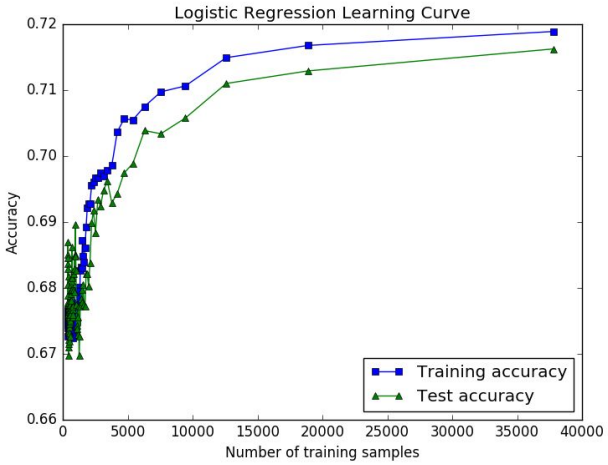

Fig 1. Logistic Regression Learning Curve

4.2.1.3 Conclusion for Logistic Regression

Our results show that hero selection is an important criteria in winning the game, since we are able to get an accuracy of 68% with just hero ids in the feature vector. However, since logistic regression is a weighted sum of our feature vector it does not capture the synergistic relationships between heroes, hence to find the set of heroes which work well together as a team we need to use K Nearest Neighbors.

4.2.2 K Nearest Neighbors

K Nearest Neighbors (a.k.a. kNN) is a non-parametric classification method which can predict the given item's class based on the k closest neighbors or training examples present in the dataset. We decided to choose kNN to identify the synergy between the *heroes*, this will help recommend the set of heroes which will work well together as a team and increase the team's chances of winning the game.

4.2.2.1. Creating Prediction Model

With kNN we can put weights to matches which are similar to the given input feature vector. For a match if we have ten heroes, it gives us more information on the possibility of winning the match as compared

to the impact of a single hero in overall matches. The steps followed are:-

a. We decided to use the sklearn's kNeighborsClassifier library[4] to create a model using the input data.
b. We divided the training and test sets in 90:10 ratio.
c. Using this training data, we created a model with neighbours being the length of the input vector.
d. We used this model to test our accuracy on the test dataset.

4.2.2.2. Runtime Complexity

The runtime complexity of kNN is O(KNM) where:

K = number of clusters/classes required

N = number of features

M = size of training data set.

The input matrix is very large - it consists of four hundred and ninety eight columns (features - N=498) and four thousand rows (number of records - M=40000), this makes the algorithm run very slowly. It takes almost ten hours to complete the whole process. In order to reduce the time, we used a multiprocess library to parallelize the process. We decided to take five threads to best optimize the processes. This helped reduce the running time from ten hours to under five hours - a *gain of almost 50%*.

4.2.2.3. Analysis

On running our model on the test dataset we get an accuracy of about 55.4% and on the training dataset we get an accuracy of about 58.4%, this shows that our models are not overfitted. As given in the *Fig 2*. KNN performs fairly better with the training set as compared to the test dataset. Although KNN should perform better than logistic regression, in this case, since the feature vectors are so huge, the algorithm fails to provide a good classification.

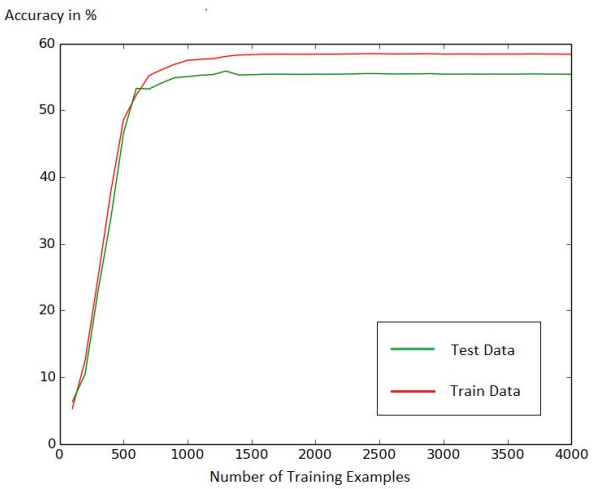

Fig 2. KNN Learning Curve

4.2.2.4 Conclusion for KNN classification

KNN doesn't fit the curve of good classification model. This model gives an accuracy of just over 55%, which is as good as random guessing. Increasing the number of training data samples may create a better model for K Nearest Neighbor considering the fact that we have almost four hundred and ninety eight feature in our input vector. We could also apply better weights on the input during the classification to get better results.

4.2.3 Random Forest Classification

Another approach we decided to take was to use Ensemble method - Random Forest Regression to recommend the hero. On some detailed research, we found that this algorithm can provide a much better performance as compared to other existing algorithms like Logistic Regression. Random Forest Regression runs efficiently on large datasets, since it is an ensemble algorithm it also scales well as the size of the dataset increases.

4.2.3.1 Methodology

We build different trees of data in this algorithm, with each tree built about one third of data is left out, this left out data is used to

get the unbiased estimate of the classification error as other trees are added to this forest. After each tree is constructed, all of the data is traversed through the tree and the algorithm calculates proximity for each computed case. Two cases occupying the same node increases their proximity by one. This proximity is normalized by dividing with the total number of trees and then used to apply to missing data.

4.2.3.2 Creating Prediction Model

We used the same feature vectors, training dataset and test dataset as K Nearest Neighbors. We used the standard RandomForestClassifier sklearn library[3] to compute the model. Since the interaction of the data increases with the increase in number of trees, we decided to use four different values for the number of trees, i.e.(1,5,10,20) and to compare the different results obtained with each value.

4.2.3.3 Analysis

After conducting the experiment, we found that Random Forest was much faster and gave accurate results as compared to K Nearest Neighbors. Although it was not as fast as logistic regression, its accuracy was better than logistic regression. With the increase in number of trees, the algorithm's accuracy increased at the cost of running time.

The accuracy obtained for various number of trees (1, 5, 10, 20) is shown in *Fig 3*.

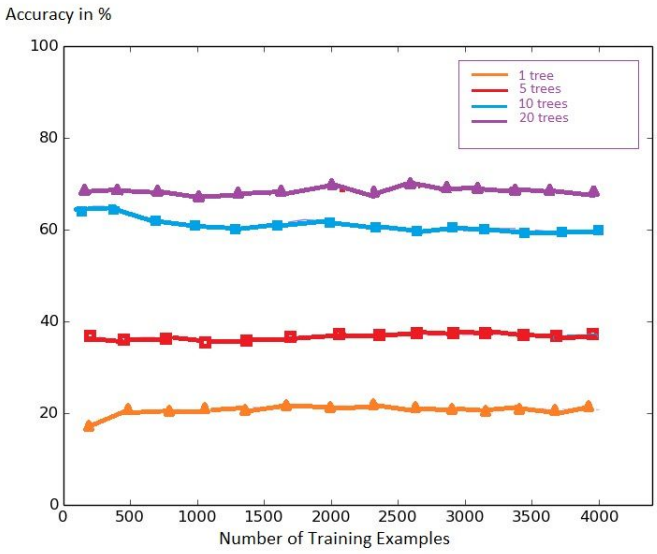

Fig 3. Random Forest change in accuracy with number of trees

Analysing the graph we believe the appropriate value of the number of trees should be somewhere between 10 and 20 to provide a balance between speed and accuracy.

5. Conclusion

Using Logistic Regression we have provided an accurate prediction of the possibility of winning for any of the team based on the *hero* chosen, with accuracy ranging from 71% to 67% for different algorithms.

Random Forest Algorithm accurately predicted the outcome of the matches in test dataset. K Nearest Neighbors was not able to predict the outcome with such high accuracy, but what it lacked in accuracy it made up in run time complexity.

Overall our accuracy didn't exceed 68%, this is majorly because there are other features we couldn't take into consideration because of the tight schedule we were in. Also, to get a high prediction accuracy, we will also need to consider the scenario of how the game is being played, since results vary drastically based on the choice of strategy used by the

gamers and other factors. Unfortunately that data is not available anywhere and it will be quite difficult to extract such data.

6. Future Work

In the future we would like to explore more algorithms to be able to compare the machine learning algorithms for Heroes Of Newerth and also to provide the best classification algorithm. We would also like to explore the possibility of different feature vectors and how to incorporate it in our algorithms,check the accuracy and compare the results across different feature vectors. This can be helpful to find which features are unimportant to be decisive for a player to choose.

7. References:

[1] Aleksandr Semenov, Peter Romov, Kirill Neklyudov, Daniil Yashkov, Daniil Kireev Applications of Machine Learning in Dota 2:Literature Review and Practical Knowledge Sharing

[2] Features of Random Forests https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[3] sklearn RandomForestClassifier library http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[4] sklearn KNeigborsClassifier library http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5] Book on efficiently using MongoDB http://openmymind.net/mongodb.pdf