DOTA 2 Game Prediction

Zhenyu Bi z1bi@ucsd.edu
Yu-Chun Chen yuc330@ucsd.edu
Chengming Dong ax006536@ucsd.edu
Rui Qin r3qin@ucsd.edu
Shuibenyang Yuan shy166@ucsd.edu

Abstract

As of now, eSport has emerged as a new kind of competitive sport. Its popularity has inspired attempts to predict game results by using machine learning techniques. Here, an evaluation of models and feature representations are presented to predict the results of DOTA 2, a well-known game in eSport currently. Models used include KNN Classifier, Logistic Regression, Random Forest, Multilayer Perceptron, and Naïve Bayes. The main goal is to reach an accurate prediction of DOTA 2 games with data we have.

1. Introduction & Motivation

DOTA 2 is one of the most successful games on Steam. According to statistic provided by Statistia, DOTA 2 reached its high peak of concurrent player number record in March 2016 at nearly 1.29 million gamers playing at the same time. It has also been in the list of the most popular games played in the world since it is released, and this trend still continues. Since DOTA 2 is a competitive online computer game, gamers like our group members are all very interested in whether we can produce accurate prediction of DOTA 2 games, as a successful prediction may suggest ways for us to win the game ourselves. However, winning a game is not merely about individual skills. A balanced and reasonable draft (different combination of heroes in one team) is very essential because there will be different styles of play according to different drafts. Thus, applying machine learning to predict the outcome of each game is intuitive for individual gamers to improve game experience and win more games. Furthermore, we can come up with a counter draft of opponents' draft timely based on the analysis of data during the draft stage.

Aside from the giant gamer group, DOTA 2 also has its own mature and booming esports industry. Similar to popular sports like basketball and football, DOTA 2 has its own league of different regions holding millions of games every year. The most famous and worldly watched game series is The International, DOTA 2 Championships. It is held annually and has the largest single-tournament prize pool of any esports event, with each iteration continually surpassing the previous year's, with the most recent one having one over $34 million. An experienced data analysis coach team is very common and necessary for strong professional esports teams nowadays, since it can give players a mass of detailed advice of how to run the whole game in respect of different opponents and scientifically help design proper tactics, thus improving the probability of winning to a great extent. It has already formed a trend of data analysis in esports. Using data to predict the outcome of the game is also prevalent among lottery companies and websites.

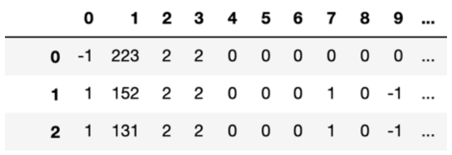
DOTA 2 is a player versus player real-time strategy game. 5 players pick a hero from the hero pool respectively according to different game mod, including "All Pick", "Random Draft", "Single Draft", "Captains Mode", to form a team against the other 5 players. Every hero has its own primary attributes, which is divided into strength, agility and intelligence. Each hero can act as several roles, such as: *Carry*: The core of a team who is assumed to obtain the most economy benefits and deal the most damage to enemy heroes. *Initiator*: The role who starts a team fight which is crucial to win it. *Disabler*: It has a guaranteed disable for one or more of their spells to restrain the action capability of enemy heroes. *Durable*: This role has the ability to last longer in team fights and it tries to bear as much incoming damage as for its' teammates. *Support*: A role needs less gold and benefits teammates with healing or power up spells, and so on.

Since every hero has its own weakness and strength, an all-round draft with high winning probability should consist of as many types of roles as possible. Each player controls his or her own hero with unique abilities and spells to gain gold by killing enemy minions and enemy heroes as well as destroying turrets, which can be considered as accumulating advantage in a game. Players can increase the power of their hero by purchasing items with the gold they earned. The ultimate goal is to destroy enemy's base, which is the "ancient".

Our group decide to use data preprocessing and methods including KNN, Logistic Regression, Random Forest, Multilayer Perceptron based on PCA, in order to offer hopefully accurate prediction of a DOTA 2 game following some related works discussed bellow. Besides, we use original feature to produce Naive Bayes of both Bernoulli and Gaussian. With the assistance of all these techniques, the goal is to predict the outcome of a single game based on the draft of heroes, game mode and game type.

2. Dataset

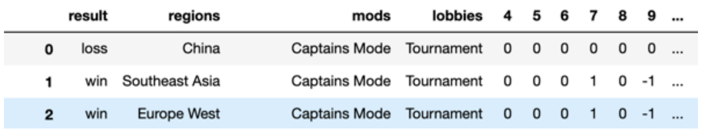
Our dataset without null values, provided by UCI Machine Learning Repository, contains 117 categorical value columns of 92,650 DotA 2 games, in which one column represents the result of the game, either win or lose, while other 116 columns describe the properties of the game. Following is a snap of the dataset.



	0	1	2	3	4	5	6	7	8	9	...
0	-1	223	2	2	0	0	0	0	0	0	...
1	1	152	2	2	0	0	0	1	0	-1	...
2	1	131	2	2	0	0	0	1	0	-1	...

Figure 1: Dataset

As shown above, first four columns of the dataset is given in integer codes that represent different column values, and we are given a couple json files to decode them. Specifically, the first column represent the result. The rest 113 columns are the one-hotted result of the heroes chosen for a game: 1 indicates the hero chosen for a team, while -1 indicates the hero chosen for the opposing team. After using the json files, the first four columns of the dataset can be decoded to the following Figure 2.



	result	regions	mods	lobbies	4	5	6	7	8	9	...
0	loss	China	Captains Mode	Tournament	0	0	0	0	0	0	...
1	win	Southeast Asia	Captains Mode	Tournament	0	0	0	1	0	-1	...
2	win	Europe West	Captains Mode	Tournament	0	0	0	1	0	-1	...

Figure 2: Decoded Dataset

3. Exploratory Analysis

Then, an exploratory analysis is performed on this dataset. The dataset contains 48,782 wins and 43,868 losses. Although slightly different, the dataset is generally balanced with respect to results. In the following Figure 3, we would like to compare the number of games in each region, game mode, and game lobby in the left column, along with the number of games in the same categories while separated according to game results in the right column.

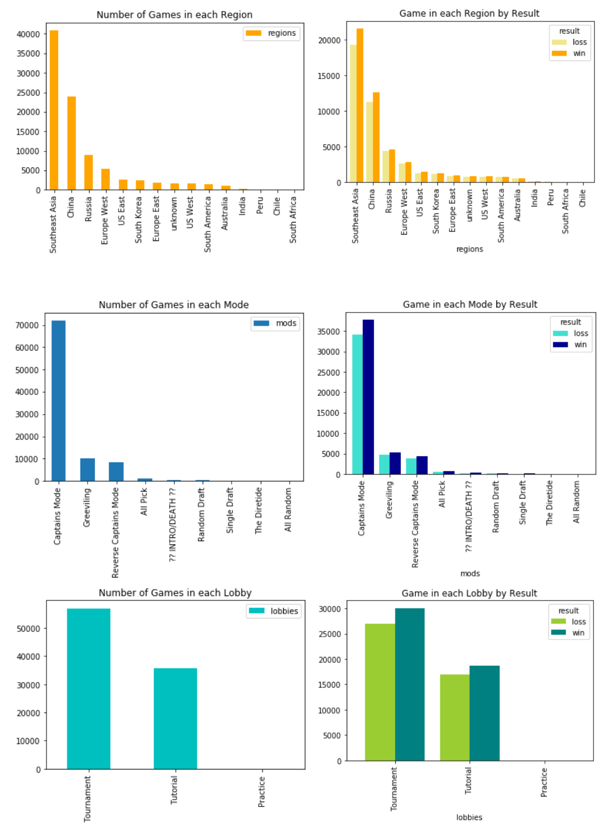


Figure 3. Number of Games in each Region/Mode/Lobby and by Result

From left column, we see that South East Asia hosts the most games, nearly half of the games, followed by China who hosts more than 1/4 of the games. Then, around 15,000 of the games are hosted in Russia and Western Europe. We can also observe that majority of the games in the dataset are played in Captain Mode, while around 10,000 are played in each of Greeviling and Reverse Captain Mode. Then in the third row, we see that more games are tournaments than tutorial, while there are no practice games. From the right column, in which distributions of each categories separated by the game results are shown, we observe that the distributions are similar in patterns compared to the left column, and winning games are all slightly higher than lost games generally, following the overall trend of the dataset. This similar trends is intuitive as location and lobby should not have much influence on results. The small influence of game mode, on the other hand, might by due to hero imbalance.

Then, we take a deeper look into information of heroes and compositions. From Figure 4 below, we observe that most frequently chosen heroes are generally the same among all the compositions and among the winning compositions. The top five chose heroes are: Mirana, Phanton Assasin, Pudge, Legion Commander, and Juggermaut.

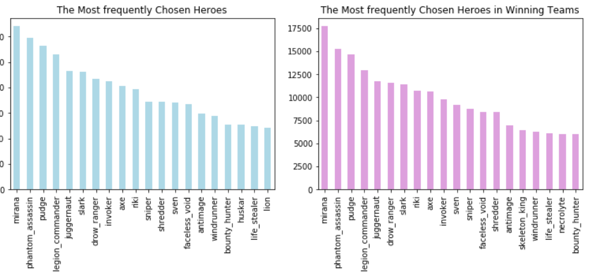


Figure 4. Most Frequent Heroes among all Teams and among Winning Teams

Moreover, winning rate is recorded for each hero. From Figure 5 below, we observe that heroes with highest winning rate are Ominiknight with 0.61 win rate, Elder Titan with 0.59 win rate, and Necrolyte with 0.56 win rate. Heroes with highest winning rates are fairly different from most frequently chosen heroes. This might be because that mostly chosen heroes possess necessary functions for a team composition so that every team has to pick them, while winning heroes are keys to victories. Another possible reason is that these heroes with highest winning rates are chosen for only a small portion of games, and thus produce biased result.

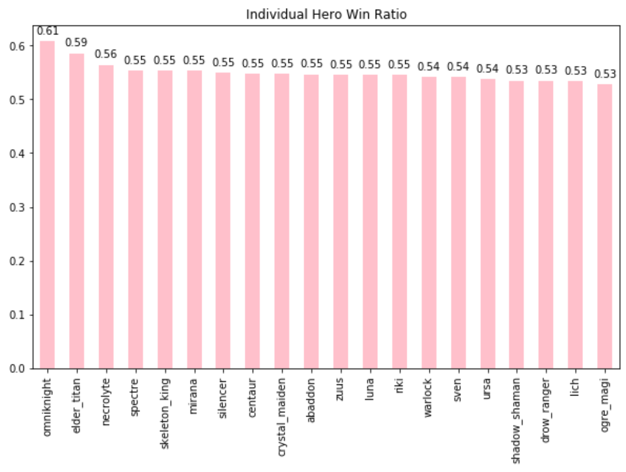


Figure 5. Win Rate for the Top 20 Heroes

Besides individual heroes, we also investigate hero pairs. From figure 6 below, we observe that the top chosen hero pairs are also very similar between all compositions and those among winning compositions. The top five hero pairs are: (Pudge, Phanton Assasin), (Mirana, Pudge), (Mirana, Legion Commander), (Mirana, Phanton Asssasin), and (Jauggernaut, Mirana).

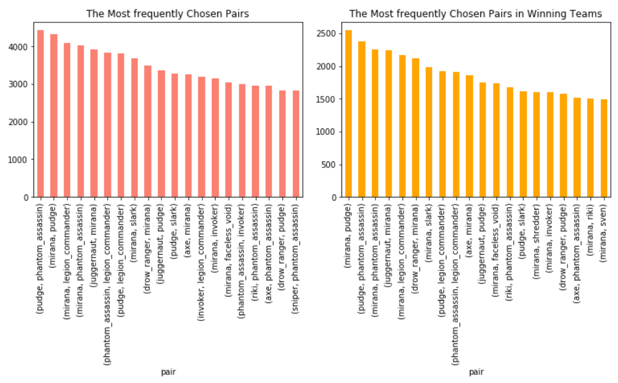


Figure 6. Most Frequent Hero Pairs

Again, winning rates are recorded for each pair and shown in Figure 7 below. The top five pairs with highest win rates are: (Venomancer, Omniknight), (Chen, Brewmaster), (Omnikight, Treant), (Death Prophet, Visage), and (Chen, Troll Warlord). Interestingly, Omniknight shows up in two of these pairs, which is consistent with the individual hero win rate ranking above. We also confirm that frequently chosen pairs are different from pairs with highest win rates, due to either or both of the reasons mentioned before.

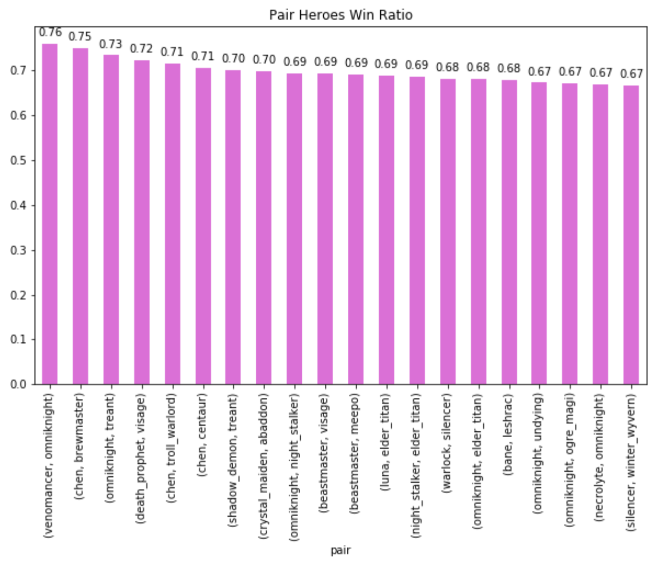


Figure 7. Win Rates for the Top 20 Pairs

After individual heroes and hero pairs, we investigate the whole five-hero compositions. The compositions that show up the most are: (vengefulspirit, beastmaster, luna, lycan, abaddon) and (dazzle, batrider, ancient_apparition, invoker, techies), both ten times. We realize that compositions must have been very different among all the teams so that the most chosen compositions only show up ten times. On the other hand, the compositions that show up the most among winning teams are: (vengefulspirit, beastmaster, luna, lycan, abaddon) and (axe, mirana, sven, lion, invoker), which show up ten times and four times respectively in winning teams. Note that the composition (vengefulspirit, beastmaster, luna, lycan, abaddon) wins all the ten games it shows up. We choose not to analyze the win rates for complete compositions as many compositions have 100% win rate due to small number of games they show up.

4. Related Work

There are several successful pieces of paper related to machine learning applied in DOTA 2 game outcome prediction in the past. The paper by Petra Grutzik, JoeHiggins and Long Tran has the similar goal of predicting the outcome of a DOTA 2 professional matches. They create 3 features sets to test which data would provide the most prophetic structure, and then put two models into practice, SVM and Neural network and compare the test accuracy of these two. It turns out to be that the results produced by machine learning is very close to human performance, and further improvement and augmentation of algorithm and method can actually cross the threshold of human prediction.

Kaushik Kalyanaraman, on the other hand, uses detailed augmented algorithm, which is genetically based on co-occurrence graphs, to improve the prediction results of the existing model and detail the entire process. It considers DOTA2 match as a hero recommender in a recommendation engine. The test accuracy asymptotically approaches 69.42% and does not change much with an increase in the training set size, showing the importance of hero composition in a team.

5. Methods

In this paper, we will predict if the team will win the game when they choose a specific hero team composition competing with other teams' hero composition. The predictive task will be interesting and practical since it can be applied in the DOTA 2 gaming ban picks not only in personal use but also in professional games.

5.1 Metric

We will use accuracy as our metric to evaluate our model in this project.

5.2 Feature Selection

As mentioned in exploratory analysis, we found the features other than heroes less interesting since the variance of winning result of each feature is more relatively consistent than the feature of heroes. To confirm this assumption, we built a naïve logistic regression feeding in features except heroes, and the accuracy $\approx$ 0.5, which nearly equals to random guessing in binary classification. Thus, our model will only consider the features of 113 heroes.

5.2 Data Preprocessing

The heroes feature in the original data is a 113-dimension one-hot encoded feature, with following condition:

$$\begin{cases} 0\ the\ hero\ is\ not\ chosen \\ 1\ the\ hero\ is\ chosen\ by\ this\ team \\ -1\ the\ hero\ is\ chosen\ by\ the\ oppnents'team \end{cases}$$

Although the feature is preprocessed with one-hot encoding, we want to distinguish the difference between chosen by the target team and chosen by the opponents' team, so we split the feature into two sperate 113-dimension one-hot encoded features, noted as X_1 and X_2 respectively. X_1 represents the heroes chosen by this team, and X_2 represents the heroes chosen by the opponents' team with -1 changed to 1.

In other words, the original hero selection feature X is broken down into two matrices X_1 and X_2, in means of considering the general advantage of Radiant over Dire, is encoded as follows:

$X = [X_1, X_2]$, where

$$X_{1,ji} = \begin{cases} 1\ if\ hero\ i\ is\ on\ Team\ 1\ during\ match\ j \\ 0\ otherwise \end{cases}$$

$$X_{2,ji} = \begin{cases} 1\ if\ hero\ i\ is\ on\ Team\ 2\ during\ match\ j \\ 0\ otherwise \end{cases}$$

By default, we consider Team 1 as our target team. For evaluation purpose, we randomly shuffled $X = [X_1, X_2]$ and split the dataset into training and testing sets. The first 80% is used for training and the rest 20% is used for testing.

5.3 K-nearest neighbors Classification with PCA

Since we have 226 features in our original data set, we decide to use PCA to reduce the difficulty of computation. We use the Python scikit-learn package, and set K = 2 because it's a binary classification to predict if the team win the game. For tuning parameters, we tried two weight function: the first one is the uniform weights that all points are equally weighted and the second one is that closer points have larger weight, and we also choose {'auto', 'ball_tree', 'kd_tree', 'brute'} as parameters for tuning. Afterwards, the default parameter of KNN classifier works the best. KNN eventually results in 0.756 training accuracy and 0.485 test accuracy with distance weights. It is observed that test accuracy is significantly lower than training accuracy, which indicates that KNN is overfit even with PCA applied. KNN algorithm requires no assumption of our data, but is especially computionally heavier and can also overfit quickly.

5.4 Logistic Regression with PCA

We then used Logistic Regression with PCA. In Logistic Regression, the L2 norm, square root of the sum of the square of vectors is used. We tried to minimize the cost function to find the result of the events. The formula of the loss function is given below:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log\left(\exp\left(-y_i\left(X_i^T w + c\right)\right) + 1\right)$$

Since it is possibility-based algorithms, it is useful in binary classification. For parameters tuning, we perform grid search for the following parameters: {penalty: = {'l1', 'l2'}, C = {.001, .01, .1, 1, 10, 100}}. The best parameter was l2 and 1 for validation set. It results in 0.557 training accuracy and 0.511 test accuracy. We noticed that Logistic Regression gives a slightly higher accuracy than KNN does.

Logistic Regression Classification is beneficial as it gives us probability information and also requires no assumption. However, as our features are not roughly linear, this model is not the best option.

5.5 Random Forest with PCA

For Random Forest, we again utilize the Python scikit - learn package. We set maximum depth as 2 since we are performing a binary classification. To tune the algorithms, four values of the number of trees are tested, 5, 10, 50, 100, and 10 gave us the best validation accuracy. The training accuracy gained from Random Forest is 0.528 and test accuracy is 0.527. We conclude that random forest achieves a higher accuracy than KNN and Logistic Regression in our case.

By using Random Forest, weights of the features are taken into account and that overfitting seldom happens as parameters number of trees and number of features on each tree are both tuned to maximize the validation accuracy. However, it also produces low accuracies here.

5.6 Multilayer Perceptron with PCA

We then implemented a Multilayer Perceptron classifier via Keras with PCA transformed dataset of first 40th features. We use 'Adam' algorithm as our optimizer, and we use binary cross entropy as our loss function since it is a binary classification. The layer structure of the model showed in below:

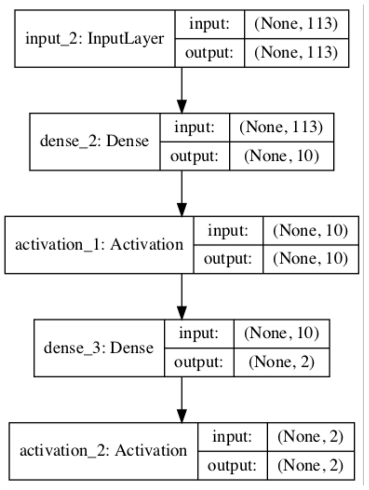


Figure 8. Model Layer Structure

The input layer is the original X of hero layer, and we use Dense layer with shape 10 as our hidden layer, which means it has 10 latent factors that may influence the prediction result. Then we use RELU to activate the layer. Finally, it has a shape 2 output layer with Sigmoid activation in the end. We train our model with 2 epochs and 8 batch size. The training curve of the model shows below:

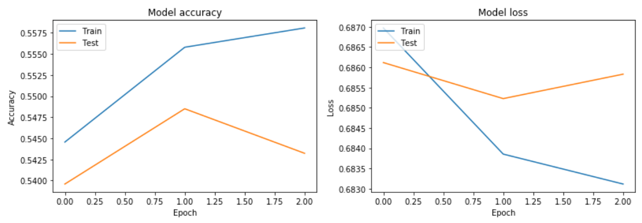


Figure 9. Training Curve

From the plot, we observe that, the model will be overfit after epoch 1. We tried to optimize the result by changing batch size and adding dropout, but it would be still overfit after epoch 1. Then we make assumption that the dataset is not very sufficient for deep learning.

The testing accuracy of the model is significantly better than a random guessing model with 54.7% with 1 epoch when first 40th PCA transformed hero feature feed in.

5.7 Multilayer Perceptron with original feature

We also implemented a Multilayer Perceptron classifier via Keras with original dataset (X in Data Preprocessing section). We use ‘Adam’ as our optimizer, and we use binary cross entropy as our loss function since it is a binary classification. The layer structure of the model showed in below:

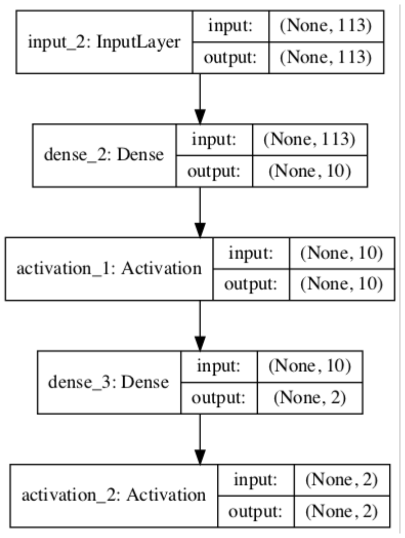


Figure 10. Model Layer Structure

The input layer is the original X of hero layer, and we use Dense layer with shape 10, again, as our hidden layer. Then we use RELU to activate the layer. Finally, it has a shape 2 output layer with Sigmoid activation in the end. We train our model with 2 epochs and 8 batch size. The training curve of model shows below:

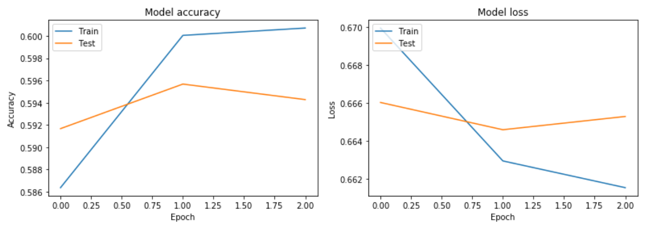


Figure 11. Training Curve

From the plot, we observe that, the model will be overfit after epoch 1. We tried to optimize the result by changing batch size and adding dropout, but it would be still overfit after epoch 1. Then we make assumption that the dataset is not very sufficient for deep learning.

The testing accuracy of the model is significantly better than the baseline model with 59.7% with 1 epoch. Multi-perceptron approaches are computationally faster than utilizing scikit-learn packages for extremely large dataset and are probably the best models we can think of. Its limitation rests upon its variety of parameters, which results in long tuning process when training models.

5.8 Gaussian Naive Bayes with original feature

The Gaussian Naive Bayes model assumes the likelihood of the features to be Gaussian, which is given below:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)}$$

The parameters σ_y and $\mu_y$are estimated using maximum likelihood.

GNB assumes that features should be independent to each other, and that they follows a normal distribution. We also have to assume that the features have continuous values, but we one-hot encoded our feature (which made it non-continuous). This could be the reason for Gaussian Naive Bayes Classifier worse performance compared to Bernoulli Naive Bayes Classifier as we will show later.

The training accuracy for this model is 58.8%, and testing accuracy for this model is 58.5%. We thus consider Gaussian Naïve Bayes to be better than KNN, Random Forest and Logistic Regression.

5.9 Bernoulli Naive Bayes with Original Feature

An even better model for this dataset and this specific predictive task is Bernoulli Naive Bayes. We assume that the data is distributed according to multivariate Bernoulli distributions; in other words, we consider each feature to be binary-valued, and classify the dataset accordingly.

Similar to GNB, BNB also assumes that features should be independent to each other, and that they follows a normal distribution. However, BNB assumes that features are discrete, or more specifically, binary. Such assumption made BNB to have the benefit of explicitly modelling the absence of terms in the feature array.

The training accuracy for this model is 60.2%, and testing accuracy for this model is 59.9%. This accuracy is the best among all models that we have tried.

Advantages of both Naïve Bayes Classifier includes: it doesn’t require much data to make classifications (uses maximum likelihood estimation), strong performance in practice, help alleviate “the curse of dimensionality” (the need for data points grow exponentially when number of features grow, but Naive Bayes assumes that the features are independent thus can be estimated as a onedimensional distribution). On the other hand, it is computationally harder especially when we have many categorical features, which is its limitation.

6. Result

PCA	Model	Train Accuracy	Test Accuracy
With PCA	KNN	0.756	0.485
	Logistic Regression	0.557	0.511
	Random Forest	0.528	0.527
	Multilayer Perceptron	X	0.547
With Original Feature	Multilayer Perceptron	X	0.597
	Gaussian Naïve Bayes	0.588	0.585
	Bernoulli Naïve Bayes	0.602	0.599

Figure 12. Prediction Result Table

Observing the table above, we found attempts that involve PCA process all produce very low test accuracies, often not much different from a random guessing model. For KNN classifier, although it has a fair train accuracy of 75.6%, it also produces the lowest test accuracy of 48.5%, suggesting that the model has overfitted the training data and that KNN is not the best model to predict DOTA 2 games when we only have hero data. Other attempts with PCA, although without clear sign of overfitting, also produce low accuracies. Note that with Multilayer Perceptron and PCA, the test accuracy is slightly higher than other attempts with PCA, indicating that latent factors, most likely the synergistic effect among hero selections, exist. On the other hand, attempts using original features without PCA process all produce accuracies near 60%. Although 60% is still not an optimistic result given a binary classification task, it is significantly higher than what we obtained with the attempts that involve PCA.

We thus concluded that when predicting a DOTA 2 game, it is best when we have the complete hero composition data from both competing teams, as our results demonstrate a poor reconstruction with PCA. This conclusion is the most obvious when comparing results of 5.6 and 5.7, as both approaches utilize the same model but different feature representations and 5.7 produces a higher accuracy. Using complete hero composition information is the key to better predictions with approaches 5.7 to 5.9.

We also concluded that we have reached the potential limit of the given data in prediction task. This is shown by the similarly near-to-sixty accuracies we obtained by approaches 5.7 to 5.9. This result, along with the fact that we have considered latent factors in some approaches, indicates that such accuracy might be the ceiling considering the data we have. In other words, to achieve higher prediction accuracy, we might have to include more data, such as player information.

7. Discussion and Future Work

From the result and conclusion above, we learn that complete hero composition information is much informative compared to a reconstruction using PCA. The result suggests that synergy and interactions among and against heroes are important factors to the outcome of a DOTA 2 game. This result is also intuitive as DOTA 2 is a team-based game in which hero combinations matter.

On the other hand, we also learn that 60% is probably the ceiling of prediction accuracy we can achieve with merely hero composition information, suggesting that we need more data input to obtain higher accuracies. It is also a reasonable result as a competitive game should not only depend on hero compositions, but also players themselves. Moreover, eSport brings excitement only when the results are uncertain. A high accuracy of our models would only suggest hero imbalance, as we can accurately predict results merely by hero compositions without player information.

If we have more time, we would aim to obtain player information either by web API or scraping (legally). In this way, besides hero composition, we also take into account player skills and styles. As the report *To win or not to win?* has shown, adding player information could likely boost the accuracy. The suggestion is also reasonable because different play styles, along with the synergy and interactions among them, can largely affect a competitive team-based game.

As a result, two extensions are promising: obtaining player data input such as career profile (how long has this player played each hero) or individual player win rate, and obtaining team data such as the past win rates of teams with the same team members. While the former can provide features that represent individual player influence, the latter can take into account the synergistic effect among players in the models.

8. References

Grutzik, Petra. Higgins, Joe. Tran, Long. “Predicting outcomes of professional DotA 2 matches (2017).

Kalyanaraman (2014). To win or not to win? A prediction model to determine the outcome of a DotA2 match.http://cseweb.ucsd.edu/jmcauley/cse255/projects/KaushikKalyanaraman.pdf

9. Python Packages

9.1 Keras

https://keras.io/getting-started/sequential-model-guide/

https://keras.io/

9.2 Scikit-Learn

KNN:
https://scikit-learn.org/stable/modules/generated/
sklearn.neighbors.KNeighborsClassifier.html

Logistic Regression:
https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression

Random Forest:
https://scikit-learn.org/stable/modules/generated/
sklearn.ensemble.RandomForestClassifier.html

Naïve Bayes:
https://scikit-learn.org/stable/modules/
naive_bayes.html