World of Warcraft Battleground Analysis

Daxin Niu

Brown Data Science Initiative

December 2021

Github: https://github.com/daxinniu/World_Of_Warcraft_BG_Analysis

1. Introduction

1.1 World of Warcraft

World of Warcraft is one of the most popular MMORPG games that was created by Blizzard Entertainment. The game contains two major game modes which are Player Versus Player mode and Player Versus Environment mode. One of the most popular game modes in the PvP section is Battleground. This game mode assembles 10 to 40 players (Depending on the map) from each faction (Alliance/Horde) and has the Horde compete against the Alliance.

1.2 Why Interesting

As one of the most popular MMORPG that has lasted over 16 years, World of Warcraft has a quite large player base. Analyzing a World of Warcraft Battleground dataset could allow players to gain insights on whether their choice of battleground, class, faction, and role could help them achieve victory in a game.

1.3 Objective

Our main objective for this project is to try to predict whether a player will win or lose when they play a battleground game. We will be given a player with detailed information and predict whether the player will win the game. Eventually, we want to predict with high accuracy.

1.4 Data Information

We obtained our dataset from Kaggle. We are only using the wowbgs.csv file for battleground analysis. This file contains different characters' statistics after each of their games. Each line of the dataset represents an individual player and their own statistics for a single game. The dataset originally contains a “Win” and a “Lose” column. If the player won the game, they will have 1 in the “Win” column. Otherwise, they will have 1 in the “Lose” column. To simplify the data for our analysis, I have combined the columns into one “Win” column where winning will give us 1 in the row and 0 otherwise. This “Win” column is our target variable. The “Lose” column will be dropped. We have also dropped the “Code” column where it provided multiple code names for each battleground. The description for this column is vague and the original author excluded this column from his analysis as well. The “Honor” column will be dropped as well since it will expose some information about the target variable.

Given that our target variable only contains 1 or 0, we are facing a classification problem. After the initial cleaning, our dataset has 3726 rows and 11 columns.

1.5 Related Projects

There are several projects that are related to this dataset. One user on Kaggle used multiple CSV files to predict which faction would win the game. He applied XGBoost and

achieved an 85.5% accuracy. The author also performed feature analysis and found out that the Kills column plays an important role as a feature.

There is another author who performed basic EDA on the dataset. He found out that Alliance earn more honor than Horde during bonus events weekends which is quite interesting.

2. Exploratory Data Analysis

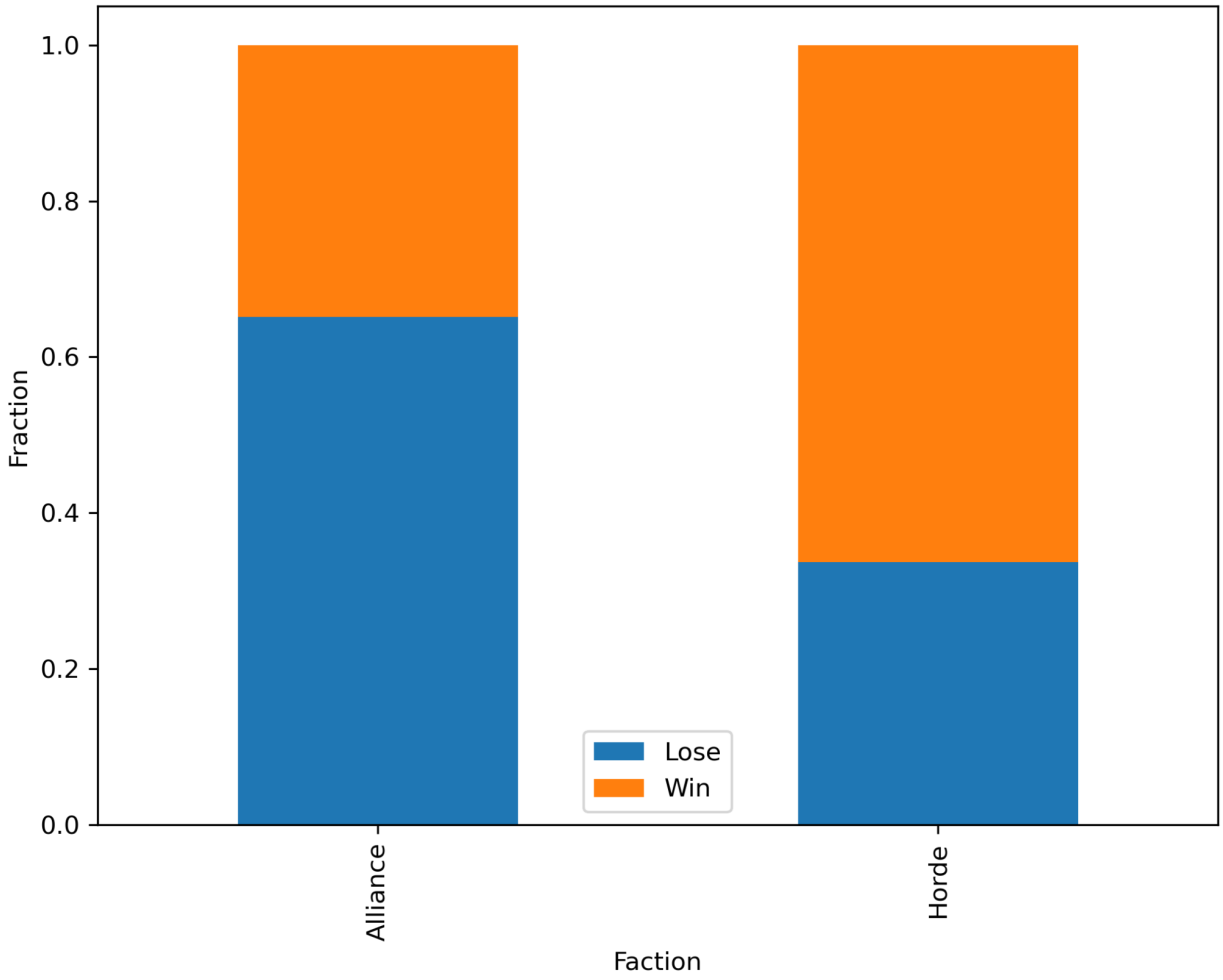

Figure 1: Win fraction by Faction. Horde tends to win more games.

From Figure 1, we can see that among all the data we have, the Horde faction tends to win more battleground overall. This could an important feature that could help us to predict whether a play would win or not based on their faction.

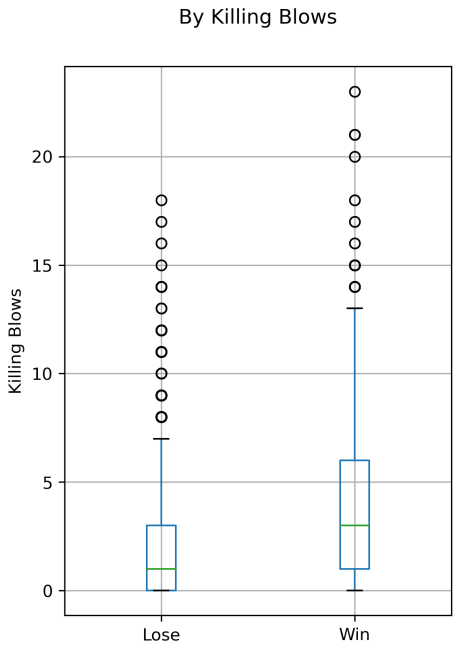

Figure 2: Boxplot on Killing blows for Losing and Winning. Winning side have a significant higher median.

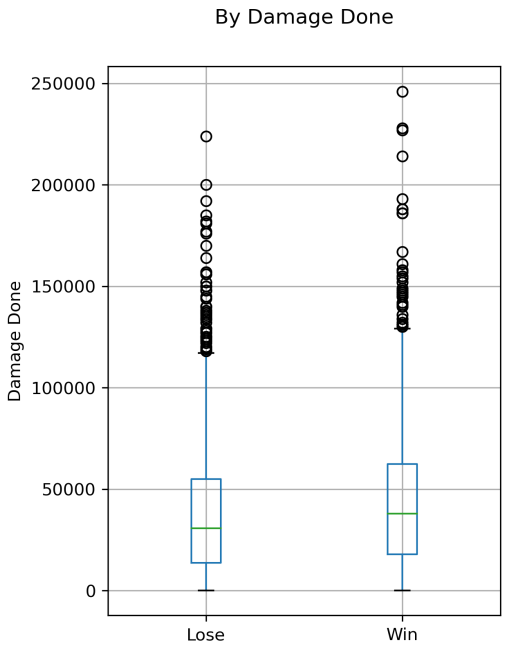

Figure 3: Boxplot on Damage Done for Losing and Winning. The spread for both side are quite close.

Figure 2 and Figure 3 are boxplots illustrating Killing Blows and Damage Done for winning and losing. Contrary to common belief, winning players do not have much more damage done than losing players. But they do tend to have a higher spread in killing blows.

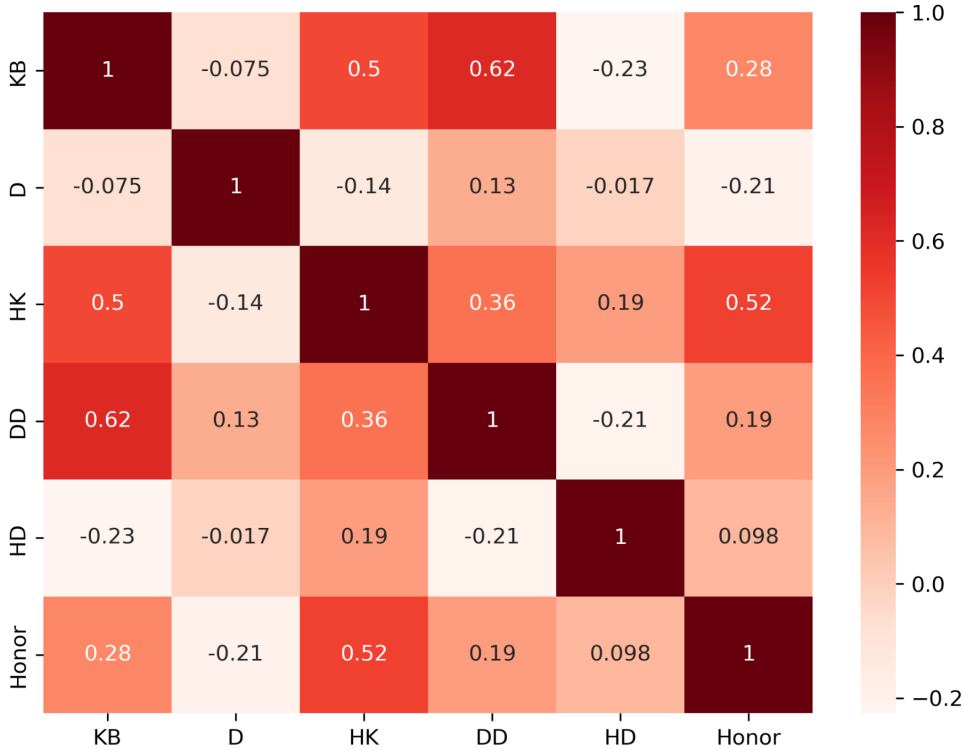

Figure 4: The above image illustates the correlation between the numerical features.

Figure 4 above shows the correlation between our numerical features. We can see that Killing Blows and Damage Done have a quite high correlation. This makes sense because players that obtain more kills tend to do more damage overall. Killing Blows also have a negative correlation with Healing Done since healers usually don't focus on killing.

3. Methods

Our dataset is independent and identically distributed given that each line of our data point is a random variable that has the same probability distribution as every other point and they are mutually independent. Furthermore, our data points were generated using the same method with the same features. From the author's dataset description and our observation, we know that our dataset is not a time series data and it does not have a group structure.

3.1 Splitting Strategy

Given that our dataset only contains 3726 rows, we have a quite small dataset. Therefore, the dataset will first be split with 20% for testing and 80% for others. After that, we will apply K-Fold Cross-Validation with 4 folds for the training and validation split. Our basic train test split should work fine given that our data is independent and identically distributed. The classes are quite balanced so we did not have to do a stratified split. This split strategy fits our ML question quite well given that we are aiming to predict whether a random player with random faction, class, battleground, etc would win the game. Applying the basic random split would work for our goal.

3.2 Data Preprocessing

Our dataset originally had 11 features. Among these features, "Battleground", "Faction", "Class", "Rol", "BE" are categorical features. Although these features are categorical, they are not ordered. Therefore, OneHotEncoding will be applied to these features. We only have missing values in the 'BE' column so we added SimpleImputer ahead of OneHotEncoding to fill missing values. The reset 6 features are "KB", "D", "HK", "DD", "HD", "HD" which are all continuous. These features represent different statistics such as "damage done", "healing done" from each player in each game and they usually do not have an upper bound. Therefore, we simply applied StandardScaler to them. After preprocessing our data, we will end up with 34 features.

3.3 Machine Learning Pipeline

For simplicity, I have combined the previous steps into a function to help us build a clean pipeline. I have constructed a function that takes X, y, preprocessor, ML_algo, param_grid, n_random_states. The model would try a given number of random states. For each random state, it will do the splitting and cross-validation mentioned above. The given model will be trained and the test score will be calculated for each random state. Model tuning will also be done using

GridSearchCV with the param_grid passed in for each state. Using this function, we will just need to modify our ML_algo, param_grid for different machine learning algorithms.

I have attempted 5 models for my task which are Logistic Regression, Ridge Classifier, Random Forest, SVC, and XGBoost. Given that our dataset is balanced and we want to know the overall performance instead of analyzing the model's ability to identify underrepresented classes, accuracy would be a good evaluation metric.

To prevent uncertainties due to randomness, I have tried 10 random states for each model. The test scores for each random state were recorded and returned for each model.

3.3.1 Logistic Regression

The first model that I tried was Logistic Regression. There aren't many parameters to tune for this model. But one thing that I have chosen is C. This is a parameter for the regularizer. The value must be a positive float and smaller values give stronger regularization. I have tried 0.01, 0.1, 1, 10, 100 for C.

3.3.2 Ridge Classifier

The second model I tried was Ridge Classifier. Although this model sound similar to the basic logistic regression, they are actually different in some way. For our binary classification task, Ridge Classifier converts the target to 1 and -1 and builds a Ridge model. Similarly, I have tuned the parameter related to the regularization term. The parameter here is alpha which needs to be a positive float. Larger values would give stronger regularization. I have used 0.01, 0.1, 1, 10, 100 since this covers a quite large range for the parameter.

3.3.3 Random Forest

The third model I ran was Random Forest. This model tends to have uncertainties due to randomness. To reduce such issues, I have run 10 random states for the model and recorded the test scores. For this model, I tried to tune max_depth and max_features. Max_depth is essentially how deep each tree is. I have tried 1, 3, 10, and 30 given that these numbers cover a quite large range. (From potentially underfit to overfit) Another one is max_features which is the number of features to consider for the best split. This would give us control over how many features each node could have which is important. I have tried the provided "auto", "sqrt" and "log2".

3.3.4 SVC

The fourth model I tested was SVC. SVC tends to take a long time to train when the dataset is large. Luckily, our dataset size is quite small so we can try this model. There are two things we tuned for this model. This first one is C which is also a regularization parameter. This needs to be positive so we tried 0.1, 1, 10. The other one we tuned is gamma. This is essentially the parameter used to define how far we want the line to influence points. We have tried 0.1, 1, 10 as well.

3.3.5 XGBoost

The last model I have tested was the XGBoost classifier. There are two common parameters to tune for this model. One is max_depth which is the max depth for each tree in the model. I have tested 1, 2, 3, 5, 10, 20, 30 which covers a quite large range. The second parameter I tuned was the learning rate. This indicates how far we want the optimization algorithm to go for each step. Trying different values might help us to find the local minimum easier. I have tried 0.01, 0.05, 0.1, 0.5 for this parameter.

4. Results

We have 10 test accuracy scores for each model from the 10 random states. We will take the mean of the score and select the highest one as our best model.

The baseline score was calculated using the percentage of "Win" in our dataset. There are 1889 wins out of 3726 lines of data which gives us 0.507 as the baseline score. Out of our five models, XGBoost performed the best and ended up with an accuracy score of around 0.812 which is almost 60% better than our baseline model. The average test scores of all five models are shown in the following table.

Model	Average Test Scores
LogReg	0.778552
Ridge Classifier	0.778820
Random Forest	0.809383
SVC	0.799196
XGBoost	0.811930

Table 1: Models and their average test scores

From our table, we can see that XGBoost gives us the highest average test accuracy. We have calculated the SHAP value in the notebook for local feature importance and also used XGB's get_score method with three different ways (weight, total_gain, total_cover) to get the global feature importance. The top three features are HK, HD, and Faction_Alliance. These features do make a lot of sense since these are key indicators of how well a player is performing in each game. The least important feature seems to be Rol_dps which is a bit surprising. People often believe DPS plays an important role but it does not seem to be the case in a team environment.

5. Outlook

At the current moment, our model is having decent accuracy. But our modeling approach might not have the best results since our dataset is small. There still exist ways to improve our models. We could try to remove features with high correlation and just keep one to see if we can get better results. Feature selection using mutual information could be something to consider. If we can have additional player gear level information or anything else player specific, we might be able to gain more insights and get better models.

6. Reference

1. Blesa, Carlos. “World of Warcraft Battlegrounds.” Kaggle, 3 Sept. 2018, https://www.kaggle.com/cblesa/world-of-warcraft-battlegrounds.
2. Mandaloreultimate. “World of Warcraft - Battlegrounds Eda & Prediction.” *Kaggle*, Kaggle, 26 Aug. 2019, https://www.kaggle.com/mandaloreultimate/world-of-warcraft-battlegrounds-eda-prediction.
3. Messner, Steven. “The Best Mmos in 2021.” Pcgamer, PC Gamer, 8 Jan. 2021, https://www.pcgamer.com/the-best-mmos/.