





Predicting the final placement in video games: **An Example in PUBG**

MIS 637 Final Project
Author: Tianrui Wang
Instructor: Mahmoud Daneshmand
Date: 3 MAY 2019



INTRODUCTION

PlayerUnknown's BattleGrounds (PUBG) is a battle royale-style video game that enjoyed massive popularity. With over 50 million copies sold, it's the fifth best-selling game of all time, and has millions of active monthly players.

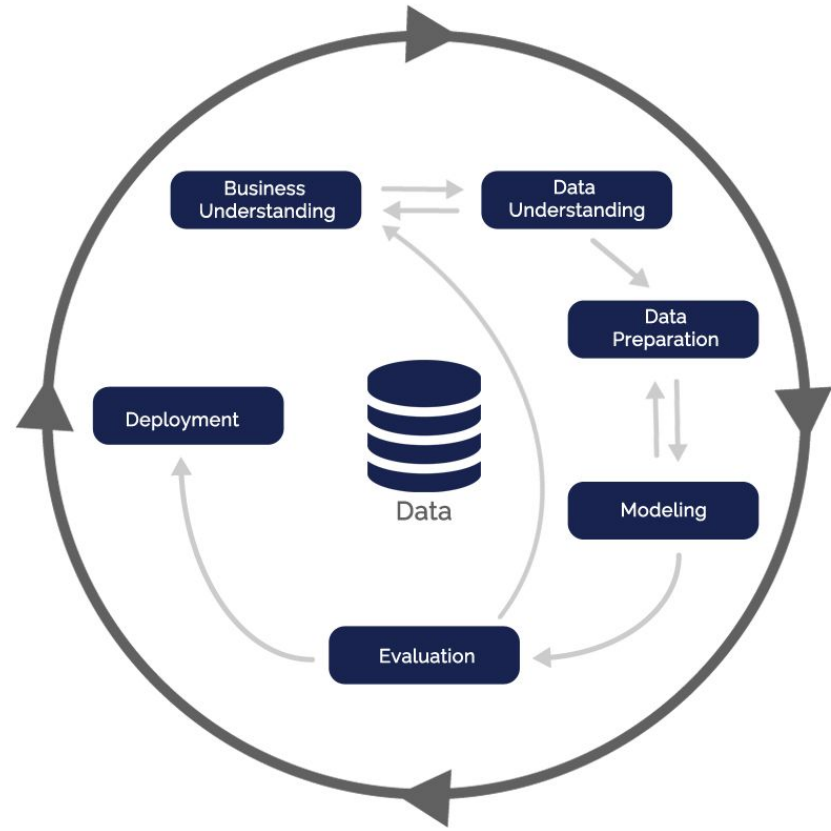
In each game, 100 players are dropped onto an island empty-handed and must explore, scavenge, and eliminate other players until only one is left standing, all while the play zone continues to shrink.

Players can choose to enter the match solo, duo, or with a small team of up to four people. The last person or team alive wins the match.



Cross Industry Standard Process for Data Mining (CRISP-DM)

1. Research Understanding Phase
2. Data Understanding Phase
3. Data Preparation Phase
4. Modeling Phase
5. Evaluation Phase
6. Deployment Phase



Research Understanding

The game's anonymized player data are shared with the public through PUBG Developer API.

The objective of this project is to predict the players' final placement classes through the final in-game stats and initial player ratings, in order to make a best strategy for gaming, detecting possible cheaters, and more.

There are a few underlying questions require resolving through deployment of data mining techniques:

- Are there interdependent relationships among the input variables?
- Are cheaters outliers? How to detect cheaters? Shall we remove statistical outliers or not?
- Given a continuous target variable with a range from 0 to 1, how to transform it into a categorical variable?

Data Understanding

- **Data Source**
 - The dataset is retrieved from Kaggle.com as they collected the data through PUBG.
- **Data Shape**
 - 444697 entries, 29 columns
- **Target Variable**
 - winPlacePerc - a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match.
 - The original numerical variable is transformed to a categorical variable, **win**, where 1 refers to winning the game (winPlacePerc = 1), and 0 refers to losing the game (winPlacePerc < 1).

Data Understanding

- **Label Variables**
 - **Id** - Player's Id
 - **groupId** - Integer ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
 - **matchId** - Integer ID to identify match. There are no matches that are in both the training and testing set.
- **Numerical Variables**
 - **assists** - Number of enemy players this player damaged that were killed by teammates.
 - **boosts** - Number of boost items used.
 - **damageDealt** - Total damage dealt. Note: Self inflicted damage is subtracted.
 - **DBNOs** - Number of enemy players knocked.
 - **headshotKills** - Number of enemy players killed with headshots.
 - **heals** - Number of healing items used.
 - **killPlace** - Ranking in match of number of enemy players killed.
 - **killPoints** - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.)
 - **kills** - Number of enemy players killed.
 - **killStreaks** - Max number of enemy players killed in a short amount of time.
 - **longestKill** - Longest distance between player and player killed at time of death. This may be misleading, as downing a - player and driving away may lead to a large longestKill stat.

Data Understanding

- **Numerical Variables (Con't)**

- **matchDuration** - Duration of match in seconds.
- **maxPlace** - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
- **numGroups** - Number of groups we have data for in the match.
- **rankPoints** - Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API's next version, so use with caution. Value of -1 takes place of "None".
- **revives** - Number of times this player revived teammates.
- **rideDistance** - Total distance traveled in vehicles measured in meters.
- **roadKills** - Number of kills while in a vehicle.
- **swimDistance** - Total distance traveled by swimming measured in meters.
- **teamKills** - Number of times this player killed a teammate.
- **vehicleDestroys** - Number of vehicles destroyed.
- **walkDistance** - Total distance traveled on foot measured in meters.
- **weaponsAcquired** - Number of weapons picked up.
- **winPoints** - Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.)

- **Categorical Variables**

- **matchType** - String identifying the game mode that the data comes from. The standard modes are "solo", "duo", "squad", "solo-fpp", "duo-fpp", and "squad-fpp"; other modes are from events or custom matches.

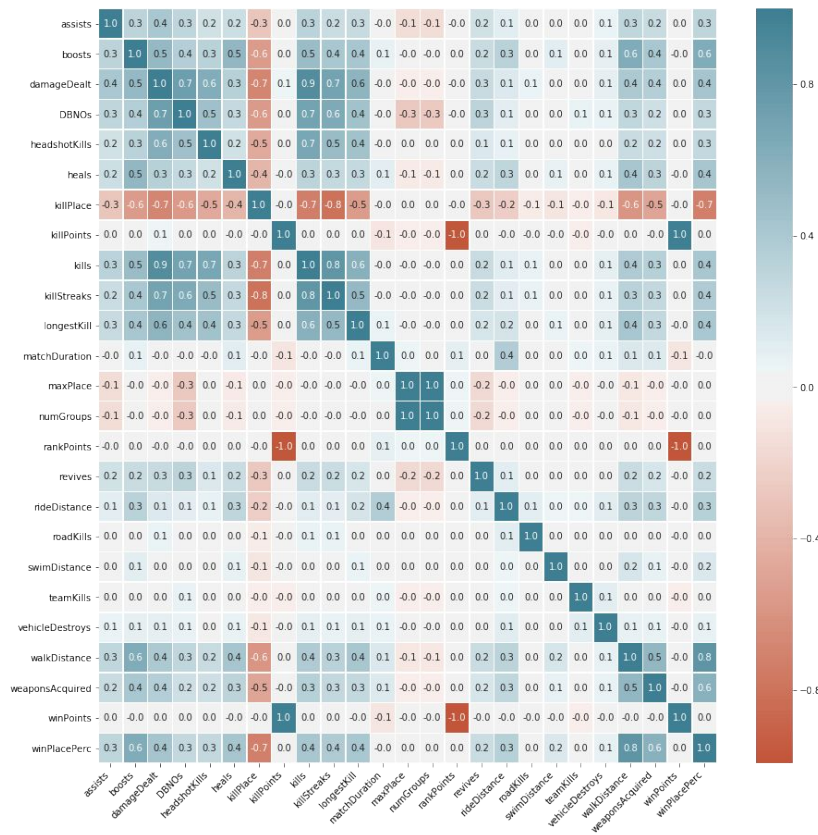
Data Understanding

First Five Rows

Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	kills	killStreaks	longestKill	matchDuration	matchType
6b8e8c092259b412e6cc			0	1	507.6	5	1	0	4	0	4	2	72.69	1868	squad-fpp
0e975f106f5e6d147c13			0	0	322.1	0	0	0	11	0	3	2	54.76	1388	solo-fpp
8e889f47991e53984a7f			0	5	100	0	0	1	23	991	1	1	118.4	1320	solo
fa063cb6a3204b9f9dc8			0	1	175	1	0	7	19	0	2	2	16.81	1450	duo-fpp
d417c0a02fc561ad0a4f			2	3	338.2	0	0	9	22	1375	1	1	23.31	1867	duo

maxPlace	numGroups	rankPoints	revives	rideDistance	roadKills	swimDistance	teamKills	vehicleDestroys	walkDistance	weaponsAcquired	winPoints	winPlacePerc
27	23	1512	1	4514	0	0	0	0	1282	10	0	0.5385
93	91	1494	0	0	0	0	0	0	285.2	2	0	0.3913
99	97	-1	0	0	0	93.26	0	0	2264	5	1498	0.8469
47	46	1500	0	0	0	0	0	0	1363	4	0	0.587
44	43	-1	1	0	0	0	0	0	3945	9	1589	0.7907

Data Understanding - Pearson Correlation



In terms of the target variable (unprocessed data), there are a few variables with high correlations:

- walkDistance (highest positive)
- boosts (positive)
- weaponsAcquired (positive)
- killPlace (highest negative)

We could also observe that correlations among killPoints, rankPoints, winPoints are all 1.0 or -1.0, indicating they are essentially the same.

Data Preparation - Data Cleaning

Missing Value Detection

- There are no missing values in the dataset.

Outliers Detection

- One of the research interest is to detect cheaters in the game. Players performed abnormally in the game could be considered cheaters and thus affect the prediction as outliers.
- The most obvious anomaly is killing without moving (**kills** > 0 & **rideDistance** + **walkDistance** + **swimDistance** = 0)
 - There are 146 instances found in the data set and they are removed as outliers.

Data Preparation - Data Cleaning

Data Transformation

- Transform **winPlacePerc** (continuous numerical variable) into a categorical variable, **win**, where:
 - 1 corresponds to 1st place in the game ($\text{winPlacePerc} = 1$);
 - 0 corresponds to losing the game ($\text{winPlacePerc} < 1$).

Feature Selection

- Remove **Id**, **groupId**, **matchId**, **matchType** for modeling
- Remove **killPoints**, **winPoints** due to high correlation

Data Preparation - Data Division

After data cleaning, the dataset has 444551 entries with 21 input variables, 1 target variable. It is divided into training set and test set:

1. Training set: 75%, used to develop the model
2. Test set: 25% , used to evaluate the model

Modeling - algorithm

Classification and Regression Tree (CART) is selected to model the training data for the following reasons:

1. Decision tree algorithms are **relatively robust to outliers**, and requires little data preparation.
2. Decision tree uses **a white box model**. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret. This project aims to produce explainable rules that could benefit the game players.
3. The purpose of the project is to classify a categorical target variable into two classes. CART constructs **binary trees** using the feature and threshold that maximizes “goodness” at each node. Hence, it would be easier and straightforward to use CART algorithm.

Modeling Software & language

The project uses **python** and **scikit-learn library** to model the training data.
scikit-learn uses an optimised version of the CART algorithm.



scikit-learn is a free software machine learning library for the Python programming language. It has following advantages:

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Modeling - Model 1: General Model

Codes

```
from sklearn import tree
```

```
clf = tree.DecisionTreeClassifier()
```

```
clf = clf.fit(X_train, y_train)
```

```
clf
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,  
                        max_features=None, max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,  
                        splitter='best')
```

Accuracy = 0.9568

```
from sklearn import metrics
```

```
y_pred = clf.predict(X_test)
```

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.9568284475157012

Modeling - Model 1: General Model

Codes

```
from sklearn import tree
```

```
clf = tree.DecisionTreeClassifier()
```

```
clf = clf.fit(X_train, y_train)
```

```
clf
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,  
                        max_features=None, max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, presort=False, random_state=None,  
                        splitter='best')
```

Accuracy = 0.9568

```
from sklearn import metrics
```

```
y_pred = clf.predict(X_test)
```

```
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.9568284475157012

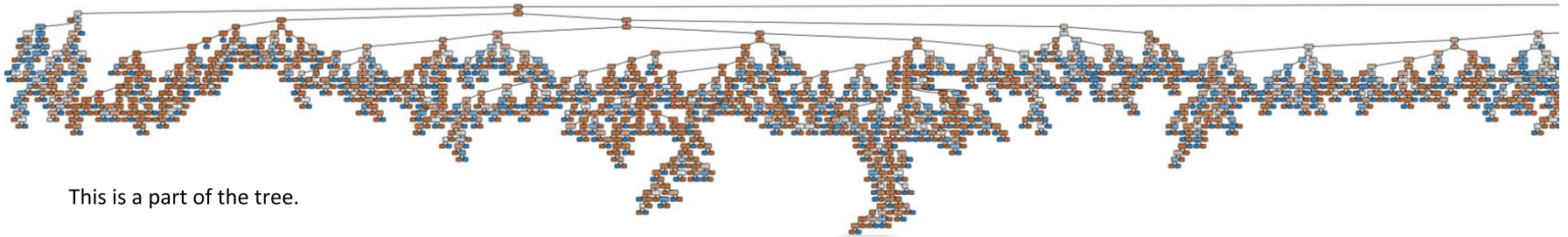
Modeling - Model 1: General Model

Problem:

The tree is too large and hard to interpret.

Solution:

Consider pruning the tree to get it simpler, e.g. changing the `max_depth` of the tree.



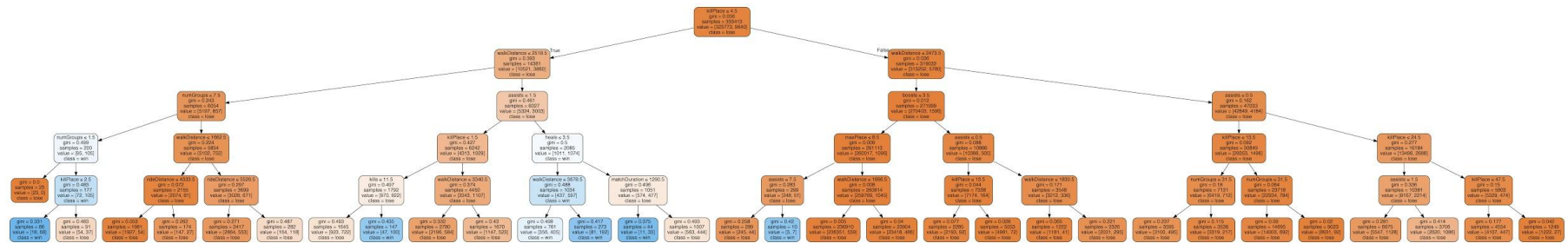
This is a part of the tree.

Modeling - Model 2: fixed max_depth

After testing the parameter of max_depth from 4 to 7, I found that max_depth = 5 yields a good balance between accuracy and complexity of the tree.

```
# max_depth = 5
clf1 = tree.DecisionTreeClassifier(max_depth = 5)
clf1 = clf1.fit(X_train, y_train)
print("Accuracy:", metrics.accuracy_score(y_test, clf1.predict(X_test)))
```

Accuracy: 0.9720527632312981



Evaluation - Cross Validation

5-Fold Cross Validation is used to test how well the model is trained upon the given training set and test it on unseen data.

The result of the cross-validation shows that the model accuracy did not change due to random sampling.

```
from sklearn.model_selection import cross_val_score
clf0 = tree.DecisionTreeClassifier(max_depth = 5)
scores = cross_val_score(clf0, X, y, cv=5)
scores
```

```
array([0.97113968, 0.97134213, 0.97178045, 0.97184794, 0.97188136])
```

Evaluation - Confusion Matrix

The confusion matrix and the classification report shows a high accuracy in predicting the “0” case, which is the losing case; however, the model shows a relatively low accuracy in predicting the “1” case, which is the winning case.

The reason is that we have an unbalanced dataset where winning cases are 2.33% of total cases. However, as the research aims to derive feature importance and provide references to the game players, the importance of the features could be a more important evaluation criteria.

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, clfl.predict(X_test)))
print(classification_report(y_test, clfl.predict(X_test)))
```

```
[[107752   222]
 [ 2884   280]]
      precision    recall  f1-score   support

      0       0.97       1.00       0.99       107974
      1       0.56       0.09       0.15         3164

   micro avg       0.97       0.97       0.97       111138
   macro avg       0.77       0.54       0.57       111138
  weighted avg       0.96       0.97       0.96       111138
```

win	
0	431747
1	12804

Evaluation - Feature Importance

The left chart shows a feature importance ranking for Model 1 (General Model);

The right chart shows a feature importance ranking for Model 2 (Max-depth = 5).

Since Model 2 is limited by the depth of the tree, it only included 10 variables out of 21.

The top two important features are **killPlace** and **walkDistance** for both models.

Interestingly, there are fundamental differences between the two models from the 3rd important feature.

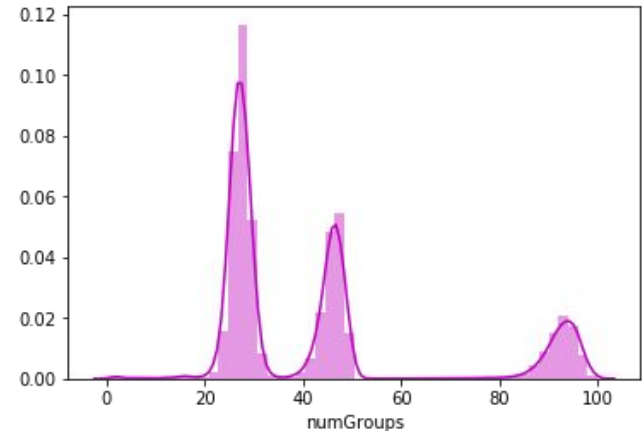
	feature	importance
6	killPlace	0.152023
20	walkDistance	0.145509
10	matchDuration	0.091467
2	damageDealt	0.080646
9	longestKill	0.069371
13	rankPoints	0.057984
15	rideDistance	0.052673
12	numGroups	0.045772
0	assists	0.043031
5	heals	0.042851
21	weaponsAcquired	0.038948
1	boosts	0.037012
11	maxPlace	0.036208
3	DBNOs	0.023949
7	kills	0.018195
17	swimDistance	0.018156
4	headshotKills	0.017427
14	revives	0.015195
8	killStreaks	0.008320
19	vehicleDestroys	0.002420
18	teamKills	0.001609
16	roadKills	0.001236

	feature	importance
6	killPlace	0.533866
20	walkDistance	0.263743
0	assists	0.138751
12	numGroups	0.029066
15	rideDistance	0.010341
1	boosts	0.009521
11	maxPlace	0.004334
5	heals	0.004159
7	kills	0.004112
10	matchDuration	0.002107
8	killStreaks	0.000000
9	longestKill	0.000000
4	headshotKills	0.000000
13	rankPoints	0.000000
14	revives	0.000000
3	DBNOs	0.000000
16	roadKills	0.000000
17	swimDistance	0.000000
18	teamKills	0.000000
19	vehicleDestroys	0.000000
2	damageDealt	0.000000
21	weaponsAcquired	0.000000

Deployment - Feature Importance

The feature importance derived from the tree model could be used as a reference to the PUBG players in order to improve their gaming performance:

- **killPlace**, **assists**, **damageDealt** are 3 major index reflecting players' effective damage in the game, and thus ranked top in feature importance.
- **boosts** and **heals** are two index indicating positive effects of effective protections to win the match.
- **walkDistance** and **rideDistance** both ranked top, indicating active participation could help players win the match.
- Interestingly, **numGroups**, ranked 4th in one of the feature importance.
 - Since there are 3 normal mode to play the game (solo, duo, squad), numGroups = 25 implies squad model; numGroups = 50 implies duo; numGroups = 100 implies squad.
 - This might imply different rates of winning in different modes; or playing in a match with 100 ppl vs less than 100 ppl results in different winning chances.



Deployment - Future Research Direction

Future research could be done in the area of cheating detection in the Video Games.

Due to limited access to cheating data from the video game, at current stage, we could only make assumptions about the cheaters and remove abnormal data as outliers from the training set.

If provided with labeled data on cheaters and non-cheaters in the game, machine learning and classification methods could be used to identify cheaters efficiently. There are already implementations

References

Kaggle.com (2019). PUBG Finish Placement Prediction. Retrieved from <https://www.kaggle.com/c/pubg-finish-placement-prediction/>

PUBG.com (2019). PUBG Developer API. Retrieved from <https://developer.pubg.com/>

Saed, S. (2018). PUBG developer identifies a new cheating pattern, looking into new cheat detecting measure. Retrieved from <https://www.vg247.com/2018/01/19/pubg-developer-identifies-a-new-cheating-pattern-looking-into-new-cheat-detecting-measure/>

Scikit-learn.org (2019). Decision Trees. Retrieved from <https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart>