DATA1030 Final Project

Rank Prediction Using CS:GO Damage Data

Haibo Li
Data Science Initiative
Brown University
October 2022
Github: https://github.com/Haibo3939/DATA1030-Project

Introduction

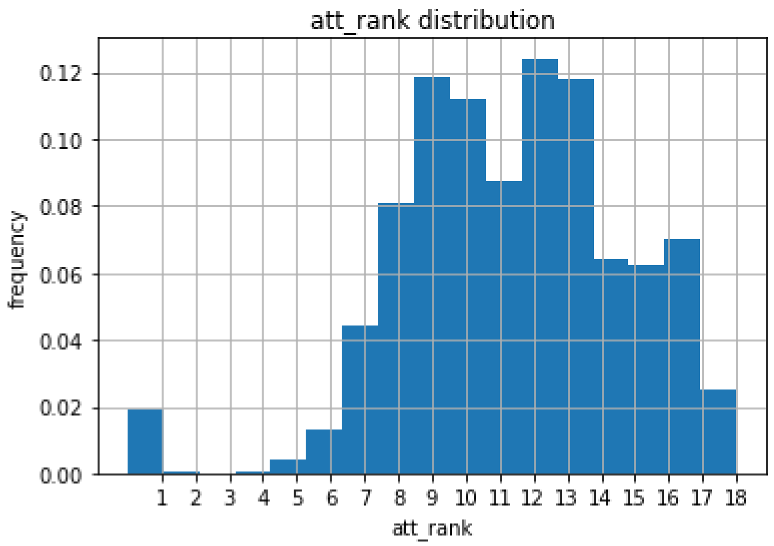
Counter-Strike Global Offensive (CS:GO) has been one of the most successful and popular FPS games for the past decade. There are also an increasing number of competitive gaming platforms in the industry and each of them has to rank players based on their in-game performance. Thus there exists the demand for a fair and intelligent model to make rank predictions for players. There are two important reasons why a fair model for rank prediction is demanded. We can use rank prediction as a Smurf Detector to create a better gaming experience for gamers. That is, when smurf happens, the algorithm can detect and prevent them. Moreover, rank prediction can be used to rank new players. In this project, we will use damage data downloaded from kaggle which contains 955,466 observations and 33 columns to deploy a regression machine learning model to predict rank for players based on their damage performance specifically. The data is collected from ESEA game demos.

Feature Description (**Appendix**)

There are several machine learning models built upon this dataset. Daniel Mazzone did a round outcome round prediction using Multi-Layered Perceptron. The machine takes into account T and CT economy, the map, whether the bomb is planted, and four other features and the model accuracy can reach to 70%, which is only slight better than random guessing. It is probably because features are limited to the game information, and not representing the players skills, which is one of the most important factors to consider when winning a round.

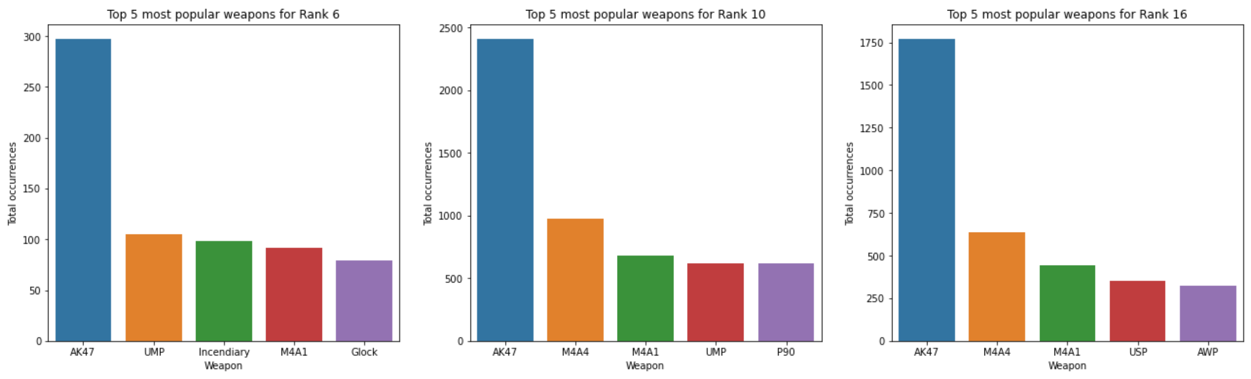
Exploratory Data Analysis

We are building regression models for rank prediction because there are sufficient categories and we are rounding the floating results to the nearest integers, or ordinal categories. The target variable att_rank is an ordinal variable that ranges from 0 to 17, where 1 being lowest rank and 17 being the highest. As we can see from the distribution of att_rank, it has the bell-shape without rank = 0, which represents those players who have not been ranked yet. We drop these data for future prediction; therefore, we are only focusing on ranked players from rank from 1 to 17.



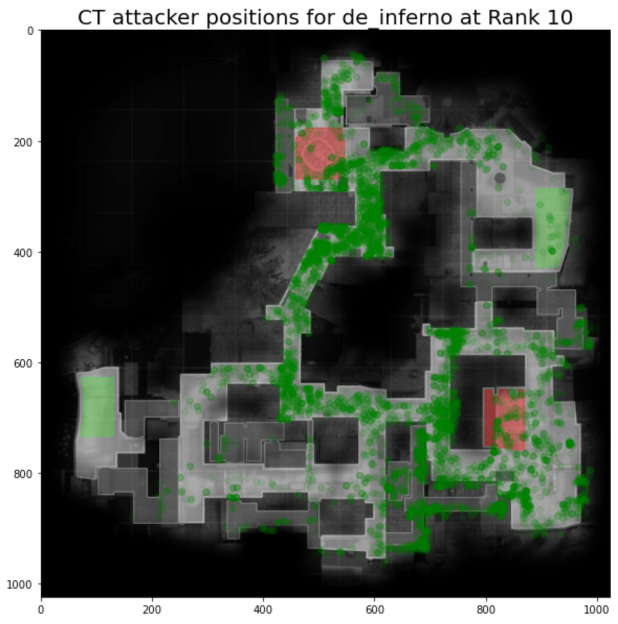
(figure 2.1 att_rank distribution showing a bell)

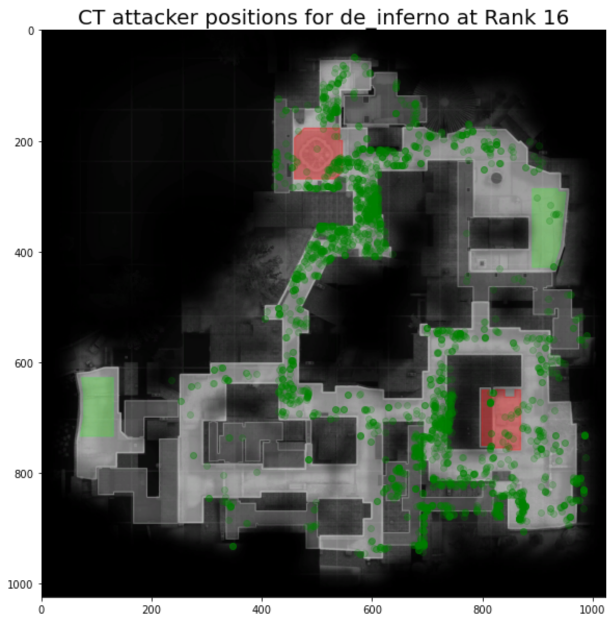
Figure 2.2 shows the relationship between the most commonly used weapons. Rank 6, Rank 10 and Rank 16 are selected to represent players at different levels. AK47 and M4's are the most popular guns across all three Ranks. As shown in the figure, people at lower ranks buy submachine guns (UMP, P90) more frequently than Rank 16, and Rank 16 are more likely to buy sniper rifles (AWP) than Rank 6 and 10. This is because submachine guns are easier to control in CS:GO and AWP requires much more gaming hours to master.



(figure 2.2 Top 5 most popular weapons for Rank 6, 10, and 16.)

The attacker's position is also a factor people often forget when predicting ranks. As a matter of fact, the attacker's position shows intuitive correlations with the target variable att_rank. That is, for players at Rank 10, the bottom left corner of the map is filled with dots, meaning CT are pushing from top right corner all the way to bottom left and causing damage near T spawn. This phenomenon is more uncommon for players at Rank 16, because, intuitively, higher level players are more experienced and are not willing to take risks to play aggressively to push.



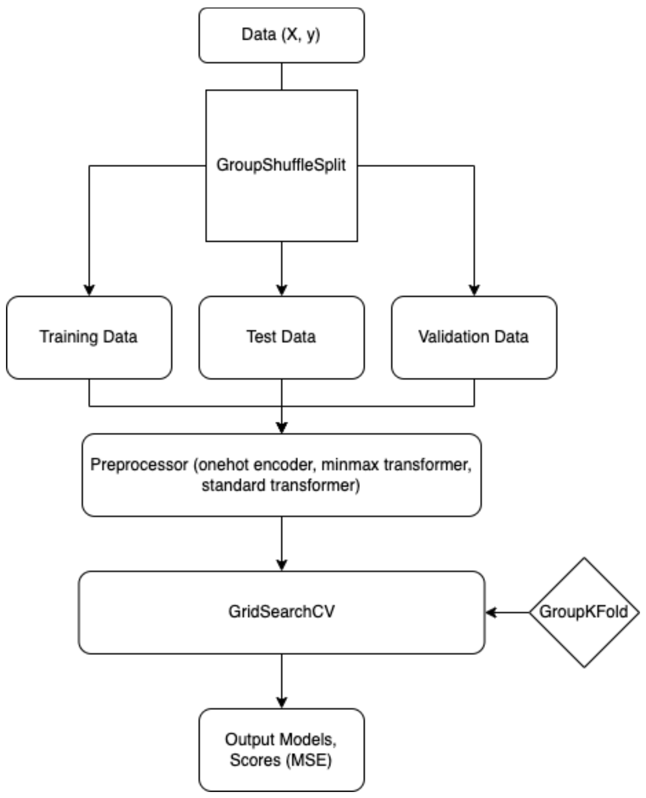


(left: figure 2.3CT attacker position for inferno at Rank 10. There are many observations from the lower left quadrant meaning lower level players are more likely to play aggressively.)

(right: figure 2.4 CT attacker position for inferno at Rank 16. There are much less observations from the lower left quadrant meaning higher level players are more thoughtful and are not willing to take risks as much as Rank 10 players.)

Methods

Our damage dataset contains damage observations, not players, which means there are multiple damage observations for each *att_id.* This makes the data non-iid since observations from the same player are highly correlated and this gives the dataset the group feature. Therefore, we are splitting out data using GroupShuffleSplit by *att_id.* This splitting process can benefit our future use as a new player comes in and requests for their rank as they make predictions after grouping by *att_id*. There is one column with missing values, *bomb_site,* and it is highly correlated with *is_bomb_planted*. The column *bomb_site* will have NULL values if and only if *is_bomb_planted* is False. Thus *bomb_site* will be dropped. For categorical variables, *map, att_side, is_bomb_planted, hitbox, wp_type, round_type, and wp*, we use One-Hot Encoding to quantify categories into vectors. An OrdinalEncoder is used to transform victim rank, *vic_rank*; like the target variable, *att_rank, vic_rank* has the same logic but is the rank only for victims, not attackers. For continuous variables, *seconds, hp_dmg,* and *arm_dmg,* we are using MinMaxEncoder to transform. This is because these variables have obvious minimum and maximum boundaries. Variable *seconds has* a time-series structure but it resets to 0 at each new round, which makes it extremely difficult to analyze as a time-series data, and there is not a sufficient amount of data available within each round. Therefore, we treat *seconds* as a continuous variable and use a MinMaxEncoder to transform. For variables hp_dmg and arm_dmg, it is intuitive to use a MinMaxEncoder since they are bounded by 0 and 100. We use the StandardScaler transformer for other continuous variables, *att_pos_x, att_pos_y, vic_pos_x, vic_pos_y, ct_eq_val,* and *t_eq_val,* given that either they have skewed distributions or are not clearly bounded.


Data (X, y)
GroupShuffleSplit
Training Data
Test Data
Validation Data
Preprocessor (onehot encoder, minmax transformer, standard transformer)
GridSearchCV
GroupKFold
Output Models, Scores (MSE)

(figure 3.1 Model Pipeline: from data splitting to evaluation)

Pipeline

We train the model using 5 different random states, 1 to 5. For each random state, we split the data into training, test, and validation using GroupShuffleSplit with ratio of 60, 20, 20. Then we fit transform training set, and transform the test set and validation set by pre-defined preprocessors. Then we deploy a GridSearchCV to assemble the pipeline. With the help of GridSearchCV, we are able to tune in each model using a corresponding parameter grid. Within the GridSearchCV, a GroupKFold is used for cross validation. Finally, the model is evaluated using MSE for comparison. To reduce the training time for the purpose of this project, we partition the data to 2% with the help of GroupShuffleSplit and use this 2% of data to mimic the behavior of all data points. After partitioning, there are 19109 data points; originally, there were 955466 observations.

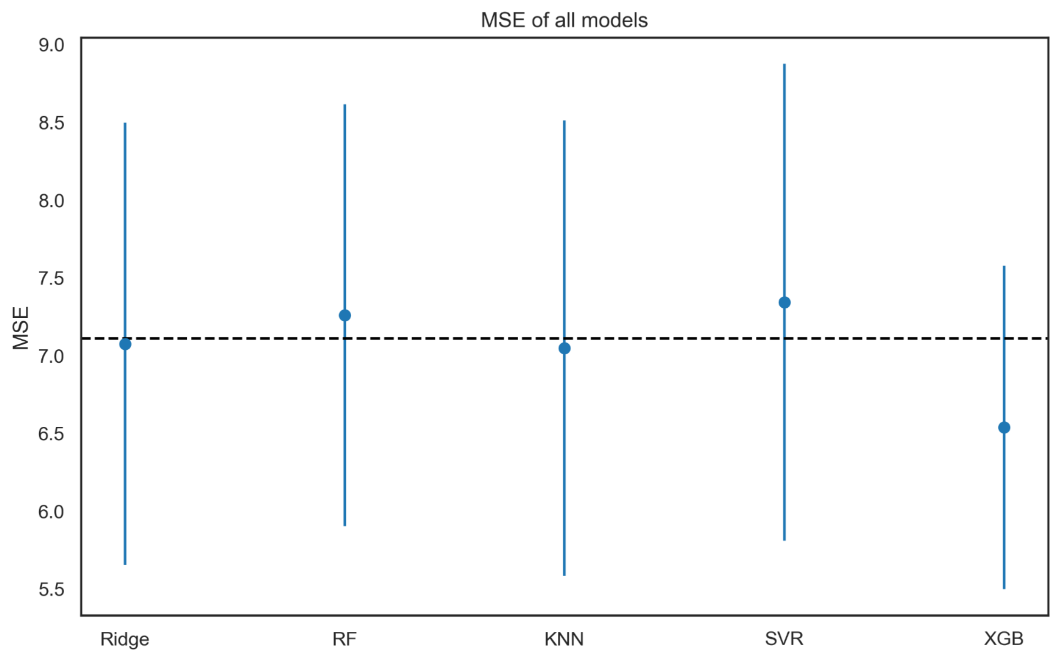
Models

Five different regression models are used for comparison. There are four models imported from scikit-learn, Ridge Regression, Random Forest Regressor, Support Vector Machine Regression, K-Neighbor Regression. In addition, an XGBoost Regression is deployed and it is directly imported from the xgb package.

Model	Parameters
Ridge Regression	alpha: np.logspace(-6, 6, 50)
Random Forest Regressor	n_estimators: [100, 200, 300, 400] max_depth: [5, 10, 15, 20, 30, 40, 50]
Support Vector Machine (SVR)	kernel: [sigmoid, rbf, poly] gamma: np.logspace(-6, 6, 10) C: np.logspace(-2, 2, 3)
K-Neighbor Regressor (KNN)	weights: [uniform, distance] n_neighbors: np.arange(300, 600, 10)
XGBoost Regressor	reg_alpha: [0.1, 1, 10, 100] reg_lambda: [0, 0.1, 1, 10] max_depth: [5, 10, 15] colsample_bytree: [0.6, 0.8, 1] subsample: [0.6, 0.8, 1]

(table 3.2 parameter selections for each model)

Table 3.2 shows the parameter grid selections within each GridSearchCV for the best performance model. Specifically, there is one linear model, Ridge Regression and all other models are non-linear. We use Mean Squared Error for model evaluations because it is an evaluation metric used for regression models to check how close estimates are to actual values. Intuitively, lower values indicate better fits.



(figure 3.3, uncertainty measures due to splitting and non-deterministic ML models. The errorplot shows the means and standard deviations of MSEs of five models given different random states.)

Figure 3.3 shows the uncertainty of each model. Obviously, XGB helps achieve a better model performance by both lowest mean and standard deviation of MSEs. All other four models have mean MSE near the baseline model, which is predicting the mean of the target variable for every entry. Despite the fact that Ridge regression and KNN have mean MSEs lower than the baseline, they do not improve the baseline model as much as the XGB model.

Results

The baseline mode MSE is 7.109, which is calculated by the MSE between predicting the mean of the target variable for every entry and the actual target variable. Figure 4.1 shows the mean and standard deviation of model MSEs. Obviously, XGB outputs the best model with an MSE score of 4.52190023, and it is 2.488 standard deviations away from the baseline model.

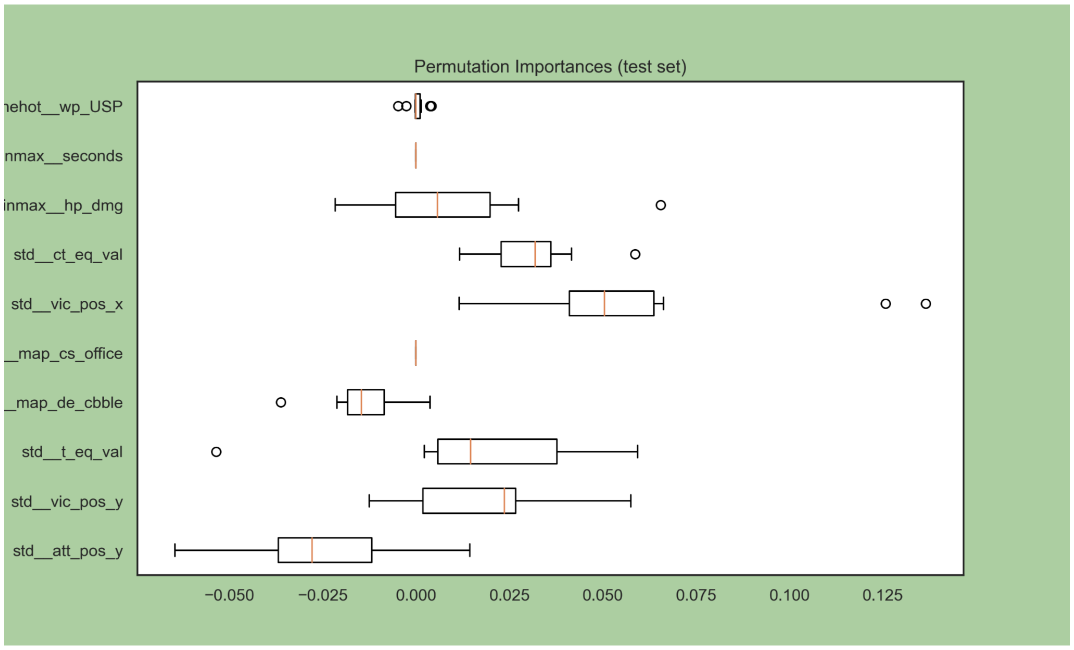
Model	MSE Mean	Best MSE Score	MSE Std
baseline	-	7.10932112	-

Ridge Regression	7.07648804	4.79392266	1.42098615
Random Forest	7.26025508	5.04766997	1.35692461
KNN	7.04855256	4.61950863	1.46221175
SVR	7.34381755	4.72840385	1.53271998
XGB	6.53970172	4.52190023	1.03991658

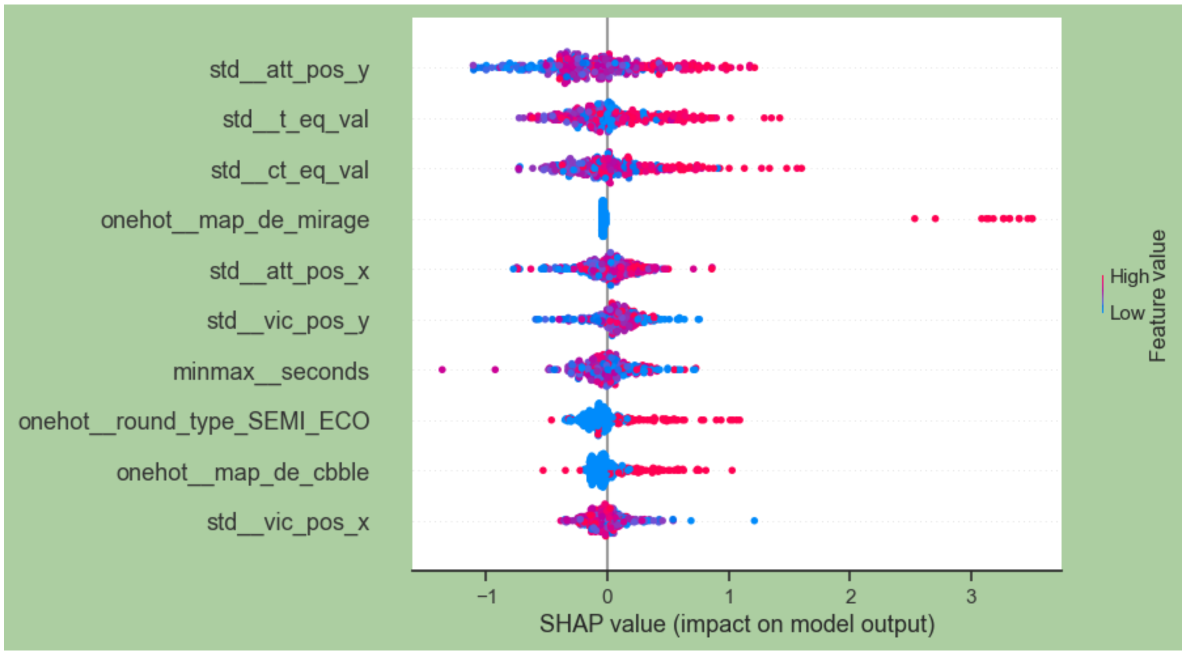
(figure 4.1 model performance. The table contains mean, std of MSEs for all five models and it also prints out the best MSE for each model.)

Global Feature Importance

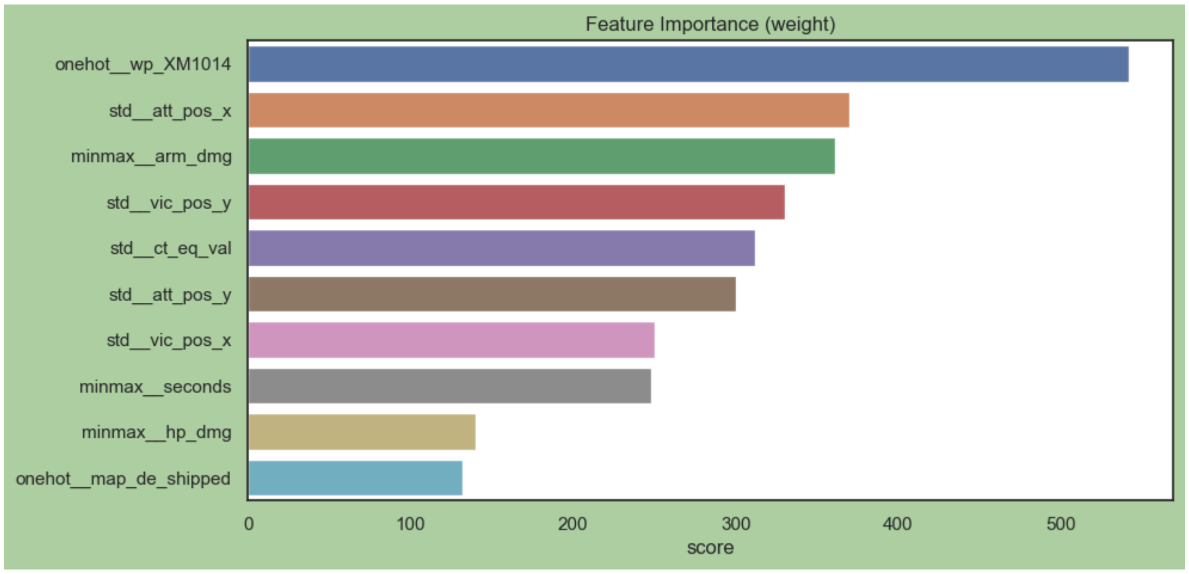
We choose three different methods to calculate the global feature importance: Permutation importance from scikit-learn, global shap values, and XGB built-in weight metric.



(figure 4.2 Permutation Global Feature Importance.)



(figure 4.3 Global Shap Values)

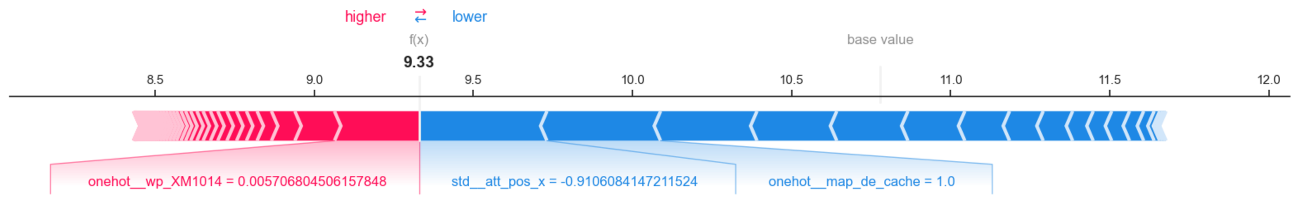


(figure 4.4 Feature Importance by weight metric)

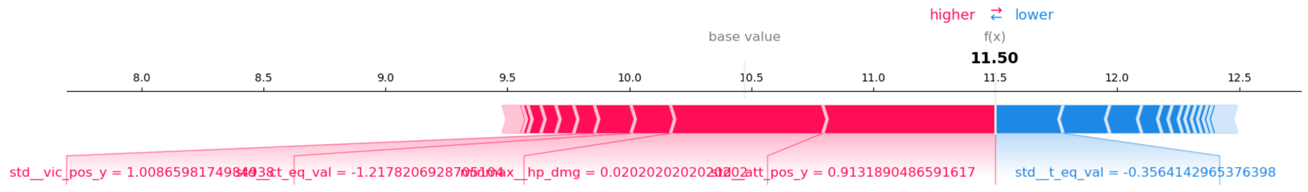
From figure 4.2 we can see that the victim's position of being attacked, vic_pos_x, vic_pos_y, play an important role in making predictions as well as the attacker's positions. Figure 4.3 and Figure 4.4 also verify that both the attacker's and the victim's positions are important in making predictions in the best XGBoost regression model. In addition, equipment value contributes to the prediction making process significantly. One intuitive interpretation is that if a player can make a lot of damage with low equipment value, they have a greater chance to be ranked at higher levels. There are also interesting findings such as weapon XM1014 is ranked the most important feature according to weight metric. This is interesting because XM1014 is a shotgun that is not bought frequently in game. This finding suggests that players at a certain rank are more likely to buy this weapon in game, and this characteristic is exclusive for these certain ranks.

Local Feature Importance

The local feature importances from data points at index 30 and 200 both show the attacker's position plays an important role in making predictions, regardless of positive or negative predictions. When we look at the local feature importance we can also find interesting information already suggested from the global feature importance above. The local importance suggests for index 30 that weapon XM1014 is the most important feature that determines the prediction positively.



(figure 4.5 Local Feature Importance at index 30)



(figure 4.6 Local Feature Importance at index 200)

Outlook

Although the parameter tuning can help achieve slightly better model performance, the results are not satisfactory. There are several issues with the pipeline. For example, our model does not summarize the information within a group. In fact, the damage observation within a group can vary, given the fact that a player can play both professionally and poorly within each round.

Another issue is that the rank prediction itself cannot be as accurate as we expect. This is because the difference between each rank is unnoticeable. In other words, the more games you play on a platform, the higher rank you can potentially reach; and ranks are also subject to the platform's ranking algorithm.
The potential improvement can be made by summarizing the group information and doing feature engineering. In addition, more data from lower ranked players could be collected to enrich the data. Finally, more robust models, such as Neural Networks can be used to capture the unnoticeable difference between each rank.

Reference

Daniel Mazzone, CSGO Data Analysis and Machine Learning, (2020), https://www.kaggle.com/code/danielmazzone/csgo-data-analysis-and-machine-learning

SIDDHARTH MANI TIWARI, CS:GO Economics and Positioning, (2021), https://www.kaggle.com/code/manitiw/cs-go-economics-and-positioning

Appendix

Data description:

The target variable is labeled *att_rank* in the dataset and it stands for attacker's rank.

map: categorical; a cs:go map for this game
- de_cache, de_cbble, de_dust2, de_inferno, de_mirage, de_nuke, de_overpass

date: ordinal; the date when game starts
round: continuous: specify the number of rounds
tick, seconds: continuous: the server time and actual seconds
att_team, vic_team: categorical; attacker's team name and victim's team name
stt_side, vic_side: categorical; attacker's and victim's side
- T, (terrorists), CT (counter-terrorists)

hp_dmg, arm_dmg: continuous; HP damage and armor damage
- integer range from 0 to 100

is_bomb_planted, bomb_site: categorical; whether and bomb is planted and where is it planted
hitbox: categorical; the victim's body part been shot
- Head, RightLeg, LeftLeg, Chest, Stomach, RightArm, Generic, LeftArm

wp, wp_type: categorical; weapon name and weapon type

- wp: AK-47, AWP, M4A4, etc
- wp_type: Pistol, Grenade, SMG, Rifle, Sniper, Equipment, Heavy

award: categorical; money award

- 300, 600, 100, 1500, 900

winner_team, winner_side: categorical; name of the winning team and its side

att_id, vic_id: attacker id and victim id

vic_rank: ordinal; victim's rank ranged from 0-18

att_pos_x, att_pos_y: continuous: attacker's x and y position on the map

vic_pos_x, vic_pos_y: continuous: victim's x and y position on the map

round_type: categorical; round type

- PISTOL_ROUND, ECO, SEMI_ECO, NORMAL, FORCE_BUY

ct_eq_val, t_eq_val: continuous; the equipment value for CT and T

avg_match_rank: continuous; *average match rank*