Problem statement

Counter Strike: Global Offensive (CS:GO) is a competitive multiplayer first person shooter game developed by Valve and Hidden Path Entertainment. Since published in 2012, CS:GO sells over 25 millions of copies while continues to be one of the most-played game in the world. However, one issue that always plagues the game since release: cheating.

Cheaters uses a variety of techniques/programs to gain an unfair advantage. Machine learning algorithm can detect these anomalies very well. In fact, Valve is currently using a deep learning neural net called VACnet to help fighting against cheaters. We will attempt to build a similar neural net but at a much smaller scale.

This project will apply machine learning to detect cheaters in CS:GO. The algorithm will give a probability of the player is a cheater based on his lifetime in-game performance and some characteristics of the steam profile. If the probability is high, replays should be reviewed by other players to determine the player's legitimacy. The result should not be used for banning players. Banning a legitimate player means losing a paying customer.

Data Collection

Fortunately, steam platform which hosts CS:GO provides an API which makes collecting data much easier. Registered Steam users are identified by a unique Steam ID, which can be used to retrieve public player information, in-game metrics, and instances of bans due to cheating detected by Valve Anti-Cheat (VAC). VAC banned status will become the label where True represents cheater and False represents non-cheater.

How do we get a list of steam IDs that contains both cheaters and non-cheaters? Steam does not have that list (at least publicly). One approach would be crawling multiple lists of steam users who play CS:GO and hope for the best. However, most players are not cheaters and thus, this simply takes too much time.

The other approach is much more elegant. Steam communities has built many websites to check if a player is VAC banned in the past. For this project, VACBanned.com and VAClist.net will be used. If a player is suspicious in game,

other players can search him on these websites for his VAC status. Thus, the steam IDs from these websites will contain more true cheaters.

API Process

After obtaining a list of steam IDs, each is passed through four API calls for data collection:

1. GetPlayerSummaries

- communityvisibilitystate that flags if a profile is private or public.

- If private: no statistics available, skip to next steam ID.

- If public: continue to the next call.

2. GetOwnedGames

- playtime forever: a less experienced player who performs too well is suspicious.

- playtime last 2 weeks: helps labeling. If an account is VAC banned from CS:GO, they cannot play CS:GO again. Thus, if playtime last 2 weeks is greater than 1, change the label to False.

- number of games owned: a player is less likely to cheat if he has multiple games (purchased with money) in his library.

3. GetPlayerBans

- VAC banned status: become labels.

4. GetUserStatsForGame

- returns a list of hundreds of statistics on player's performance.

Labeling Issues

There are two main problems with using VAC ban status as the label:

1. A player can be banned at other games beside CS:GO. This remains one of the biggest flaws but this is the best label we can do given no internal data from Valve. However, CS:GO is the most played on steam for many years by a large margin. It is also the most vulnerable to cheat programs which increase the

probability that a player is VAC banned from CS:GO. The second most played game is Dota 2 which limits the possibilities of cheats due to almost everything being server side.

2. Cheaters can manually adjust their program so they can appear innocent. If they are clever enough in the settings, it is impossible to detect them without identity which programs enable they hack in the background.

3. VAC ban status is not 100% reliable. There are many cases of wrongly banned accounts and there are many cheaters' accounts are not banned. But Valve has improved VAC significantly in the last decade which saw a conviction rate of up to 80-90% True positive.

Features engineer

The API can returns over 250 features for each user in CS:GO. This means that some thoughtful cleaning must be done. As advised by players, many calculations below give a better understanding of a player performance. The final number of features is 190.

accuracy_{gun_type} = total_hit_{gun_type} / total_shot_{gun_type}

win_ratio = total_wins / total_rounds_played

kill_to_death_ratio = total_kills / total_deaths

total_wins_per_hour = ((total_wins / total_time_played) / 60) / 60 (seconds -> hours)

mvp_per_round = total_mvps / total_rounds_played

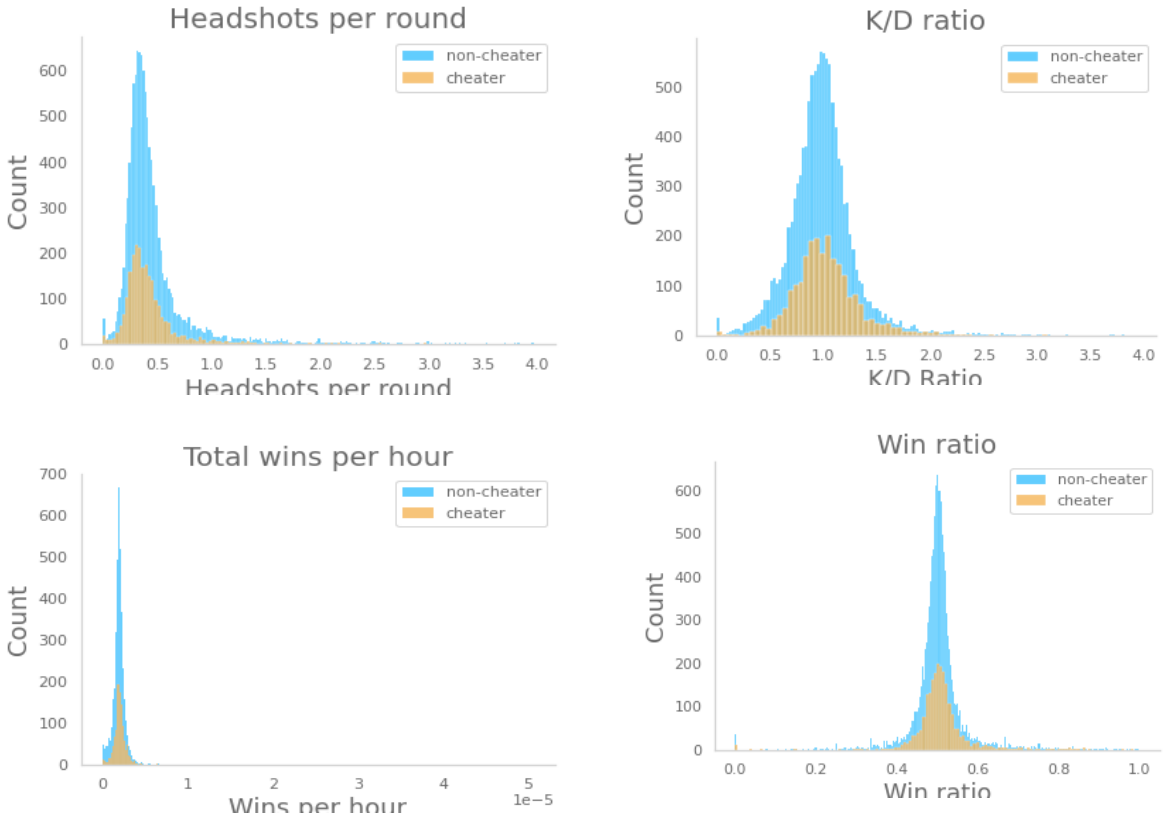
total_headshots_per_round = total_kills_headshot / total_rounds_played

Some exploratory data findings

It is very hard to identify cheaters by looking at separated factors. The reason for this is threefold:

1. Free-to-play
2. Snappy combat
3. Easy account creation and easy to run

A quick look at these simple figures below shows vertically identical performance metrics between cheaters and non-cheaters.



However, a combination of these factors is useful and give a better chance of detecting cheaters. But first, lets fix the unbalanced data issue.

Imbalanced data and sampling methods

As with any fraud detection kind of problem, the data is imbalanced as cheaters are the minority. Our data also scraped using websites to check for cheaters which means the percentage of cheaters is also higher than normal. Some model adjustments and sampling methods are used to combat against the imbalanced.

Model adjustments:

1. Initial bias - reduce learning time and loss

2. Class weights - gives more emphasis to TRUE label

Sampling methods

1. Down sampling

- Random under-sampling: randomly removes observations of the majority class.

- Near-miss: select examples based on the distance of majority class examples to minority class examples.

2. Up sampling

- Random over-sampling: randomly duplicates observations from the minority class in order to make its signal stronger.

- SMOTE: varies attributes of the observations to create new synthetic samples.

Models

Each data set from samplings is fitted with 3 algorithms:

- Logistic Regression: serves as a baseline

- Random forest: tree based high performing algorithm

- Neural network: dense 5 layers NN

Metrics

Since accuracy is not a good indicator for model performance in imbalanced data cases, some of the useful statistics used are:

1. Precision = TP / (TP + FP)

2. Recall = TP / (TP + FN)

3. ROC-AUC curve

While this is a classification problem, outputs in the form of probability is more useful here for a practical reason: threshold for cheater label can be manually adjusted.

This is extremely useful to solve the label uncertainty issue. Banning a non-cheater is much worse than not banning a cheater in this case. Thus, False Positive rate must be very low for the model to be considered successful. If the probability of cheating is high, manual review of the player is recommended.

Results

Unfortunately, the model is not deployed on a server to get a better understanding of how this work. However, we learn that by manually adjusting the probability threshold, we can adjust the precision and recall manually to the accepted rate (which is to be determined by the developers). Moreover, over-sampling methods are working splendidly in this case. More data would saturate and improve the neural network significantly.