Predicting CS:GO Round Winners With Deep Learning

Andrew Dettor
University of Rochester
Rochester, New York
adettor@u.rochester.edu

Abstract

CS:GO (Counter-Strike Global Offensive) is a popular video game that's played competitively in tournaments. Live betting services allow people to gamble on competitive CS:GO matches while they're happening. Deep learning methods could allow one to place more accurate bets. This project uses an LSTM to predict which team will win the next round of a given match with ~65% accuracy, which is above the baseline of 50%. Previous attempts at this problem suffered from methodological errors, which were amended. Two feature importance methods were employed to try to help competitive players get better at the game. They elucidated features that correlate with round wins, but don't necessarily cause round wins.

1. Introduction

1.1. Live Betting

Live Betting services allow people to gamble on sports events that are in-progress. This contrasts with traditional sports gambling, where bets are locked in place before the game or match starts. E-sports events (competitive video gaming) also have live betting services.

1.2. CS:GO

One of the most popular E-sports is Counter-Strike: Global Offensive (CS:GO for short). CS:GO is a 5v5 tactical first-person shooter where Terrorists try to plant a bomb and Counter-Terrorists try to stop them. A match of CS:GO is separated into rounds, where each round is a few minutes long. The Terrorists (T's for short) can win a round by eliminating all the Counter-Terrorists (CT's for short), or by planting and exploding a bomb at a pre-defined bombsite. The CT's can win a round by eliminating all the T's or by defusing the planted bomb before it explodes. The teams switch sides after 15 rounds. The first to win 16 rounds wins the match.

1.3. Improving at the game

CS:GO isn't very complex; it is extremely deep. The concept is simple but mastering the game can take many years or even decades. As you play, your character doesn't get better; you do. As such, players are always looking for ways to improve. Many hope that someday they can compete in E-sports events for their large prize pools.

1.4. Goal of this project

The goal of this project is to predict which team (T or CT) will win the next round, given the game thus far. Ideally, one can use this to place more accurate live bets. Additionally, with model explainability methods, one can improve their own skill at the game by analyzing what features lead to more round wins.

2. Related Work

2.1. Predicting match winners

Much of the related published research focused on predicting which team will win the entire match. Björklund et. al. [1] achieved ~65% accuracy using a multi-layer perceptron. Their approach was to cluster individual players into classes and use those clusters as features. Ondřej et. al. [2] aggregated round-based data and achieved 63% accuracy with a Random Forest model.

2.2. Predicting round winners

Rubin, Allen [3] used players' weapon and equipment choices at the start of each round and the current score of each team to achieve 64% accuracy with a Random Forest. Lillelund, Christian [4] used snapshots of individual rounds to predict which team won the round with ~81% accuracy using a multi-layer perceptron. Huang et. al. [5] used the same dataset as [4] and predicted with 79% accuracy using XGBoost.

These results look promising, but there is a major problem with [4] and [5]'s dataset and methodologies. Both assumed the round snapshots were i.i.d.. As a result, many snapshots of the same round and many rounds of the same match were present in the train and test sets, causing target leakage. Rubin, Allen [3] did not publish their dataset, but this issue was not addressed in their report, meaning they likely also had rounds of the same match present in the both the train and test sets. Predicting the winners of rounds is fundamentally a time-series problem, which needs to be properly addressed for reliable results. No previous analysis has accounted for this issue.

3. Methods

3.1. Dataset

The dataset for this analysis was taken from Kaggle.com. It was uploaded by user "KP" in 2018, and contains data from 14,921 matches, spanning 377,629 rounds, so that's about 25 rounds per match [6].

It has information about grenade throws, damage incurred, kills, bomb plants, round lengths, weapons and equipment used, and most importantly, which side won each round. Some attributes were essentially snapshots of features of rounds and some rounds were missing entirely.

KP parsed this data from a third-party CS:GO matchmaking service called ESEA. ESEA is a paid service, so its players are probably more serious than the average CS:GO gamer, meaning their matches more closely align with professional play.

3.2. Problem Formulation

This is a multi-channel variable-length multiple time-series binary classification problem with missing data and missing labels. Here the binary classification was 1 for CT's winning the next round and 0 for T's winning the next round. There are many covariates with the round winner: there is an arbitrary number of rounds in a match, there are many matches, the rounds in a match form a time-series, and some rounds are completely missing. To make the data ready for modelling, data were aggregated to the per-round level, missing rounds were filled with the mean of each feature for that round number, and missing labels were masked when calculating models' performance.

3.3. Modelling Approach

To account for the time-series nature of this problem, an LSTM (Long-Short Term Memory) model was used. LSTM's can accept variable-length 3D input, where the input is of shape matches×rounds×features. Matches can be batched by the number of rounds they have, so no padding is needed. The general structure of the LSTM was a GRU (Gated Recurrent Unit) followed by a Dense layer followed by a Sigmoid function. See Figure 1 for a visual of the architecture. The model was optimized using Binary Cross-Entropy Loss.

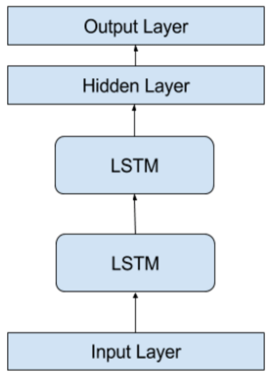

Figure 1: Model architecture [7]

The model was fed N rounds of data to predict the winner of round N+1. N here ranged from 1 to 29, since most matches end at 30. To illustrate, at the start of each epoch round 1 of every match was trained on and the winner of round 2 was predicted, then rounds 1 and 2 were trained on to predict the round 3 winner, and this continued until rounds 1 through 29 were fed in and the round 30 winner was predicted. See Figure 2 for a more visual explanation. Each match was seen multiple times within an epoch, but the model never saw future data from the same match. This vastly increased the amount of data, since a single 30-round match would be used 29 times. Matches were randomly shuffled just in case they had any sort of ordering.

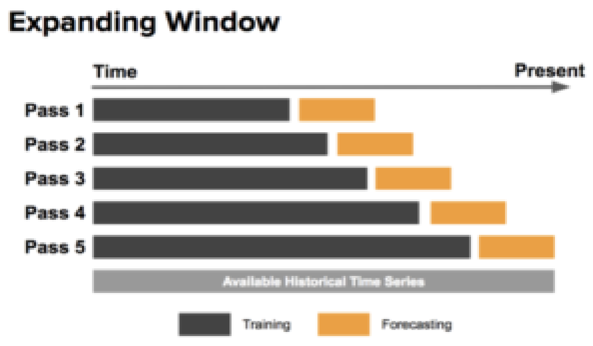

Figure 2: Time-series training procedure [8]

It is important to note that subsets of matches are highly collinear, so using many subsets could lead to overfitting. However, deep learning models see the same data multiple times across epochs anyway. Overfitting was unlikely to occur for that reason and because of the large sample size of matches.

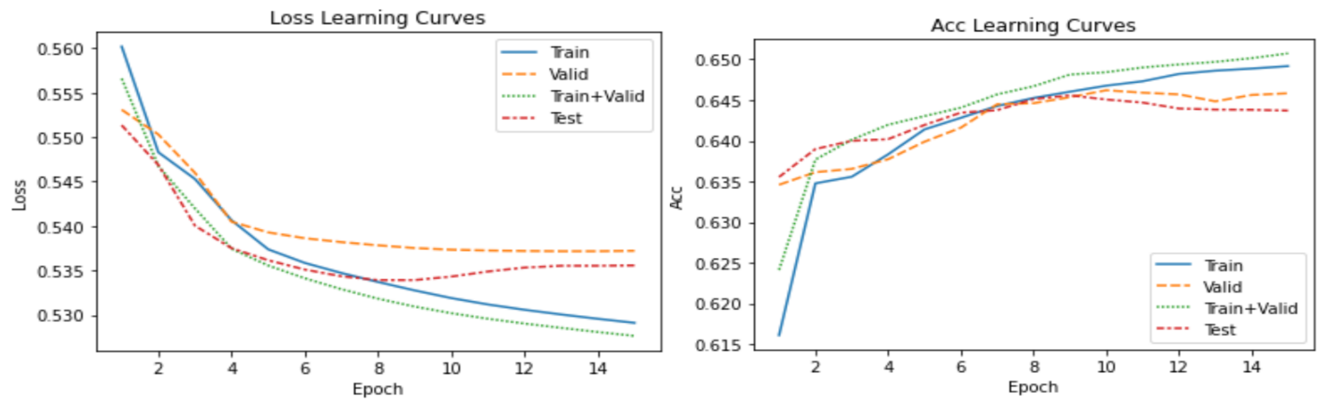

Figure 3: Accuracy and loss learning curves with optimal hyperparameters.

3.4. Validation Strategy

Matches were stratified by their total round length. Within each stratum, 70% were put into the training set, 15% in the validation set, and 15% in the test set. This strategy accounted for the fact that longer matches would be sampled more times, potentially making the final train/test/valid splits different from the desired 70:15:15 ratio. Furthermore, stratifying by matches ensures that no time-series is in both the train and test sets, eliminating the possibility for target leakage.

Grid-search was employed to find the optimal hyperparameters. The hyperparameters here were batch size, learning rate, number of layers in the GRU, and hidden size of the GRU.

3.5. Feature Importance

Two feature importance methods were employed to see what features lead to more round wins. One was model-based and the other was model-agnostic, meaning they could potentially highlight different important aspects of the feature space.

The model-based method used the average gradient of the loss with respect to each feature. This is possible thanks to backpropagation within the neural network. The gradient of the input data was calculated, then averaged over the matches and rounds dimensions. This isn't perfect, since data with more rounds was represented disproportionately in the average, but it's a good proxy. Theoretically, higher magnitude gradients are more "important", since they have more impact on the loss.

The model-agnostic method used was permutation importance. Within each batch, the values for a certain feature were randomly shuffled. This theoretically breaks the relationships between that feature and all the others. The whole objective of predictive modelling is to map relationships between variables, so if doing this doesn't significantly increase the loss, then that feature isn't important to the model. This procedure is done for every feature.

4. Experiments

4.1. Validation Results

The optimal set of hyperparameters was found to be a batch size of 256, 2 GRU layers, 16 hidden nodes in the GRU, and a learning rate of 0.001. This combination had a validation loss of 0.537 and a validation accuracy of 64%. Each combination of hyperparameters was trained with for 5 epochs. See Figure 3 for the training and validation sets' learning curves.

4.2. Test Results

The final model was trained for 7 epochs using the optimal hyperparameters, since training for longer led to overfitting. The training set here was the previous train+valid sets, so 85% of the original dataset. The model achieved 64.5% accuracy on the test set, with 59% precision and 63% recall. So out of all the positive cases, the model predicted correctly 63% of the time, and when the model predicted positive, it was correct 59% of the time. See Figure 3 for the train+valid and test sets' learning curves.

While these results aren't amazing, they are certainly better than the baseline accuracy of 50%. This approach had worse performance than [3,4,5], but it was more methodologically sound.

Feature Importance Results

The most important features using the model-based feature importance method were the winning side of each round, Terrorists' pistol kills and damage, round length, and Counter Terrorists' incendiary grenade hits. The model-agnostic method only really picked up on the importance of the winning side feature, probably because it's viewing the model from an outside perspective. See

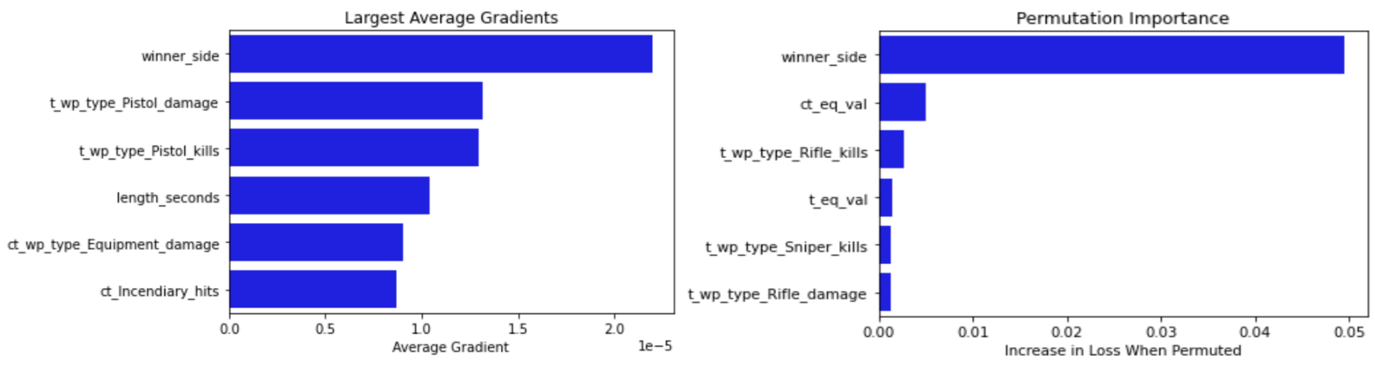

Figure 4: Top 6 results from each feature importance method.

Figure 4 for the top 6 most important features according to each method.

These features all make sense in the context of the game. Obviously, if a certain side has won a lot so far, it's more likely they'll win the next round. The first round of each half is the pistol round, and it's extremely important for the outcome of each half since the winner of the pistol round gets more money to spend on items on round 2. This can have a snowball effect for the rest of the half. When CT's are doing well they can afford the expensive incendiary grenade, which is very effective at stopping T's, meaning CT's are more likely to win the round. Furthermore, the CT's job is to fend off the T's as long as possible, so if the rounds are longer, they are more likely to win.

Overall, these feature importance results capture much more correlation than causation. There isn't much to glean from them to improve one's own gameplay, but they help validate the model's integrity since they align with intuition.

5. Conclusion

The model had fair results, but it's probably not good enough to use in a real context. Yes, 65% is better than 50%, but a 35% chance of losing money is still risky.

Furthermore, using this model in a live context would prove difficult. One would somehow have to get into the tournament match being played to collect the appropriate data. It very well might be possible, but the game's infrastructure would make it exceedingly difficult.

There are many changes that could be made to improve performance. More granular data collection and better feature engineering would provide more signal to the model on what truly causes one team to win over the other. The dataset used here had little info on the individual players, which is extremely important in team-based games. Additionally, one could engineer features to capture teams' movement patterns or other gameplay patterns. There also could be changes to how the problem was conceptualized that would improve performance.

References

[1] A. Björklund, W. J. Visuri, F. Lindevall, and P. Svensson, "Predicting the outcome of CS:GO games using machine learning," 2018. https://www.semanticscholar.org/paper/Predicting-the-outcome-of-CS%3AGO-games-using-machine-Bj%C3%B6rklund-Visuri/4d6da1977825705540e19046999708d29226db70 (accessed Dec. 20, 2022).

[2] O. Švec, "Predicting Counter-Strike Game Outcomes with Machine Learning." Czech Technical University in Prague Department of Cybernetics, Jan. 2022. [Online]. Available: https://dspace.cvut.cz/bitstream/handle/10467/99181/F3-BP-2022-Svec-Ondrej-predicting_csgo_outcomes_with_machine_learning.pdf

[3] A. Rubin, "Predicting Round and Game Winners in CSGO," Center for Open Science, OSF Preprints u9j5g, Jan. 2022. Accessed: Dec. 20, 2022. [Online]. Available: https://econpapers.repec.org/paper/osfosfxxx/u9j5g.htm

[4] "Predict winners in CS:GO with Keras [80%]." https://kaggle.com/code/christianlillelund/predict-winners-in-cs-go-with-keras-80 (accessed Dec. 20, 2022).

[5] W. Xu Huang, J. Wang, and Y. Xu, "Predicting Round Result in Counter-Strike: Global Offensive Using Machine Learning," in *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, Apr. 2022, pp. 1685–1691. doi: 10.1109/ICSP54964.2022.9778597.

[6] "CS:GO Competitive Matchmaking Data." https://www.kaggle.com/datasets/skihikingkevin/csgo-matchmaking-damage (accessed Dec. 20, 2022).

[7] O. Aydin and S. Guldamlasioglu, "Using LSTM networks to predict engine condition on large scale data processing framework," in *2017 4th International Conference on Electrical and Electronic Engineering (ICEEE)*, Apr. 2017, pp. 281–285. doi: 10.1109/ICEEE2.2017.7935834.

[8] Novawaly, "Is there a way to get a Sliding Nested Cross Validation using SKlearn?," *Stack Overflow*, Jun. 14, 2019. https://stackoverflow.com/q/56601488 (accessed Dec. 20, 2022).