Project Deliverable 2: Predicting round outcome in Counter Strike:® Global Offensive

Robert Statham

1 Introduction

Counter Strike:® Global Offensive (CS:GO) is a team-based tactical first-person shooter played in rounds. Each match involves two teams of five players playing a best of 30 rounds. At the start of a match, the two teams are randomly assigned to play either Terrorist side (commonly referred to as T-side) or Counter Terrorist side (CT-side). The objective for the team assigned to T-side is to plant the bomb-- which one player must have in their inventory-- at one of the two bomb sites and successfully detonate it. Expectedly, the objective for the CT team is to disarm the bomb if one is armed, before the bomb detonates. However, a team automatically wins the round if all of the members of the opposing team are eliminated. After 15 rounds have been played, the two teams swap sides (e.g., T-side now plays CT-side). As the match progresses, each player will earn cash for completing certain actions such as winning the round, eliminating an opponent, planting a bomb, or defusing a bomb. This cash can be used at the beginning of a round to purchase more powerful gear such as body armor, grenades, or various guns ranging from pistols, to shotguns, to sniper/marksman rifles. As one may expect, how a team uses their cash supply and what items they use said cash on are vitally important to whether they win a given round.

Using machine learning, we hope to develop a model capable of accurately predicting which team will win the round given data taken in snapshots during a round. Examples of features included in the data are how many members of each team are alive, the combined hit-points (health) of each team, whether or not T-side possesses bomb, number of players on CT-side with defusal kit, as well as what number of each gun is present on each team.

Considering CS:GO is a game and, in some ways, a sport, it is very difficult—nay, impossible—to deterministically predict the outcome of a round before it is over. Knowing this, there has not been a systematic method of determining match outcomes developed ever. So, analysts and commentators must resort to educated guesses when predicting match outcomes. It must be noted, however, that using machine learning will not change the fact that CS:GO is non-deterministic, meaning, our model will not be able to predict things like clutches, mistakes, or human error. Our model also cannot factor in the specific players making up each team or the players relative skill to one another.

It is important to examine this problem using machine learning because it allows us to gain useful insights into what elements of a team contribute most to the outcome of a round. For example, we may find that the number of players using a certain gun on a team will substantially impact the round outcome. Expectedly, our model will be most useful to analysts who work for professional CS:GO organizations looking to give themselves and edge over the competition.

To achieve our desired result, we will of course start by analyzing and adequately splitting the data. Next, we will apply and evaluate various classification models to determine the one best suiting our problem. Then, within this selected model, we will analyze the model's hyperparameters and determine which hyperparameters will be changed; these changes will be tested on our validation data set. All the while we will make sure to use cross-validation when evaluating our models and hyperparameters. Once the most optimal model and hyperparameters are found, we will finally evaluate our model on our test data set and examine and interpret our results. We hope to finish this project producing an accurate model useful to esports large organizations as well as interested fans.

2 Literature Review

Below we have collected three papers discussing topic related to our problem.

2.1 Xu Huang et al.

The work by (Xu Huang et al.)[1] set out to solve the exact same problem as us, as well as used the same dataset that we have selected. However (Xu Huang et al.) also adds another feature not included in the original dataset; they include two more features that quantify the average player skill of either team using a method called *"TrueSkill."* Next, for feature selection, (Xu Huang et al.) calculated the correlation between every two features and condensed those which were highly correlated. (Xu Huang et al.) trained several different models both including the *TrueSkill* metric as well as without. Such models used are Decision Trees, XGBoost, Logistic Regression, and Neural Networks. However Neural Networks were the only model that benefitted from the addition of the *TrueSkill* metric with an increase of about one percent. (Xu Huang et al.)'s optimal model was a Neural Network with two hidden layers consisting of 100 neurons. This optimal model achieved an 83% accuracy on the training set and a 78% accuracy on the test set.

2.2 Ondřej Švec

This baccalaureate thesis from (Ondřej Švec)[2] aims to tackle a similar problem as us in that they aimed to predict the overall match outcome however we aim to predict the individual round outcomes. Another difference is their source of data. (Ondřej Švec) acquires their data by scraping the HLTV website. HLTV is a website containing data of every important match ever played in the competitive scene. (Ondřej Švec) chose this because of its potential for a much larger amount of data than can be acquired elsewhere. (Ondřej Švec) trained several different machine learning models consisting of three non-neural network models and three neural network models. The non-neural network models trained where Logistic Regression, Random Forest classifier, and k-Nearest Neighbors; the three neural networks trained where the linear model, the convolutional model, and the embedding model. All models were evaluated using accuracy. The random forest proved to be the best non-neural network model whereas the convolutional neural network proved to be the best out of the neural networks. The random forest scored an accuracy of 99.8% on the training set but only 63% on the test set which indicates some potential overfitting. The convolutional NN scored a 66.4% on the training set and a 59.8% on the test set.

2.3 BJÖRKLUND, ARVID, ET AL.

This baccalaureate thesis written by (Björklund, Arvid, et al.)[3] sets out with the same goal as the previous paper, to predict the overall match outcome as opposed to the outcome of individual rounds. However, (Björklund, Arvid, et al.) acquired their data by downloading it from a popular 3rd party host of competitive CS:GO matches called FACEIT. (Björklund, Arvid, et al.)'s files included data from 6000 games from the top 1000 ranked players in the European Union region. (Björklund, Arvid, et al.) used a feedforward neural network for simpler training. (Björklund, Arvid, et al.) used a *kmeans* clustering to cluster the data and reported results for k=8, 16, and 32 clusters. Optimal weights were determined using an evaluation algorithm and results were also given using different methods for weight determination. (Björklund, Arvid, et al.)'s best performing model achieved an accuracy of 65.11% on the test data.

3 DATA EXPLORATION

Our dataset was collected as part of the CS:GO AI Challenge hosted by Skybox, who originally published this data. The data was collected by taking 'snapshots' of a round every 20 seconds from professional matches that were played between 2019 and 2020. Our data contains no player information therefore it does not include any distinct demographics. In our classification task, our data's distribution is right down the middle: includes data where both sides win equal amount of times.

Our dataset is made up of 120,000 individual records with 97 features each. Below is a table of our features and their descriptions.

Variable	Definition	Key	Variable	Definition	Key
`time_left`	The time left in the current round in seconds.		`X_helmets`	Number of helmets on the X team.	e.g., Number of helmets on the Counter-Terrorist team.
`ct_score`	The current score of the Counter-Terrorist team.		`ct_defuse_kits`	Number of defuse kits on the Counter-Terrorist team.	Defuse kits allow the CT-players to defuse the bomb faster
`map`	The map the round is being played on.	E.g. de_dust2, de_inferno and de_overpass	`X_players_alive`	Number of alive players on the X team.	Range 0 to 5.
`bomb_planted`	If the bomb has been planted or not.	False = No, True = Yes	`ct_weapon_X`	Weapon X count on Counter-Terrorist team.	E.g., Ak47, Deagle and UMP45.
`ct_health`	The total health of all Counter-Terrorist players.	Player health in range 0-100	`t_weapon_X`	Weapon X count on Terrorist team.	E.g., Ak47, Deagle and UMP45.
`t_health`	The total health of all Terrorist players.	See above	`ct_grenade_X`	Grenade X count on Counter-Terrorist team.	E.g., Grenade, Flashbang.
`X_armor`	The total armor of all X team's players.		`t_grenade_X`	Grenade X count on Terrorist team.	E.g., Grenade, Flashbang.
`X_money`	The total bankroll of all X team players.	Amount in USD.	`round_winner`	Winning team of the round	CT = Counter-Terrorist, T = Terrorist

4 EVALUATION METRICS

Considering our task is one of classification and that our task deals with labels that are of equal weight (i.e., we are not testing for diabetes where one label is much more important than the other), we will be evaluating our models on their accuracy. However, we will still consider other classification metrics—such as recall, precision, and F-1 score—to gain a more wholistic view of our results.

5 BASELINE MODEL

Because our model will be evaluated on its accuracy, it is appropriate to use the Most Frequent Class or Majority Class Baseline. The Most Frequent Class baseline is simply a classifier which determines which class appears the most in the training set and classifies all test data as that majority class. For example, let's assume that more cases of Terrorist round wins appear in our training set, then our Majority Class baseline would label all data in the test set as Terrorist win.

Establishing a baseline will allow us to evaluate the performance of our models in comparison to another model. Implementing the majority-class baseline on our dataset with stratification, the baseline achieved an accuracy of 50.9%. Additionally, the results of the papers discussed above will also serve as baselines for our model. To review, Xu Huang et al., Ondřej Švec, and Björklund, Arvid, et al. achieved accuracies of 78%, 63%, and 65.11% on their test sets respectively.

6 METHODOLOGY

In order to reproduce, or better yet, exceed the results of our baselines, we mimic the work of Xu Huang et al. by implementing their feature of *TrueSkill*. Below we will outline how we do so, as well as, how we organize our data, the properties of our data, feature selection and engineering, what models we use, and how we select our hyperparameters.

6.1 SPLITTING OUR DATA

For models where we are able to use *GridSearch*, we only have to split our data into a training and test set. In this, 70% of our data was reserved for training, and 30% was reserved for testing our models.

For the one model we weren't able to use *GridSearch*, we split our data into three distinct groups: a training, validation, and test set. The training set was be made up of 60% of our total dataset and the test and validation sets were each 20% of our total dataset. We were also sure to preserve the proportionality of our dataset (i.e., ensuring that in all our datasets, there is a 1:1 proportion of rounds whose outcome is Terrorist win and Counter-Terrorist win).

6.2 PROPERTIES OF OUR DATA

Because our data contains multiple 'snapshots' of the same round, we have a potential for our model to cheat, wherein the model will look only at the 'snapshot' earliest in the round and label all future snapshots as that outcome. However, our dataset does not record the date/time that these snapshots were recorded, therefore, we can consider our data to be Independent and Identically Distributed (I.I.D.).

6.3 Feature Management

Due to the large number of features included in our dataset, it is necessary to discuss our strategy in managing them.

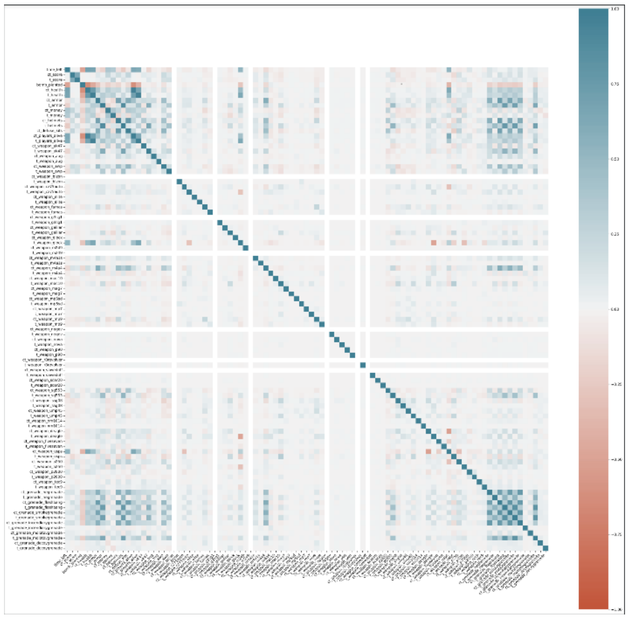
6.3.1 *Feature Scaling*

We will scale all our features to ensure no outlier data hurts our model. We do this because some features such as "`time_left`", "`ct_health`", and "`t_health`" since their values can range from 0 to 500 while most of our other features range from 0 to 5.

6.3.2 *Feature Selection*

Due to our large number of features, we must select features we believe will have little impact on the outcome of the round and eliminate them from consideration. To do this, we rely on expert knowledge in that we will consult with a former professional CS:GO player to get their opinion on which features are superfluous. This results in the elimination of the columns deemed so.

To this end, we created a correlation heat-map of our features to have a visual representation of their correlations with each other. Shown Below.



Upon consultation with the expert, they informed us that every sample in which the counts of, "`ct_grenade_decoygrenade`", "`t_grenade_decoygrenade`", "`t_weapon_r8revolver`", or "`ct_weapon_r8revolver`" are greater than zero, should be eliminated from our dataset as they are outliers and thus should not be considered by our model. These samples accounted for approximately 5% of our total dataset.

Also, upon further consultation with a former professional CS:GO player, we decided to remove the, "`map`" "`t_grenade_decoygrenade`", "`ct_grenade_decoygrenade`", "`ct_weapon_r8revolver`", and "`t_weapon_r8revolver`" features as "`map`" is unlikely to correlate enough with round outcome to justify its inclusion and the rest are now all zero due to the previous step. We also removed the "`t_weapon_mag7`" column because for every sample, its value was zero.

6.3.3 *Feature Engineering*

To increase the effectiveness of several of our models, we chose to edit the data type of a few of our labels:

- Changed "`round_winner`" from either 'CT' or 'T,' to
  - 0 if 'CT'
  - 1 if 'T'
- Similarly, changed "`bomb_planted`" from either True or False, to
  - 0 if False
  - 1 if True

We also inserted a column for the *TrueSkill* metric following the implementation outlined by *Xu Huang et al.* in that we used the individual values for team scores to derive a quantity meant to resemble: "match fairness" and thus give us a quantification of each team's likelihood to win.

6.4 Models to be Implemented

Considering the diversity of models trained in the papers cited, we will follow suite in that we will also be training a wide array of models named below:

- *HistGradientBoost*
- *Random Forest*
- *Logistic Regression*
- *K-nearest Neighbors*
- *Multi-layer Perceptron*

6.5 Selection of Hyperparameters

In order to ensure that we comprehensively check the performance of as many combinations of hyperparameters as possible, we utilize *SciKitLearn*'s *GridSearch*. *GridSearch* allows us to automatically and efficiently train a certain model with many different combinations of hyperparameters. *GridSearch* also returns the model which achieved the highest accuracy for us to immediately evaluate on the test set.

GridSearch also performs normalization automatically as well as performs k-fold cross-validation thus removing the need for a distinct validation set. In our case, we used $k = 5$.

6.5.1 HistGradientBoost

We chose to train *HistGradientBoost* because it is known to be much faster than normal *GradientBoost* when working with a number of samples greater than 10000[4].

The hyperparameters we chose to tune using *GridSearch*, were,

1. Learning Rate: [10^{-3}, 10^{-2}, 10^{-1}, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1]
2. Max Iterations: [100,500,1000,1500]

6.5.2 Random Forest

The hyperparameter we chose to tune for the *Random Forest* using *GridSearch*, was,

1. Number of Estimators: [50, 100, 120, 140, 160, 180, 200, 250]

6.5.3 Logistic Regression

The hyperparameters we chose to tune for *Logistic Regression* using *GridSearch*, was,

1. C (strength of regularization): [10^{-2}, 10^{-1}, 1, 10, 10^{2}]
2. Max Iterations: [900, 1000, 1100, 1200, 1300, 1500]

6.5.4 K-Nearest Neighbors

The hyperparameter we chose to tune for the *K-Nearest Neighbors* using *GridSearch*, was,

1. Number of Neighbors considered: [1, 2, 3, 4, 5, 6, 7, 8, 10, 20]

6.5.5 Multi-layer Perceptron

For training this model, we could not use *GridSearch*, so we re-split our data into distinct training, validation, and test sets. The test set comprises 20% of the whole, the validation, of 20%, and the training of 60%.

We used a network comprising of two hidden layers each containing 100 neurons. The two hidden layers use the ReLu activation function, and the output uses the Sigmoid activation. We had it train for 50 epochs.

7 Results

In this section, we discuss the results corresponding to each of the models we trained. As discussed above, we evaluate these models on their accuracy.

7.1 HistGradientBoost

From *GridSearch*, the optimal hyperparameters were a learning rate of 0.1 and a maximum number of iterations of 1500. Results using said hyperparameters are as follows:

HistGradientBoost

Train	***Test***
81.53%	83.95%

7.2 Random Forest

We found that the optimal value for $n\_estimators$ for our dataset was: $n_{estimators} = 200$. Results using said hyperparameters are as follows:

Random Forest

Train	***Test***
85.66%	86.55%

7.3 Logistic Regression

We found that the optimal hyperparameter values are: $C = 0.01$ and $max\_iter = 1300$. Results using said hyperparameters are as follows:

Logistic Regression

Train	***Test***
75.22%	75.52%

7.4 K-Nearest Neighbors

We found that the optimal hyperparameter value is: $k = 1$. Results using said hyperparameters are as follows:

K-Nearest Neighbors

Train	***Test***
78.54%	80.66%

7.5 Multi-layer Perceptron

Using the optimal network shape of two hidden layers each consisting of 100 neurons, we obtained the results as follows:

Multi-layer Perceptron

Validation	***Test***
75.06%	75.19%

7.6 Table of Results

Below is a table containing our results as well as the relevant results from Xu Huang et al. As you can see, we largely outperformed the results by (Xu Huang et al.) and by extension, clearly outperformed our strawman baseline of 50%.

Author:	***Model***	***Train***	***Valid.***	***Test***
Ours	*Hist-Gradient-Boost*	81.53%	N/A	83.95%
	Random Forest	85.66%	N/A	86.55%
	Logistic Regression	75.22%	N/A	75.52%
	K nearest neighbors	78.54%	N/A	80.66%
	Multi-layer perceptron (NN)	N/A	75.06%	75.19%
Xu Huang et al.	*Decision Tree*	75.04%	N/A	74.34%
	Gradient Boosted Decision Tree	75.78%	N/A	74.97%
	XGBoost	83.90%	N/A	79.19%
	Logistic Regression	74.85%	N/A	74.18%
	Neural Network(NN)	83.41%	N/A	78.14%

8 Discussion

Below we will discuss the steps we took which we believe improve the performance of our model.

We are unsure if (Xu Huang et al.) used *GridSearch* to tune hyperparameters but we believe that this helped us greatly in increasing our performance.

We also had the opportunity to consult an expert on CS:GO to help us with feature engineering and selection, which we believe helped by improving the quality of our data and, by extension, our models.

We also tried to implement a Convolutional Neural Network architecture, however, due to time and resource constraints, we were unsuccessful in our implementation.

Also, we attempted to implement a fairly new model developed specifically to be an interpretable neural network architecture built for tabular datasets called, TabNet[5]. However, due to the limited resources available for this model, we were unsuccessful in implementing it.

9 Conclusion

I gained a great deal of skills from this project that I couldn't through assignments, such as, how to write an academic paper, how to use API's much more efficiently, as well as gaining much more familiarity with these libraries like Keras, TensorFlow, and SciKitLearn. All of which will help me tremendously in the future as I go on to graduate school for data science.

As for the future of this project, once more resources for TabNet become more accessible, I hope to implement it for my project to see how it performs.

10 References

[1] Xu Huang, Weison, et al. "Predicting Round Result in Counter-Strike: Global Offensive Using Machine Learning." *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 2022, https://doi.org/10.1109/icsp54964.2022.9778597.

[2] Švec, Ondřej. "Predicting Counter-Strike Game Outcomes with Machine Learning." (2022).

[3] Björklund, Arvid, et al. "Predicting the outcome of CS: GO games using machine learning." (2018).

[4] sklearn.ensemble.HistGradientBoostingClassifier. (n.d.). Scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

[5] S. S. Paliwal, V. D, R. Rahul, M. Sharma and L. Vig, "TableNet: Deep Learning Model for End-to-end Table Detection and Tabular Data Extraction from Scanned Document Images," 2019 International Conference on Document Analysis and Recognition (ICDAR), Sydney, NSW, Australia, 2019, pp. 128-133, doi: 10.1109/ICDAR.2019.00029.