Joshua Wilson

Instructor: Alexander Lozinski

Class: AOS111

Assignment: Final Project

Predicting Counter-Strike: Global Offensive CT Round Wins with Machine Learning

Introduction

Videogames have long become a staple in global culture. Some studies show that over 50% of American households have some sort of video-game console, a statistic that doesn't account for computers and mobile devices, which are extremely popular devices to play videogames on. In contemporary society, electronic sports (E-sports) have become as popular and lucrative as many traditional sports, with fans purchasing tickets to watch tournaments and ordering merchandise to support their favorite E-sports teams. Furthermore, E-sports gambling, where participants can gamble on teams they think might win, has also seen an explosive surge in popularity even amongst people who do not play videogames. Counter-Strike: Global Offensive (CS:GO) is one game in particular that is known for its popular E-sports scene, expansive market and gambling scene, and – most importantly – its timeless gameplay.

CS:GO is a tactical 5v5 shooter played in a best-of-30 format, where teams alternate between Terrorist and Counter-Terrorist sides after 15 rounds. Each round lasts 1 minute and 55 seconds, and the first team to reach 16 rounds wins. Matches are played on one of seven maps. Terrorists win rounds by eliminating the opposing team or successfully detonating the bomb, while Counter-Terrorists win by eliminating the attackers or defusing the bomb once planted. It is part of one of the longest-lived video game series that has retained its popularity and player-count, with the first game (CS 1.0) being released in 2000 and the most recent release, CS2, averaging a daily player-count of over 1 million [1, 3]. Personally, CS (the colloquial term for any game in the series) is a game I love and have played for over a decade, of which I've managed to rack up a total of 2,224.5 hours of in-game experience. It was my extensive history with CS that motivated me to find any CS-related dataset that I may be able to use for this final project. Luckily, I found a wonderful one that lists many in-game stats during a given snapshot of a CS round. In this project, I will be using this dataset and three different machine learning models to predict round wins for the CT side.

Dataset Analysis

The dataset I have been using is called "CS:GO Round Winner Classification" and was found on the Kaggle website [2]. (https://www.kaggle.com/datasets/christianlillelund/csgo-round-winner-classification?resource=download)

According to the uploader of the dataset, it "was originally published by Skybox as part of their CS:GO AI Challenge, running from Spring to Fall 2020" [2]. It consists of roughly 700 demos from high level tournament games that were played in 2019 and 2020. All warmup rounds and restarts have been filtered, and from the remaining rounds, snapshots were taken every 20 seconds until the round winner was determined. The total number of snapshots in this dataset is 122,411. The dataset contains 97 features, including the map, time left in the round, CT and T scores, CT and T players alive, and the total money, armor values, weapons, and utility on each team. The data is arranged in a list with shape (122410, 97). Most of the features are related to the weapon and utility totals on each team, as CS:GO has 34 distinct weapons and 5 different utilities. With each weapon and utility being used to create two features, one for each of the two teams, this accounts for 78 of the 97 total features. The first 5 snapshots are shown in Figure 1.

	time_left	ct_score	t_score	map	bomb_planted	ct_health	t_health	ct_armor	t_armor	ct_money	...
0	175.00	0.0	0.0	de_dust2	False	500.0	500.0	0.0	0.0	4000.0	...
1	156.03	0.0	0.0	de_dust2	False	500.0	500.0	400.0	300.0	600.0	...
2	96.03	0.0	0.0	de_dust2	False	391.0	400.0	294.0	200.0	750.0	...
3	76.03	0.0	0.0	de_dust2	False	391.0	400.0	294.0	200.0	750.0	...
4	174.97	1.0	0.0	de_dust2	False	500.0	500.0	192.0	0.0	18350.0	...

Figure 1. First 5 rows of dataset with first 10 features.

As can be seen, this dataset contains both numerical (e.g. time_left and ct_score) and categorical (e.g. map and bomb_planted) data. I was first curious about the distribution of round wins between the CT and T sides. As can be seen in Figure 2, the CT side has 17,573 round wins, which accounts for ~51.44% of the dataset, and 16,590 round losses (aka T side round wins). This was nice to see, as that meant my machine learning models likely wouldn't significantly favor one prediction over the other as would be the case if, for example, CT round wins made up 99% of the dataset. I was intrigued to find such an even distribution of round wins, as many CS maps are thought to favor the CT side. I'd imagine the developers of CS have used this metric to better balance maps.

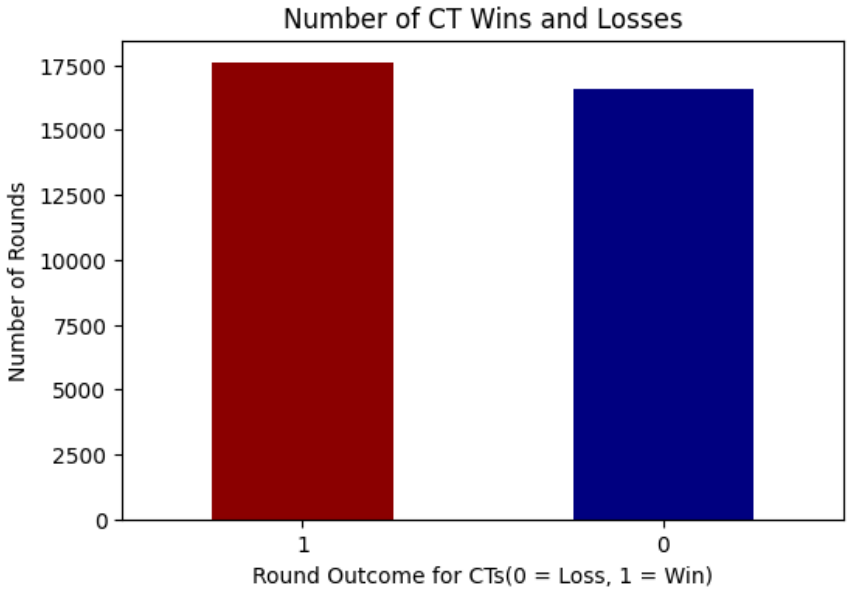


Figure 2. Amount of CT and T round wins, as well as CT round win percentage.

I was then interested in the distribution of time left in the rounds, as well as the distribution of the total money the CT side had during round wins vs. losses. I plotted this information on a bar graph and box-plot, respectively, as shown in Figures 3 and 4.

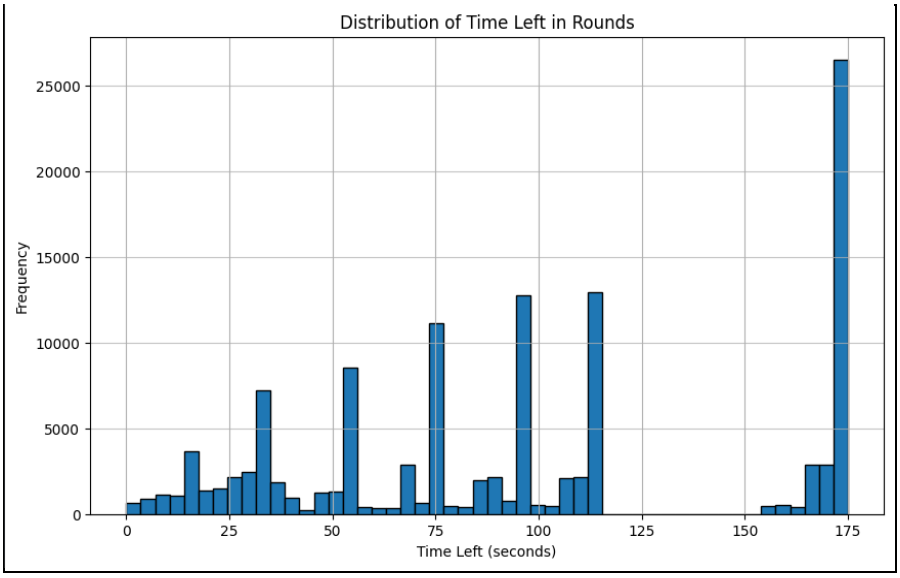


Figure 3. Distribution of time left in rounds.

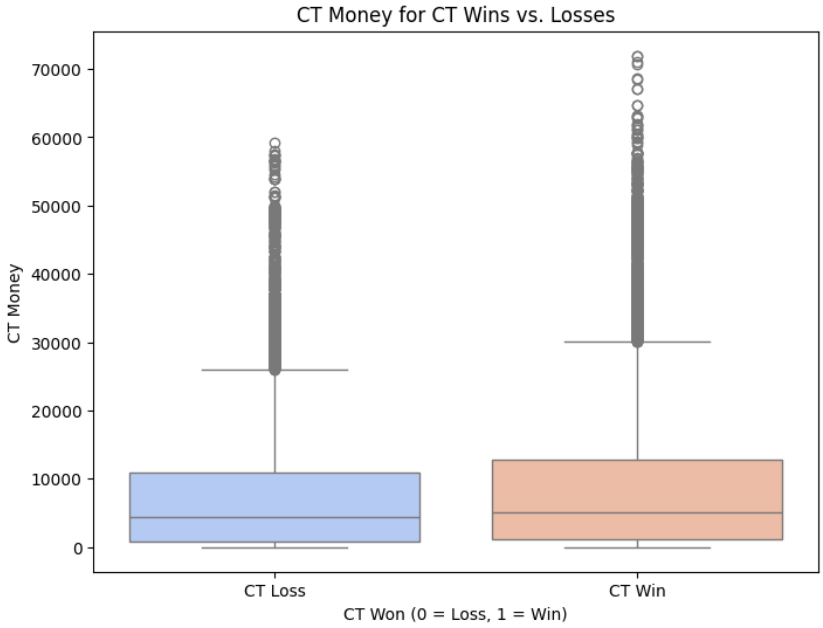


Figure 4. CT Money for CT wins vs. losses.

Figure 4 shows the expected trend that rounds that resulted in CT wins consist of snapshots of which the CT side had more total money on average. This is no surprise; better equipment costs more money. Figure 3, on the other hand, is quite interesting. CS rounds last a maximum of 151 seconds: 115 seconds is the total round length given no bomb is planted, and if a bomb is planted before the round ends (4 seconds to plant the bomb), another 40 seconds is added to the total possible round time. This calculates to 115 – 4 + 40 = 151 seconds, which is the longest possible time a round can last. However, as seen in Figure 3, many snapshots were taken during rounds with over 155 seconds remaining, with some having as long as 175 seconds remaining. My only explanation is that 'warm-up' time and time-outs were possibly included. Warm-up time refers to the 15 seconds of time allotted before each round start for players to buy weapons and equipment. Time-outs add 60 seconds to the round time and can be called by players from either team. Each team can call a time-out during the same round. With these new numbers in mind, 115 – 4 + 40 + 15 + 2(60) = 286 seconds for the longest possible round time; we do not see any snapshots taken with 286 seconds remaining in the round.

With the data exploration completed, I moved on to cleaning the data. First, I decided to only keep snapshots of rounds with >=60 seconds and <=100 seconds left. This way, my focus can be on snapshots taken during the early to early-middle of the round, as snapshots taken with less than 60 seconds of remaining round time are likely already very one-sided due to potential CT/T player deaths or health loss, and therefore not much of a 'prediction.' Additionally, I wanted to filter out many of the less impactful features. Based on my personal experience playing CS, I ultimately decided to keep 14 features, as shown in Figure 5:

```
selected_features = [
  'time_left',
  'ct_health',
  't_health',
  'ct_armor',
  't_armor',
  'ct_money',
  't_money',
  'ct_players_alive',
  't_players_alive',
  'ct_weapon_ak47',
  't_weapon_ak47',
  'ct_weapon_m4a4',
  'ct_weapon_m4a1s',
  'round_winner']
```

Figure 5. 14 selected features for future modeling.

time_left, as the name implies, is the amount of time left in a round at a given snapshot. CS rounds are very fast-paced, and if you are on the T side, you must use what little time you have to strategize a plan of attack and act on it. The more time remaining, the more of a disadvantage the CT side has. I also decided to keep more obvious features, such as the total health, total money, total armor values, and player count of the T and CT sides. Additionally, I removed all the weapon-related features except for the ak47, m4a1s, and m4a4, which are the three primary weapons in CS. The ak47 is a superior weapon to the m4a4 and m4a1s and is only purchasable by the T side. However, CT players can pick up the ak47 from defeated T opponents. I did not include features for m4a4s and m4a1s' on the T side, as very few T players ever had these weapons, as shown in Figure 6:

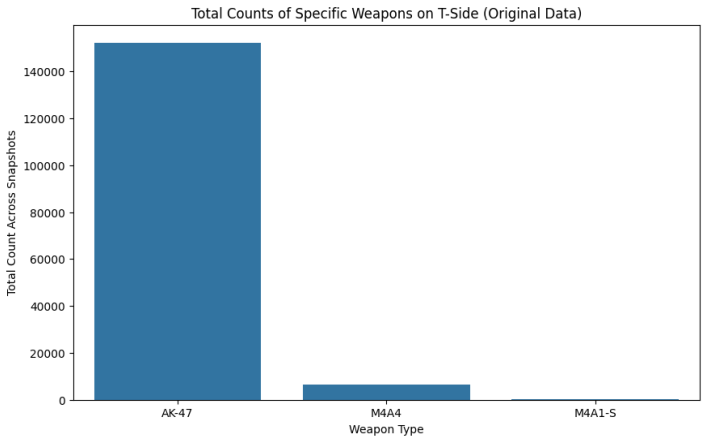


Figure 6. Total counts of specific weapons on T-side.

With the data cleaned and filtered, I then wanted to make sure all my data was numerical to make the machine learning process easier. 'round_winner' is still categorical, so I changed the data to be a binary 1 (CT win) or 0 (CT loss) using .astype(int):

```
#change "round_winner" feature to "ct_won" and make numerical
#1 = CT win, 0 = CT loss
data['ct_won'] = (data['round_winner'] == 'CT').astype(int)
data = data.drop('round_winner', axis = 1)
```

Last was to define my features as x and my predictions (CT round win/loss) as y and to split my data into training and testing data. I used an 80-20 split, respectively.

Machine Learning Models, Comparisons, and Discussion

I decided to use three categorical machine learning models that we learned about in class: logistic regression, SVM, and a random forest classifier. I started with the logistic regression model, which was the model I first thought of when I found my dataset.

Logistic regression models are designed for binary classification problems, which is exactly what my project can be classified as. My goal is to predict whether the CT side will win or lose a round given a snapshot of said round, meaning the model must predict either a 1 for a CT win or 0 for a CT loss. Per my professor, logistic regression is also best used with a smaller number of important features, which is one of the reasons I decided to filter my data to only 13 features (including round winner). As stated above, I used an 80-20 split for my training and testing data. Figure 7 shows the confusion matrix and the ROC curve with the AUC score for my logistic regression model:

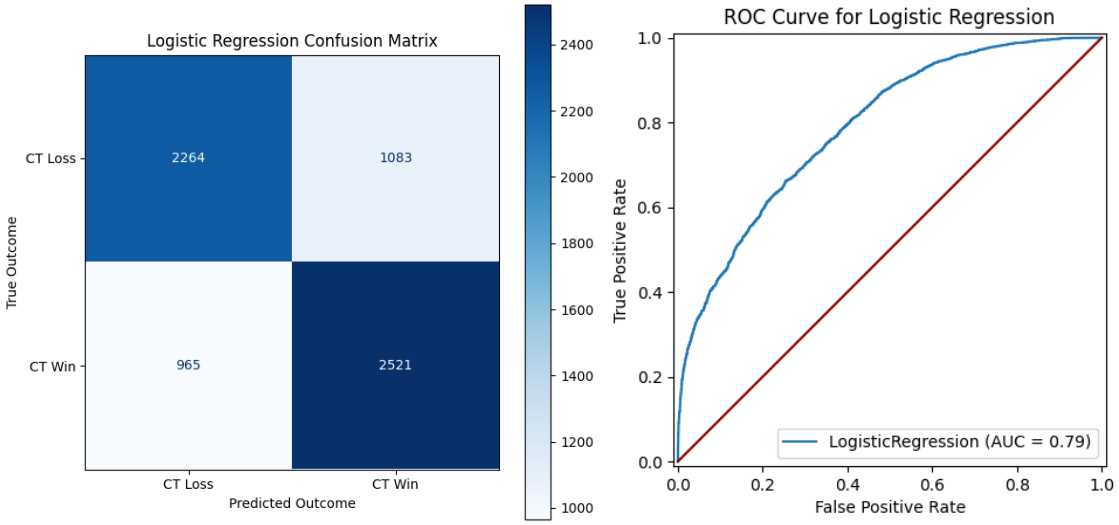


Figure 7. Confusion matrix (left) and ROC curve with AUC score (right) for logistic regression model.

The True Negatives (TN) of my confusion matrix (top-left quadrant) indicates that my model correctly predicted 2264 CT round losses when CTs actually lost the round. The True Positives (TP) of the confusion matrix (bottom-right quadrant) indicate my model correctly predicted 2521 CT round wins when CTs actually won the round. Conversely, the False Positives (FP) (top-right quadrant) indicates my model incorrectly predicted a CT round win 1083 times when CTs actually lost, and the False Negatives (FN) (bottom-left quadrant) indicates my model incorrectly predicted 965 CT round losses when CTs actually won. There was a similar number of False Positives and False Negatives, with a ratio of 1.12:1. The logistic regression model accurately predicted ~72% of the CT round wins and a lower ~67.6% of the CT round losses. The overall accuracy of my logistic regression model was ~70%, indicating my model was significantly more accurate than simple chance. This can be better visualized by looking at the

ROC curve. As seen in Figure 7, the AUC score was 0.79. The accuracy score and three error metrics all support the conclusion that my linear regression model was an accurate predictor of CT round wins.

The second machine learning model I used was the Support Vector Machine Model (SVM). I chose SVM as a second classification algorithm after being advised to do so by my professor. This model should help provide a more sophisticated approach to solving my binary classification problem which, like logistic regression models, SVM excel at, and will allow me to make a direct comparison with my logistic regression model. While logistic regression finds a linear decision boundary (such as with the sigmoid function we learned about in class), SVMs are particularly powerful for complex, high-dimensionality datasets such as the one I'm using. I have filtered the features of my model quite significantly, which may affect the accuracy of the SVM. The confusion matrix and ROC curve with AUC score are shown in Figure 8:

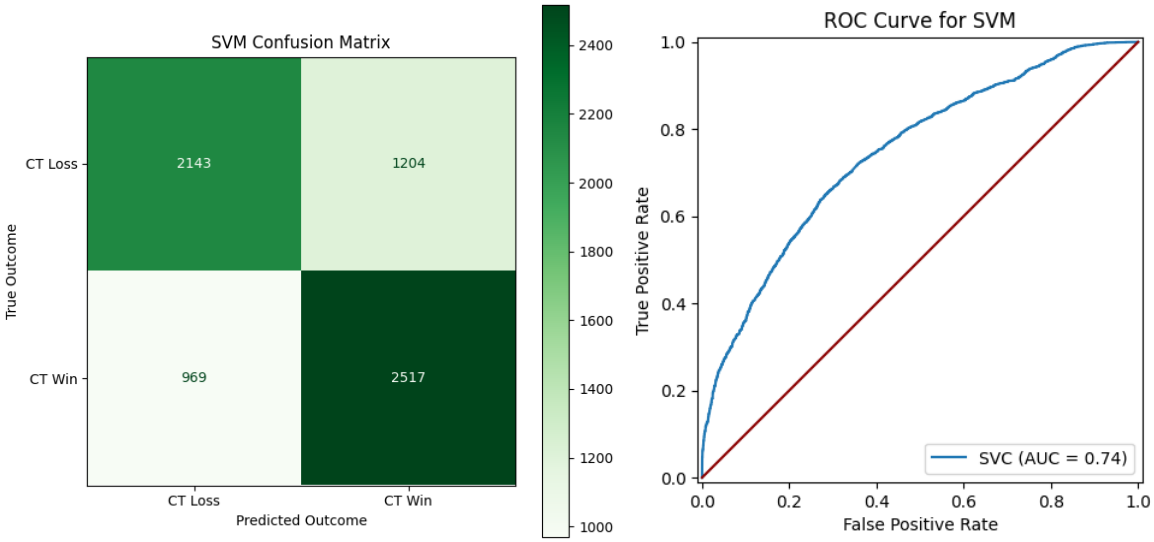


Figure 8. Confusion matrix (left) and ROC curve with AUC score (right) for SVM.

Similar to the logistic regression model, the TP, TN, FP, and FN can be seen in my confusion matrix of the SVM. Overall, the SVM seems to have more FP predictions, with 1204 FPs compared to the linear regresion model's 1083. Compared to the logistic regression model's ~72% CT win accuracy, the SVM model had the same prediction accuracy of ~72%. However, this model did have a lower CT loss prediction accuracy of ~64% compared to the logistic regression model's ~67.6% accuracy. The SVM model seems to be an overall worse predictor of CT round losses compared to the logistic regression model. The SVM also has a slightly lower overall accuracy score of 68.2% and a lower AUC score of 0.74 compared to that of the logistic regression's score: 70% and 0.79, respectively. I did learn that SVM were more sensitive to the scale of my features given the fact they attempt to separate the dataset via a hyperplane, so I'm wondering if this is a possible cause of the difference in accuracy of my models. My features were not scaled and have a wide range of values. For example, the weapon features have a range of 0-5, the health and armor features have a range of 0-500, and the money features have a range

of 0-80,000. The difference in the scale of these features may have resulted in the slight decrease in accuracy seen in my SVM in comparison to my logistic regression model.

The third and final model I decided to create was the random forest classifier. This model takes an ensemle approach by using many decision trees. This was another model we used in class and was the one that I was able to get the highest accuracy score with. It offers a nice benchmark against the simpler logistic regression model and the more complex, but potentially sensitive, SVM. Random forests operate by literally constructing a 'forest' of multiple decision trees during their training. Each tree is built in a randomly allocated subset of training data with a consideration for a random subset of features for each split. The final prediction is then determined by aggregating the predictions of each individual tree in a process akin to a majority vote. The downside of this approach with my dataset is the difficulty with visualizing the model, as even a single decision tree took a very long time to graph and is not readable at high max depths. Figure 9 shows the first decision tree of my random forest model with a max depth of 10:

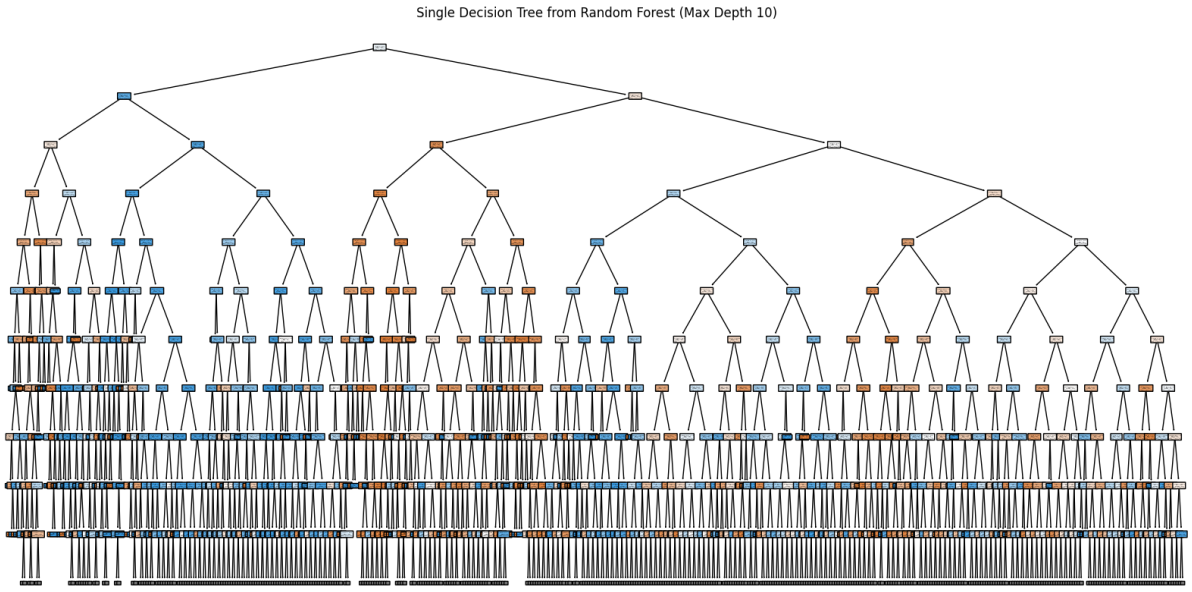


Figure 9. Single decision tree from random forest model at max depth of 10.

As stated previously, the above decision tree is simply unreadable at this depth. I did the same using a max depth of 3, which was the highest max depth I could use without losing readability, as shown in Figure 10. The blues represent a majority of CT round wins and the oranges a majority of CT round losses at the corresponding splits.

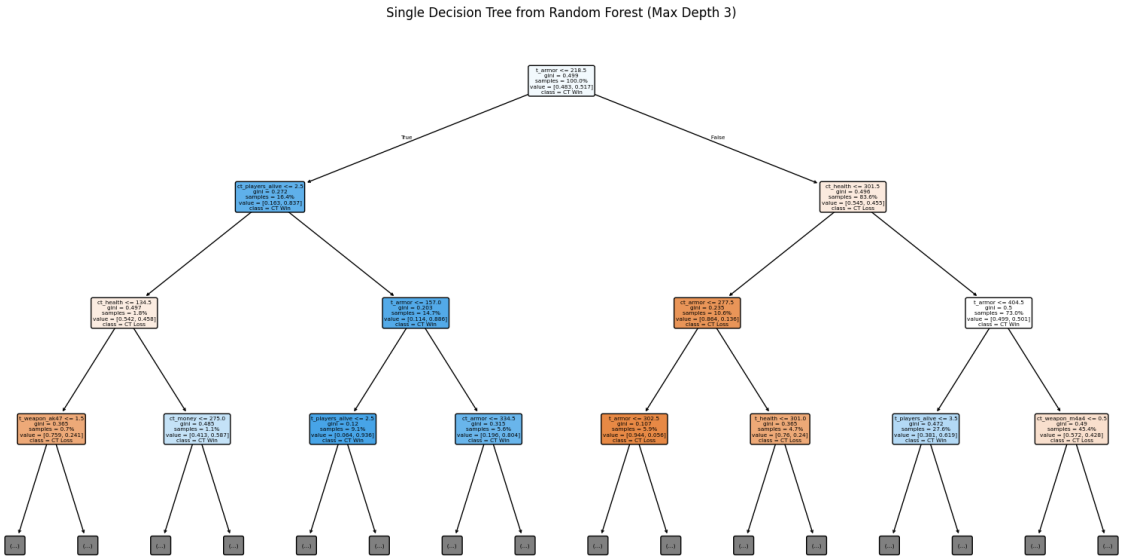


Figure 10. Single decision tree from random forest model at max depth of 3.

The confusion matrix and ROC curve with AUC score for my random forest model are shown in Figure 11.

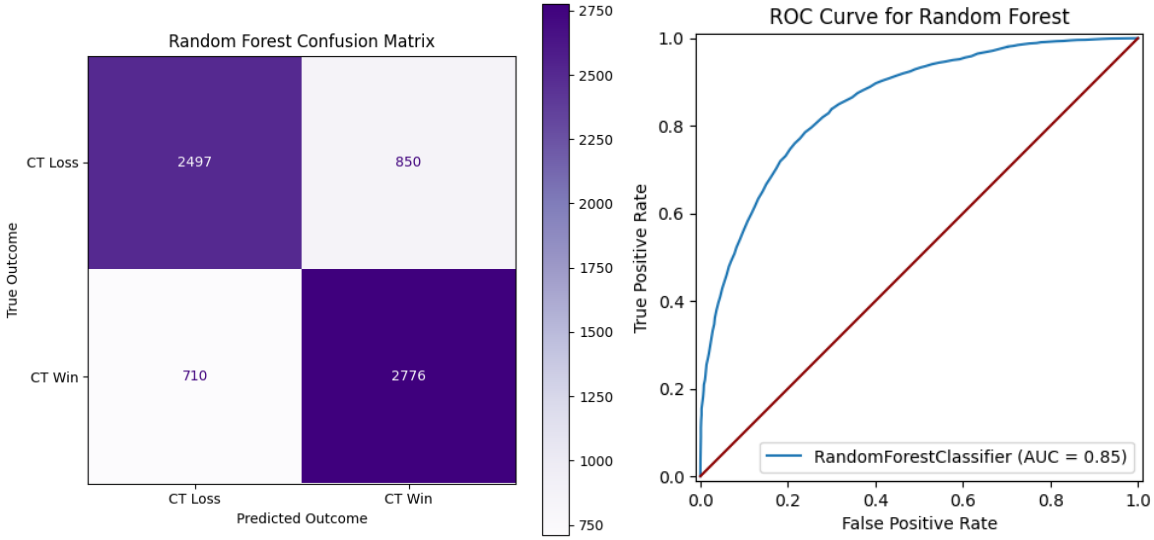


Figure 11. Confusion matrix (left) and ROC curve with AUC score (right) of random forest model.

The random forest model was significantly more accurate than both the logistic regression model and SVM. The overall accuracy of this model was ~77%, and as seen in the confusion matrix, the random forest model predicted more TP and TN and less FP and FN compared to the previous two models. This model predicted CT round wins with ~79.6% accuracy and CT round losses with ~74.6% accuracy, the highest of the three models by quite a

large margin. The AUC score was also higher, being 0.85. A table of the accuracy and AUC scores is shown in Figure 12.

Model	Accuracy	AUC Score
Logistic Regression	0.7003	0.7915
SVM	0.6820	0.7449
Random Forest	0.7717	0.8529

Figure 12. Model accuracies and AUC scores.

Additionally, Figure 13 shows the ROC curves of each model superposed on one graph, with a diagonal line plotted for reference:

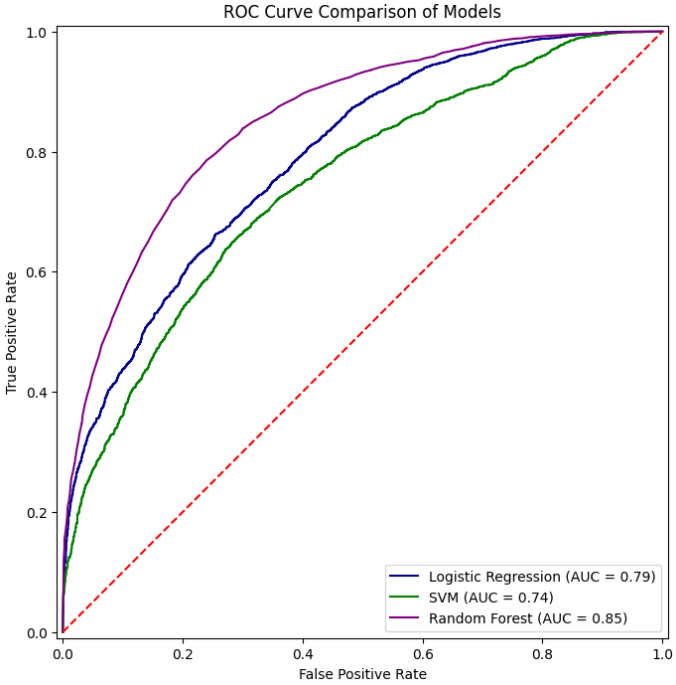
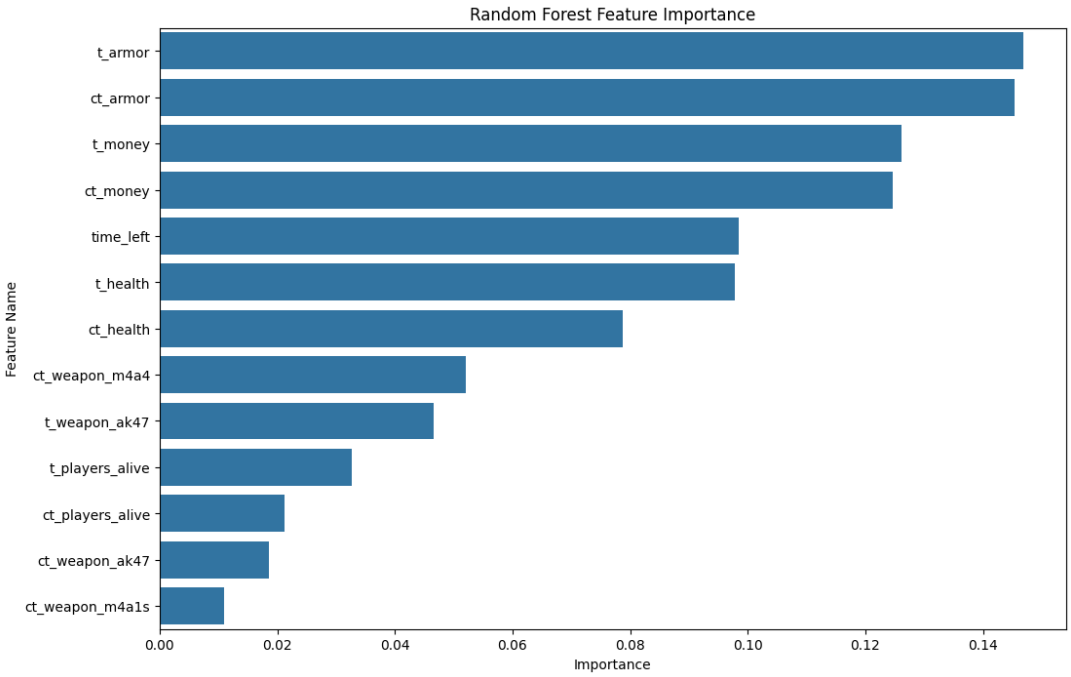


Figure 13. ROC curves and AUC scores of each model.

According to the accuracy score and all three error metrics used (confusion matrix, ROC curve, and AUC score), the random forest classifier was the overall best predictor of CT round wins and losses compared to the logistic regression model and SVM. I'd imagine this is the case for a few reasons that I learned in class: first, the random forest classifier is an ensemble method and contains many, many individual decision trees, which helps to minimize any overfitting and noise that may be present and negatively affecting accuracy in the other two models. Second, the random forest classifier are great at capturing complex, non-linear relationships, which I'd argue is the best type of model for this dataset; CS is played by humans, and upsets in the game occur all the time. Finally, third, random forests are also great at determining feature importance and can therefore effectively utilize more important features in its predictions. With this third reason

in mind, I was interested in seeing the importance of the features. Figure 14 shows this, arranged in decending order:




```
#sort my features (very nice)
feature_importance = feature_importance.sort_values(by = 'Importance', ascending = False)
```

Figure 14. Feature importance determined by random forest classifier (top) and code used to sort features in descending order (bottom).

I was initially surprised to see T and CT armor values taking the top two spots by a seemingly signifiфcant margin. Of the 13 shown features, I would have ranked them in the five least important features. I was also surprsied to see CT and T players alive near the bottom half of the graph. However, after some more thought, I determined that this ranking actually makes a lot of sense. In CS, there is a very popular strategy of 'throwing' rounds. Often, experienced players will forfeit buying mediocre weapons and utility in one round to guarantee they can afford to buy optimal equipment in the round afterwards; rather than have two rounds with mediocre equipment, many players prefer one round of bad equipment in exchange for one round of very good equipment. The reasoning behind it has to do heavily with in-game economy that is outside the scope of this paper, so this fact must be taken for granted. It is also important to note that players will often forfeit better weapons for armor, as armor is always a high priority purchase. Therefore, if one side has significantly less armor, that likely means they also have significantly lower quality weapons and utility (utility was not accounted for in the models), meaning they are more likely to lose a given round. To explain the ranking of CT and T players alive is a much easier venture: because of the time constraints placed when cleaning the data (snapshots with >=60 seconds and <=100 seconds of round time remaining), it is likely that not many players have died in the early and early-middle stages of the round.

Conclusion

To summarize, I used a CS:GO dataset that consisted of snapshots of rounds and many in-game stats of the T and CT sides at a given snapshot. Using this data, I trained three machine learning models that excelled at binary classification with the goal of predicting CT round wins. The models I used were logistic regression, SVM, and a random forest classifier. After taking the accuracy scores, confusion matrices, ROC curves, and AUC scores into account, I concluded that the random forest classifier was significantly more accurate at predicting CT round wins, with logistic regression and SVM falling in second and third place, respectively. I determined that the most important feature of the 13 features I selected was the T and CT armor values, which I was able to personally reason and make sense of given my extensive CS in-game hours (not to brag). Using machine learning to predict round wins in Counter-Strike can potentially be a great technique to help analyze weak points in specific E-sports teams, such as which equipment they should be prioritizing given specific in-game stats and round conditions. Additionally, it is possible to use similar models when gambling on E-sports, although this is not recommended. There is much room for optimization of the machine learning models used, which can be done by selecting different features (more or less, depending on the learning model) or by making the predictions map-specific rather than all inclusive, as was done in this project.

Author's Note

I know I likely should not be putting an author's note in a scientific paper, but I wanted a space where I could be a bit more informal. I initially found this project to be extremely daunting given my very limited coding skills. However, I quickly found myself having quite a lot of fun exploring and messing with the dataset, especially considering it pertains to a game I love playing to this day. Heck, after I submit this project, I'm going to play some CS2 to relax. I found myself using the homework and ICC assignments extensively during this project, which thankfully saved me several hours of time. Don't get me wrong, the coding still took an insanely long time to do haha (curse my lack of skills in coding), but thankfully importing all the libraries and stuff was super easy and extremely cool. I can't believe I was able to make three machine learning models in so few lines of code. Even harder to believe is how long it took me to write those lines of code (sigh). Anyways, thank you for the great quarter. I absolutely learned a lot, and found this final project to be a ton of fun. Machine learning is cool.

Reference Page

[1] Wiki, C. to C.-S. (n.d.). Strike beta. Counter. https://counterstrike.fandom.com/wiki/Counter-Strike_Beta

[2] Christian Lillelund. (2021). *CS:GO Round Winner Classification*. Retrieved from https://www.kaggle.com/datasets/christianlillelund/csgo-round-winner-classification?resource=download

[3] Counter-strike 2. Steam Charts. (n.d.). https://steamcharts.com/app/730#48h