PROJECT INTRODUCTION TO MACHINE LEARNING :

Using overall team and personal impact to predict a team's success on a Counter-Strike Pro Match

A work by BABOULAT Léandre & FONTAINE Paul

Abstract

In this report, we explain how we used the performance of Counter-Strike teams and the stats of individual players to predict the winner of a given game. We are both students in Computer Science learning the basics of data science, and we chose this subject because we both like to play Counter-Strike and follow competitive games, and we always thought it would be cool if we could predict the winner of a given game. Predicting the winner of something, whether it is Counter-Strike, football or Formula 1, has always been a complex problem, used in the industry to understand what makes teams win, in media to explain what makes a team good, and in the betting industry to adjust the odds of a bet. Some models to predict outcomes in the Counter-Strike scene already exist, but we wanted to try something different and predict the winner of a game in general using our own thought process. The question of which data to feed our model was harder, we tried multiple techniques and used different datasets before finding the one we are presenting here, and we had to think through some original preprocessing techniques specific to our dataset. After implementing models such as Random Forest, LinearSVC, and XGBoost, we obtained reliable results, with our best model being XGBoost reaching around 74% accuracy and a F1-score of about 0.71 on more than 120,000 professional matches. By looking at feature importance, we found that the impact of individual performance was smaller than we expected compared to the consistency of a team over time.

Introduction

The problem we are trying to solve is simple: how to predict the winner of a Counter-Strike game. This may seem like a useless question, but as we are ourselves casual Counter-Strike players, we wanted to know what makes a team be “the best” team, and creating a machine learning model to predict the winner of a match allows us to understand just that.

This is a common question in many fields. In sports, from badminton to football, creating a model to understand the opponent and to understand why the current world champion is better than the others is the norm for big teams. In our case, in Counter-Strike, we look at the history of both teams playing and, using the average performance of each team, we create a clean dataset that a machine learning model can use to predict the winner of the game based on multiple variables. One of the key issues is finding a way to correctly sum up variables so that we do not give too much useless information that would just act as noise but still give enough information so the model can detect relationships and learn.

Counter-Strike is a first-person shooter video game played online where two teams of five players go head-to-head during what we call a game. We only focus on the professional scene of Counter-Strike, so teams have five players that usually play together, even though sometimes players sign other contracts and change teams. A Counter-Strike game has 30 rounds, so the first team to 16 rounds wins the game. In each round players fight against each other and there is only one winner per round, they can buy weapons depending on the in-game economy, but these details are not necessary to understand what we decided to do.

Important terms are “kill”, a kill is when you eliminate another player, “death” it’s the opposite it’s when you get eliminated by the other player, “assist” when you deal some damage to an enemy and someone else kills him, this is useful to show that a player is useful even if he isn’t the one doing the kill, “headshot” when you eliminate a player by shooting it’s head, we usually talk about it in percentage of kills by headshot over percentage of overall kills, since the head is a really small part of the body it’s a show of skill to hit it consistently. As said previously “rounds” are a smaller unit in a game, each teams wins some and the first to 16 wins the game, this can used to see how close a game was, for example a game ending on 16-13 was really close but if it ends on 16-2 the winning team dominated.

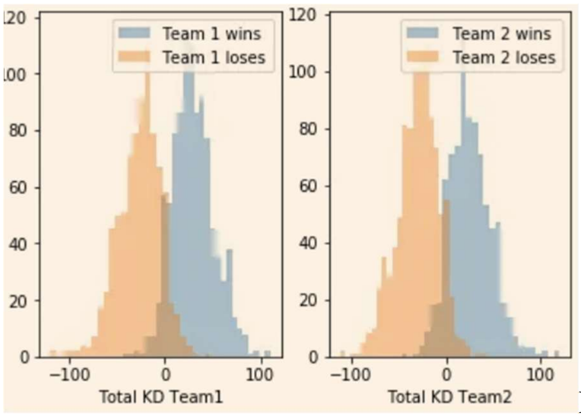
To gather all the needed data, we first went on Kaggle and looked for professional history of Counter-Strike games and found a dataset(1), but we could not find all the variables we were looking for, and the dataset stopped at 2020. We then tried HLTV.org, which has close to all information about each player(2), but they had no open API, so we coded a web scraper that goes on each player page and gathers the data we need. It took more than 10 hours to run because we did not want to overload the website, but even after cleaning this dataset and trying various models, the results were not good enough for us. After more digging we finally stumbled upon a dataset website redirecting to a GitHub(3) that had exactly what we were looking for: games from 2012 to 2023, more than 120 000 games and all the parameters we wanted. We started over from this dataset, cleaned it by removing useless variables, filling in missing data and merging tables, and we also changed how information is given so that we never use statistics from a match to predict that same match.

In the following section we will start by looking at works made by other people on this subject, we are going to compare it to different fields where a parallel can be drawn. We will continue by exploring the different methods and metrics we chose to evaluate the models. Then we will talk about the data preparation, how we did our pre-processing, and more. After that we will explain the model's implementation, how we built the different models, and how we tuned them. We will then look at the model's results and interpret them, see if they are statistically acceptable, and select the best models for our scenarios. After that we will look at selected model's result and try to explain what happened, what was expected, what was surprising, see if we can compare it to related works, and will summarize models with graphs. We will end with a conclusion, summarizing what we did and looking back at what we did right or wrong.

2. Related Works

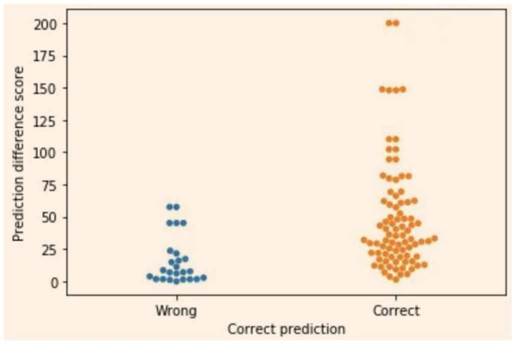
2.1 Work related to Counter Strike

Predicting the winner of a given games has been source of many Machine Learning models, however, the prediction of a games of Counter Strike has been less popular, we did find an online post talking about this problem(4), but he used a totally different approach, instead of predicting the winner directly, he predicted the Kill-Death ratio (also called K/D) with a regression model. His idea is that in most matches the winner is the team with the highest K/D, meaning that if you kill people more often than you die, your team is more likely to win.



From [(4)]

So rather than predicting the winner, he decided to predict the K/D, and then for each match he sums the predicted K/D for all players of each team and checks which team has the highest predicted K/D and compares it to the actual winner. On his test dataset he reports 83 correct predictions for 25 wrong predictions, which is an accuracy of 83 / (83 + 25) = 0.7685.



We will keep this accuracy in mind when looking at our model's performance.

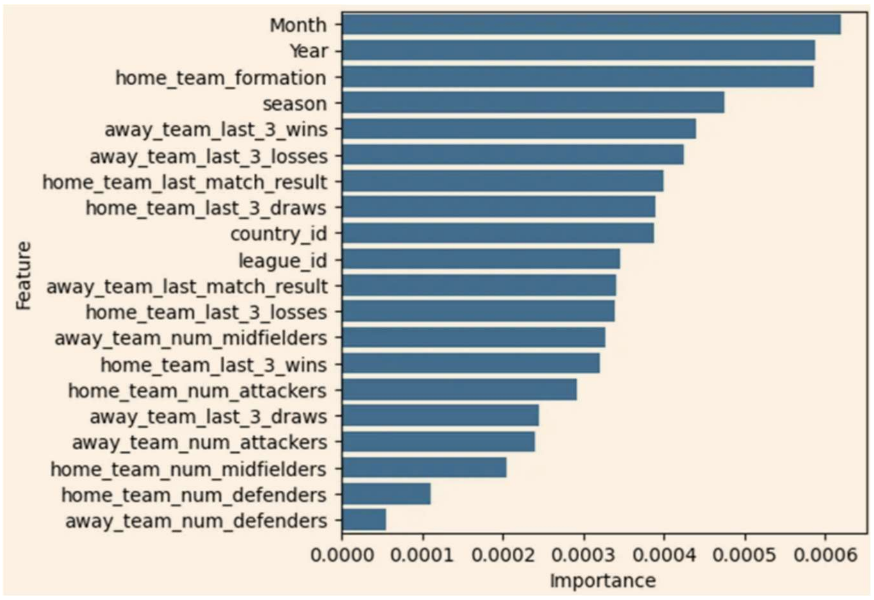
2.2 Advanced work using unsupervised Learning.

We found another study about predicting the outcome of CS:GO games using Machine Learning[(5)], the techniques they used is a way more advanced, after using some web scraping to gather data from a well know site that holds Counter Strike replay [(6)] , they extracted multiple feature describing the play style, then they put players into cluster with the k-means method, finally they use the clustered player and feed it to a neural network, and they compare the result of the neural network with a benchmark and at the end the neural network outperforms the benchmark.

Additional related work focuses on predicting the winner of a round inside a game by looking at statistics like the economy etc. and even though it's the same game this differs fundamentally from our work because their model can be used at the start of any random round to predict the outcome of it while our model needs to have some historic of both team playing to try and predict the outcome of the game.

2.3 Sports outcome prediction

Our problem can be simplified as a match outcome prediction problem, which has been extensively studied in traditional sports and gives a good foundation for esports applications. Football prediction models(8)(9)(10)(11), for example, use similar statistical approaches, combining team form, historical performance, and player metrics to predict outcomes.



These models typically achieve accuracy rates between 55-65% depending on the league or the sports,

They sometimes require more complex models like neural networks because simpler ones often have poor accuracy.

One study, for example, reports: “I have performed Logistic Regression, Naive Bayes and Support Vector Machine algorithms on the dataset with SVM giving the highest accuracy of 61.29%.”(10) The probabilistic nature of sports outcomes, creates a ceiling on prediction accuracy that also applies to CS:GO.

Reference	Problem domain	Method used	Accuracy
Medium/Analytics Vidhya(4)	CS:GO Match Prediction	XGBoost, Regression	0.7685
Chalmers University(5)	CS:GO Match Prediction	K-means, Neural networks	0.65

Football analysis(8)(9)(10)(11)	Sports Match prediction	ML Methods	0.55 to 0.65

3.Methods

3.1 Data Source and Overview

Our dataset has been made from data collected from a publicly available source online, containing game data for all professional games from 2012 to 2023. It includes over 120 000 individual game records, per-player metrics such as kills, deaths, assists, ADR, headshot percentage and rating systems, and. These features are later combined into cleaner team-average and historical features for the models.

The raw dataset contains both categorical features (team names, map names) and numeric features (player statistics), with varying degrees of missing data depending on when the matches happened, because old matches are more likely to lack some data.

3.2 The Learning problem

We will turn this problem into a binary classification problem (0 or 1):

1 means Team 1 won the match, 0 means Team 2 won the match.

This allows us to train models on the probability of Team 1 winning, and from that we can decide which team is predicted to win each game.

3.3 Machine Learning method used

3.3.1 Random Forest Classifier

Random Forest is a machine learning algorithm that constructs multiple decision trees and outputs the class determined by majority voting. We chose to use it because:

- Captures non-linear relationships and feature interactions naturally
- Provides feature importance rankings through mean decrease in impurity

In our models, we used 100 estimators with maximum depth of 10 to prevent overfitting on the training set while maintaining sufficient model complexity so the model will still learn but not memorize the dataset.

3.3.2 Linear Support Vector Classification (LinearSVC)

The LinearSVC can be used for fast binary classification, finding the optimal hyperplane that maximizes the margin between classes. We chose to use it because:

- It's really fast even with many dimensions
- It's memory efficient
- Less prone to overfitting compared to a tree-based method

In our configuration we will use dual=False for efficiency since it's recommended when n_samples > n_features, max_iter=1000, and loss tolerance of 0.127 after tuning for convergence.

3.3.3 XGBoost Classifier

XGBoost is a gradient boosting classifier that sequentially constructs decision trees, with each new tree trained to correct errors made by previous trees. We chose to use it because:

- Really efficient even with large dataset
- Built-in regularization to prevent overfitting
- Handles feature interactions and non-linearity effectively

In our models we decided to use 90 estimators with learning rate of 0.14 ant the binary:logistic objective, since we are doing binary classification, and we found this setup gave the best balance between accuracy and training time.

3.4 Performance metrics

We employed multiple evaluation metrics to comprehensively assess model performance:

- Accuracy:
  - (TP + TN) / (TP + TN + FP + FN)
  - The correct prediction over all predictions
- F1-Score :
  - 2 * (Precision * Recall) / (Precision + Recall)
  - Harmonic mean of precision and recall
  - Precision : TP / (TP + FP)
  - Recall : TP / (TP + FN)
- Classification Report :
  - Detailed per-class metrics including precision, recall, and F1-score

We chose these metrics because they measure the overall performance of the models in an intuitive way. Accuracy tells us how many predictions are correct overall, and F1-score is important because we want to balance low false positives and low false negatives.

The classification report helps us see if the model similar performance on both classes or if it is biased toward one team.

4. Dataset preparation and Exploratory data analysis

4.1 Data collection and initial assessment

The raw dataset was obtained from a public esports database containing professional CS:GO match records. Initial exploratory analysis revealed:

- Total Records : 127 275 matches
- Time Period : 2012-2023 (11 years of competitive play)
- Feature : 155 different data points
- Target Variable Distribution : Approximately 50/50 split between Team 1 and Team 2 wins (Team 1: 63,196 wins, Team 2: 63,676 wins) - balanced dataset, no class rebalancing required
- Missing values : Mostly from older matches

4.2 Data Cleaning and Preprocessing

4.2.1 Handling missing values

For missing kills, deaths, assists and similar stats, we imputed values from the 25th percentile of the distribution. In many cases missing values occur for players on teams that have not played many games, so giving them slightly under-average statistics reflects the team's true level better than using the global mean from all top professional teams.

We used the same logic for team-specific statistics.

4.2.2 Duplicate removal

The dataset initially contained duplicate entries (same match recorded multiple times). These were removed by:

- Creating a unique match identifier (game_link)
- Retaining the first occurrence of each match

- Resulting in 126,872 unique match records after deduplication

4.2.3 Feature extraction and transformation

We extracted information from the raw dataset, the information we extracted include:

- Round Level information
  - The number of rounds each team (Team1 and Team2) won during the game was extracted from extracted from string
  - Calculated the team with highest round win to determine game winner noted as 'team1_won', this feature =1 if team1 won the game and =0 if team2 won the game.
- Player Statistics Parsing:
  - KHS information parsed from "23 (10)" format meaning 23 kills including 10 kills by headshot,
  - For each players statistic we remove "%", "(", ")"
  - We extract Kills, Death, KAST (Similar to a rating feature, meaning Kill-Assist-Survive-Trade %), Average Damage Per Round, K/D differential, First Kill difference for each of 5 players per team.
- Map encoding idea:
  - We have 15 unique maps, we could do some one-hot encoding, which treats each map as independent, we implemented a team-specific map win rate tracking.
  - Each team's historical win rate on each individual map is calculated and used as a feature.
  - This captures the critical insight that teams perform differently on different maps, with some teams showing significant map expertise.

4.3 Feature Engineering

4.3.1 Historical Features

What is, in our opinion, the critical innovation in our approach was constructing historical features representing team form at prediction time. Since for each match we can't give information about the current match occurring, we calculated using matches that were chronologically prior to the prediction match,

At first, we considered all match prior to the prediction match but some teams are almost a decade old, and the performance of a team 10 years ago shouldn't impact their performance today since in this span of time team could have changed entirely.

We therefore decided to only take the last 10 matches of each team for the historical features such as win rate, average rating, average kills and deaths, KAST, ADR, K/D and first kill differential. If a team does not have enough history, we fall back to default values based on the 25th percentile of the dataset.

We also decided that for player variables it is better to only keep team averages, because using all 10 players separately would add too many noisy features rather than helping the models.

4.3.2 Maps specific win rate

Another aspect of our feature engineering logic was to track each team's win rate across each map,

For each team we would maintain a historical win rate across the 15 maps occurring in the dataset.

This can be useful as a team can be really successful over all (High win percentage) but lack competitivity on a few maps, that's were map specific win rates becomes useful.

These map specific feature are going to be used along side all the other team metrics information.

4.3.3 Difference Features

After extracting all information we created a differential feature for the model to get a clearer understanding of which team is above for certain characteristic, we did this by subtraction such as:

- winrate_diff = Team1_WinRate - Team2_WinRate
- avgratingdiff = Team1_AvgRating - Team2_AvgRating
- avgkillsdiff = Team1_AvgKills - Team2_AvgKills
- And similarly for other metrics

This provides relative team strength, this can be more useful than the absolute value, because if the models only learn "High kill diff = win" he might struggle if both have high kills diff so the differentiate allows to know which team is better for a given feature.

4.3.4 Information Leakage prevention

Something we had to be careful about was the leakage prevention, since we use data in a chronological order we decided to sort the data by date / timestamp, That way we can easily only use data from matches that are before the current occurring matches,

One small changes but that may have major impact we have compared to classic machine learning model building is how we split our data, rather than splitting randomly 80/20 we decided to take the 80% oldest data as training dataset and use the rest 20% most recent matches for testing,

This prevent information from leaking when creating the historic of each matches, since for each match we give the history of a team it felt like if we gave to our dataset a match before one they trained one there was a risk of leakage and this is similar to how we would've used it in a real world scenario. Training it on old data and then using the current matches to see if our models is correct.

4.4 Dataset summary

After preprocessing and feature engineering, our dataset has the following statistics:

Full dataset : 126,872 matches

Train dataset : 80% => 101497 matches

Test dataset : 20% => 25375 matches

Feature dimensions : 30 features including historical team metrics, map specific win rates, and difference features (7)

Class distribution :

Train win rate: 0.503

Test win rate: 0.478

Win rate meaning that team 1 won, the class are both balanced for train and test dataset so there was no need for class balancing.

5. Model implementation and optimization

5.1 Data preparation

After the feature engineering we applied standardization to the dataset, this is useful for the Linear SVC model that is distance sensitive.

5.2 Random Forest Classifier

5.2.1 Implementation Details

```
RandomForestClassifier(
n_estimators=100,
max_depth=10,
random_state=42
)
```

n_estimators=100 This provides diversity without excessive computation since more estimators can make the model for a lot longer without improvement in accuracy or F1 Score.

max_depth=10 Limits tree depth to prevent overfitting, deeper trees risk memorizing training patterns and not learning anything so poor performance on test dataset.

random_state=42 This is just to ensures reproducibility across runs.

5.2.2 Training Results

Metric	Training dataset	Test dataset
Accuracy	0.7776	0.7395
F1-Score	0.7629	0.6987

The gap between training and test performance in accuracy is 0.0381, it indicates minimal overfitting, with the model generalizing well to unseen data.

Random Forest are often probe to overfitting so this is a normal behavior.

5.2.3 Feature Importance Analysis (Random Forest)

The Random Forest model's most important features were:

Rank	Feature	Importance
1	roundswon_diff	0.217949

2	team1_histroundswon	0.151117
3	team2_histroundswon	0.150114
4	team1_hist_win_rate	0.108340

Rounds won differential alone accounts for 21% of model importance, with the top 4 features accounting for +60% of total importance. We can see that all the top 4 features are about how much rounds a team usually wins.

5.3 Linear Support Vector Classification

5.3.1 Implementation Details

```
LinearSVC(
dual=False,
max_iter=1000,
tol=0.127
)
```

dual=False More efficient when n_features > n_samples

max_iter=1000 Provides enough iterations for convergence

tol=0.127 Sets tolerance for stopping criterion found after tuning, we ran a script that tested the best F1 score for all tol with 0.001 step from 0 to 1

5.3.2 Training Results

Metric	Training Set	Test Set
Accuracy	0.7543	0.7353
F1-Score	0.7492	0.7133

LinearSVC achieved solid performance with test accuracy of 73.52%. The train-test gap of 2.23% is small so we have a good generalization.

5.3.3 Feature Importance Analysis (LinearSVC)

Using absolute coefficients as feature importance:

Rank	Feature	Importance
1	rounds_won_diff	0.110718
2	win_rate_diff	0.097842
3	team2_hist_rounds_won	0.091889
4	team1_hist_rounds_won	0.085420
5	map_winrate_diff	0.084944

We can see that once again our top feature importance are based on the performance on the overall team, we can also see the map_winrate_diff so we now this feature was still important to add, we also notice that the importance is more evenly distributed among the top variables unlike the Random Forest

5.4 XGBoost Classifier

5.4.1 Implementation Details

```
XGBClassifier(
n_estimators=90,
learning_rate=0.14,
objective='binary:logistic',
random_state=42,
eval_metric='logloss'
)
```

n_estimators=90: Fewer trees than Random Forest but more focused through boosting, after tuning we got this gave the best result

learning_rate=0.14: Controls step size for sequential tree construction, balance between convergence speed and generalization

objective='binary:logistic': It's the objective for binary classification with probabilistic output

eval_metric='logloss' We tested different evaluation metrics and they didn't have much impact

5.4.2 Training Results

Metric	Training Set	Test Set
Accuracy	0.7807	0.7409
F1-Score	0.7724	0.7092

XGBoost is the best-performing model with test accuracy of 0.7409 but a lower F1-score than LinearSVC with 0.7092 compared to LinearSVC 0.7133. The train-test gap of remains acceptable, we have a good generalization without too much overfitting.

5.4.3 Feature Importance Analysis

Rank	Feature	Importance
1	rounds_won_diff	0.366016
2	team1_hist_rounds_won	0.225963
3	team2_hist_rounds_won	0.176966
4	avg_fkdiff_diff	0.108605
5	team2_hist_avg_fkdiff	0.033991

5.5 Baseline Comparison

To verify our models genuinely learn patterns rather than exploiting biased data, we implemented a simple baseline model:

Baseline Model: Predict the team with higher historical win rate

Baseline Results :

- Test Accuracy: 0.7187

- Test F1-Score: 0.6968

This allows us to check if our models are learning something useful. Our models outperform it by a few percentage:

If we compare the Accuracy performance compared to the baseline:

Using : ((Model accuracy / Baseline Accuracy) – 1)*100

Models	Test Accuracy	Accuracy upgrade
Random Forest	0.7395	+2.89%
LinearSVC	0.7353	+2.31%
XGBoost	0.7409	+3.1%

This shows that a Baseline comparison over the last 10 matches of each team is still a good solution, but the models we made fortunately outperforms this baseline, XGBoost as our best performing models in accuracy has the greatest improve from the baseline but all " of our models still outperforms the baseline.

6. Model Validation and Performance Analysis

6.1 Model Comparison

Model	Test Accuracy	Test F1	Train Accuracy	Overfitting %
XGBoost	<u>0.7409</u>	0.7092	0.7807	+5.37%
Random Forest	0.7395	0.6987	0.7776	+5.15%
LinearSVC	0.7353	<u>0.7133</u>	0.7543	+2.58%
Baseline with Winrate	0.7187	0.6968	/	/

All Machine Learning Models are prone to at least a bit of overfitting, this is a normal behavior and here all our 3 models still have really small overfitting, this doesn't impact the result since it's only by a few percent,

6.2 Statistical significance

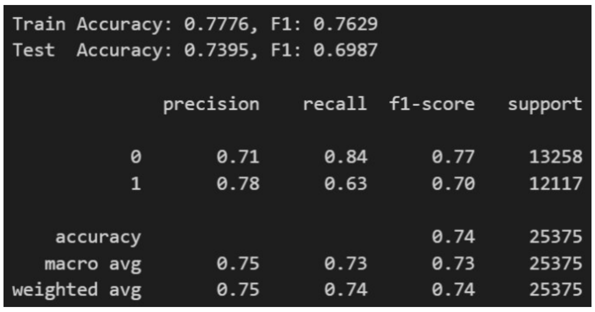
We need to understand if our results are statistically significant,

A random baseline would have an accuracy of 50% so all our models are still highly significant compared to it.

Our sample test size is from 25375 matches which is highly statistically significant.

6.3 Classification report

6.3.1 Random Forest Classification



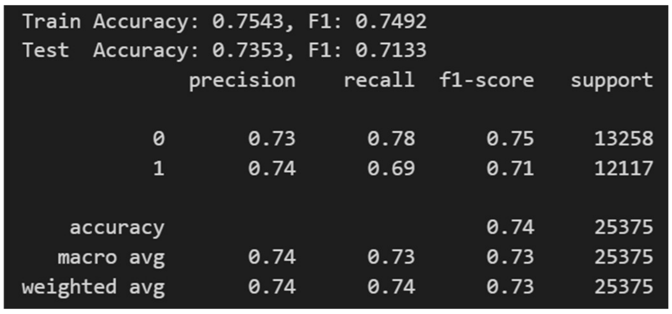
```
Train Accuracy: 0.7776, F1: 0.7629
Test  Accuracy: 0.7395, F1: 0.6987

              precision    recall  f1-score   support

           0       0.71      0.84      0.77     13258
           1       0.78      0.63      0.70     12117

    accuracy                           0.74     25375
   macro avg       0.75      0.73      0.73     25375
weighted avg       0.75      0.74      0.74     25375
```

6.3.2 LinearSVC

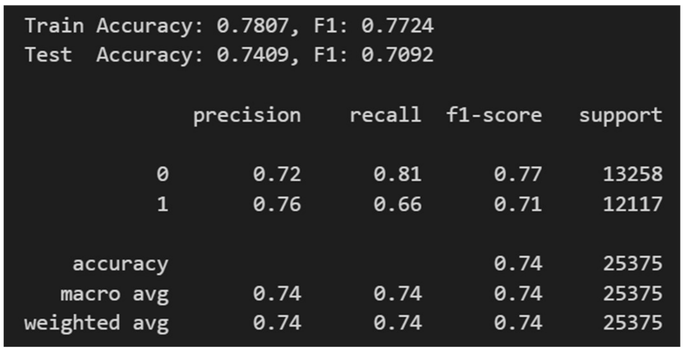


```
Train Accuracy: 0.7543, F1: 0.7492
Test  Accuracy: 0.7353, F1: 0.7133
              precision    recall  f1-score   support

           0       0.73      0.78      0.75     13258
           1       0.74      0.69      0.71     12117

    accuracy                           0.74     25375
   macro avg       0.74      0.73      0.73     25375
weighted avg       0.74      0.74      0.73     25375
```

6.3.3 XGBoost Classifier



```
Train Accuracy: 0.7807, F1: 0.7724
Test  Accuracy: 0.7409, F1: 0.7092

              precision    recall  f1-score   support

           0       0.72      0.81      0.77     13258
           1       0.76      0.66      0.71     12117

    accuracy                           0.74     25375
   macro avg       0.74      0.74      0.74     25375
weighted avg       0.74      0.74      0.74     25375
```

6.3.4 Analysis

All our models have good recall and precision with good performances across both classes.

Based on the result shown above, we selected the model XGBoost, it has the highest Test accuracy, the second highest F1 score and the model is really efficient during training.

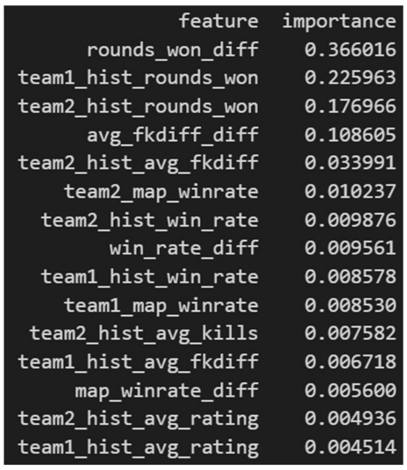
7. Analysis and recommendations

7.1 Key findings

7.1.1 Team consistency over individual performance

The most striking finding from our feature importance analysis across all three models is the dominance of team historical metrics over any individual player statistics.

Here is the 15 most important features of XGBoost Classifier:



```
              feature  importance
      rounds_won_diff    0.366016
team1_hist_rounds_won    0.225963
team2_hist_rounds_won    0.176966
      avg_fkdiff_diff    0.108605
team2_hist_avg_fkdiff    0.033991
    team2_map_winrate    0.010237
  team2_hist_win_rate    0.009876
        win_rate_diff    0.009561
  team1_hist_win_rate    0.008578
    team1_map_winrate    0.008530
 team2_hist_avg_kills    0.007582
team1_hist_avg_fkdiff    0.006718
     map_winrate_diff    0.005600
team2_hist_avg_rating    0.004936
team1_hist_avg_rating    0.004514
```

The top features across all models are related rounds won differential and team historical rounds won, An individual variable that is still important is the first kill difference; we did not expect this feature to be that important, but apparently the team that consistently gets the first kill in a round gains a large strategic advantage over the whole match.

Important but secondary features are map win rate, depending on the models their importance varies,

This validates the hypothesis that in professional CS:GO, team cohesion, momentum, and consistency are more predictive of match outcomes than the individual players' performance.

7.1.2 Map specific performance patterns

Tracking the win rates of teams on each map revealed that performance varies a lot between maps being played.
This feature showed that even if some team have smaller overall win rate, they could still be better because they were good at this specific map.

This feature also has a good level of importance on models, so its information is meaningful.

7.1.3 Our performance compared to related work

Our highest accuracy was achieved by our XGBoost with an accuracy of 74.09%

To compare it with other works on the field, the K/D regression(4) approach reports around 76% accuracy. So, our method slightly under perform against this model but our solution is easier to expand to add more features compared to their approach of simplifying the win by the highest K/D ratio.

Other work in the field such as the K-means and Neural Networks by Chalmers University of Technology(5) use way more complex algorithms but underperform with only 65% accuracy, but their work is still very interesting and maybe we could apply in another way what they tried to do.

7.2 Feature Importance Consensus

Across all 3 models we used, we see clear groups of important features:

Rank	Feature

1	Rounds won differential, team historical rounds won
2	Opening kill differential, team historical win rates, rating differentials
3	Map-specific features, individual player metrics
4	Specific death/kill counts don’t really impact the model

This is similar across different models and suggests these patterns represent genuine signal rather than model-specific artifacts.

7.3 Model Limitations and Challenges

1. Prediction Difficulty: Sports and esports outcomes have lower predictability than technical domains, unlike housing prices for example, sports involve human psychology and tactical innovation.

2. Roster Changes: Professional esports teams frequently change players, making historical features less relevant for recently reorganized teams.

3. Strategic Evolution: Counter-Strike meta-game evolves significantly, dominant strategies from 2015 differ from 2023.

4. Lack of Variables: We do not capture tactical information, patch updates, coaching changes, player injuries, or psychological factors.

7.4 Recommendations

7.4.1 For ESports Organizations

- Work toward team consistency: Coaching staff should prioritize roster stability and team cohesion over chasing individual star players, as our analysis shows team historical performance dominates individual performance. This is actually a funny side note since an actual pro team lived this exact situation(12), a pro team recruited a team full of star player that were recognized for being extremely talented players but they lacked cohesion despite their amazing individual performance.

- Map Preparation: Focus preparation efforts on maps where team performance is weakest, as map expertise is clearly valuable.
- Match Preparation: For important matches, teams should focus on understanding opponent historical patterns and recent momentum, they should study their strategy since the beginning of each round is very important as seen by the first kill feature.

7.4.3 For Future Research

- Use Unsupervised Learning: Apply k-means clustering (following[(5)]) to identify player types, style of play and assess how a player's role affects the outcomes.
- Add Contextual Features: Include tournament type, stakes, scheduling factors, player injury information, but also more complex data like number of native language in a team as a team were everyone speaks their native language might have better communication, how much jet lag are they on if they moved to a physical tournament, etc.…
- Real-Time Adjustments: Develop models that can update historical features in-game, allowing mid-match predictions

8. Conclusion

8.1 Summary of Work

This project successfully demonstrated the application of machine learning to predict competitive gaming outcome, addressing the research question: "How can machine learning predict professional Counter-Strike match outcomes using team historical performance and player statistics?"

Through data collection of 126,872 matches, innovative feature engineering based on team momentum and map-specific performance, and implementation of three complementary classification models (Random Forest, LinearSVC, and XGBoost), we achieved a best test accuracy of 74.09% with F1-score of 0.7092 on our XGBoost model, it outperformed both the Random Forest and simple historical win-rate baseline, with a 24-point improvement over random guessing.

8.2 Accomplissements

- Methodological Rigor: Implementation of proper chronological train-test splitting and careful feature engineering preventing information leakage, on dataset of 126,872 professional matches
- Feature Innovation: Introduction of team-specific map win rate tracking rather than static map encoding, capturing real strategic differences in team performance across environments
- Feature Importance Analysis: Systematic analysis revealing that team historical consistency outweighs individual player performance, showing our assumptions wrong about esports outcomes.
- Three-Model Comparison: Comprehensive evaluation of Random Forest, LinearSVC, and XGBoost on identical features and splits, with XGBoost emerging as the best model between performance and computational efficiency.
- Baseline Comparison: Demonstration with a naive approach of baseline win-rate, that shows our models is 2.22pp improvement over historical win-rate baseline

8.3 What Went Right

- Data Quality and Quantity: Successfully processed 126,872 matches without any problems.
- Feature Engineering: Created team momentum features across a sliding time window, map-specific win-rate tracking, individual performance to be processed as team average.
- Model Selection: We chose various algorithms such as decision tree, linear classifier, and gradient boosting providing different perspectives and all models ended up performing good.
- Temporal Logic: A chronological data split for train and preventing information leakage from a previous match.
- Analysis: We provided the importance of each major feature and analyzed the result, we compared models and chose the best one accordingly.

- Performance: 74.09% accuracy has a significant real-world usefulness for esports analytics.

8.4 What Could Be Improved

- Better Accuracy: 74.09% accuracy, even though it's still a good accuracy, some more complex model such as neural network might achieve better performance when trained with good enough data.

- Change Temporal Window: We tried a fixed 10-match historical window because it was in our dataset what was giving the best result, but it may not be the most optimal solution. For example, we could capture all matches before the last roster change. Meaning we would need to track each team recruitment and when it occurs, we restart the sliding window. We could also look for a better solution for team without enough matches for us to achieve a real average. And in our cases, we could look for what to do right after a roster change. Look if we need give default value or do we give the value of the previous roster.

- Feature Expansion: In the same logic an improvement would be if we considered more features. What could be an improvement are features out of the game but more physical, if the event is online or physically in an arena, if players speak the same language natively, if they are under jetlag.

- Deployment: Currently the model is only used to see test and training accuracy, but we have not had time to deploy it for real-world use, we will try to finish working on it.

- Real-Time Predictions: Our current approach only uses pre-match data, a new model could look at on-going game data to predict in live the result of a match.

8.5 Learning Outcomes

Through this first machine learning project, we learned a lot. We discovered that data preprocessing and feature engineering is just as complex as finding the correct model for a given problem. We learned the process to build a machine learning model from acquiring the data from the web and all step up to creating models and choosing the one that best fits our given problem. We also learned that sometimes the general solution cannot be applied to you, for example to the data split we add to come up with our own split method.

ANNEXES:

Bibliography:

All the sources listed in the References section were consulted during this work for a deeper understanding, idea of model design, and a comparison with related work.

License of the dataset used:

References:

(1) : https://www.kaggle.com/datasets/mateusdmachado/csgo-professional-matches
(2) : https://www.hltv.org/player/11893/zywoo
(3) : https://www.selectdataset.com/dataset/821b0c9dceefbb6ef33ee5fd4df1ba46

(4) : https://medium.com/analytics-vidhya/counter-strike-matches-result-prediction-28a8ca076971

(5) : https://odr.chalmers.se/server/api/core/bitstreams/e919ca56-3995-46b1-9546-a09b83f8b172/content

(6) : https://www.faceit.com/en/game/cs2

(7) : 'team1_hist_win_rate', 'team1_hist_avg_rating', 'team1_hist_avg_kills', 'team1_hist_avg_deaths', 'team1_hist_avg_kast', 'team1_hist_avg_adr', 'team1_hist_avg_kddiff', 'team1_hist_avg_fkdiff', 'team1_hist_rounds_won', 'team2_hist_win_rate', 'team2_hist_avg_rating', 'team2_hist_avg_kills', 'team2_hist_avg_deaths', 'team2_hist_avg_kast', 'team2_hist_avg_adr', 'team2_hist_avg_kddiff', 'team2_hist_avg_fkdiff', 'team2_hist_rounds_won', 'team1_map_winrate', 'team2_map_winrate', 'win_rate_diff', 'avg_rating_diff', 'avg_kills_diff', 'avg_deaths_diff', 'avg_kast_diff', 'avg_adr_diff', 'avg_kddiff_diff', 'avg_fkdiff_diff', 'rounds_won_diff', 'map_winrate_diff'

(8) :https://www.researchgate.net/publication/358076807_Predicting_the_Outcomes_of_Football_Matches_Using_Machine_Learning_Approach

(9) : https://medium.com/@fion.ouyang/predicting-soccer-match-outcomes-a-data-driven-approach-df6bea2429ec

(10): https://github.com/prathameshtari/Predicting-Football-Match-Outcome-using-Machine-Learning

(11): https://www.nature.com/articles/s41598-025-91870-8

(12): https://community.skin.club/en/news/why-the-falcons-superteam-failed-in-budapest

Acknowledgements:

We would like to thank our lecturer, Mafas RAHEEM, for teaching us the fundamentals of machine learning, for giving us the opportunity to work on this project, and for the guidance during the module. We also thank APU for allowing us to study in this program and be part of this course. We are also grateful to Franck DOMENGE, our lecturer at EFREI, who made us love data science.