Outcome Prediction of eSport Matches using Spatial Data and Machine Learning

Thesis

Submitted to the

Institute of Geodesy and Geoinformation Science
Technische Universität Berlin
Faculty VI - Planen Bauen Umwelt

In Fulfillment of the
Requirements for the Degree of
Master of Science

Submitted by
Tuo Kang
Matriculation Number: 395923

Supervisor: Prof. Dr. Martin Kada
Advisor: M.Sc. Valentina Schmidt

Berlin, September 28, 2020

Acknowledgements

I would like to thank Prof. Dr. Kada for providing me with the opportunity to work on this topic with him and Ms. Valentina Schmidt for her help.

I would also like to thank TU Berlin for giving me the chance to pursue this Master degree.

Special thanks to GRID eSports for providing the raw data and the support from the whole team, especially to tribe data-platform for their help on learning the language Go and parsing the data, and to Mathias Holmstrøm for his help on modelling, research materials and writing suggestions.

Additional special thanks to Christopher O'Hara for all of his support, encouragement and content editing. This will be way harder without him.

Furthermore I would like to thank the author of the parser Markus Walther for his help on starting the implementation of the whole work. Thanks to Fan Wu for his supporting on literature searching and Haozhi Sun for his help with the game CS:GO in general.

Also, I would like to thank my family, especially my cousin Katie, I wouldn't make this far without their support.

At last, thanks to whoever discovered coffee, so that I can stay awake in those nights writing this thesis.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 28.09.2020

[illegible]

. . . .

(Signature [Tuo Kang])

Abstract

Predicting game results has always been popular in different sports fields and with the development of technology. eSports has become a big part of young people's lives with growing trends in players from casual gameplay to professional gameplay. One of the most popular methods of making predictions is by utilizing machine learning methods and algorithms. By applying machine learning models, complex player behavior can be analyzed for predicting individual and long-term match results. As such, to solve this problem of this work, Logistic Regression (LR), Random Forest (RF), and LightGBM (LGBM) are used to predict the possibilities of a team winning of each round of Counter Strike: Global Offensive games. Counter Strike is a popular first-person shooting game that has a rich community and plenty of accessible data and results.

There are a seemingly endless number of features, events, and environmental factors that have an influence or correlation on gameplay and match results. In this thesis work, instead of player skills or team ratings, only the features that represent the current, real-time game situation (e.g., number of alive players) or features related to spatial data are investigated for modeling and analyzing results via machine learning models. Distance is considered to be a direct way of showing position relations, which ultimately, have a large influence on match results. Therefore, different features related to player-to-player distances and player-to-bomb distances were used to represent the positional relations in the games. Two path planning algorithms, A* and Floyd-Warshall, were used to obtain distances so that they are reflective to actual player distances in the matches.

For the three machine learning algorithms and two path planning algorithms, the data are trained one time without hyper-parameter tuning and one time with tuning. Consequently, there are a total of 12 predictions for a single set of data to compare with the best accuracy's being reported as: 88.1% for LGBM, 87.9% for RF, and 85.9% for LR and an observation that LGBM is more sensitive to parameter tuning.

This thesis verifies that the round winning situation can be successfully predicted with satisfactory results. Furthermore, ensemble machine leaning methods work better than Logistic Regression for this specific problem and setup. Lastly, in this work, the Floyd-Warshall algorithm was shown to work better with ensemble methods while A* is better with Logistic Regression.

Key Words: Machine Learning, eSports, Path Planning

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

With the rapid changes in the field of computer science, recent advancements have introduced many novel and interesting areas for the development of scientific research. A commonality in current scientific research is the dependency on data. Video games are a good area for data extraction techniques as they are rich with digitized data. Furthermore, due to the commercialized nature of video games, data acquisition is plentiful and readily available.

The goal of this thesis is to use data and tendencies to predict match results with some probability based on environmental factors. For this purpose, Counter-Strike: Global Offensive (CS:GO) is chosen for the analysis as it is a highly popular game with nearly endless data being publicly available and maintained. The concept is to take previous CS:GO game results and evaluate the requirements for a successful "round win" based on the position of the players on the digital map given some layout and predict future outcomes utilizing different factors via machine learning.

Predicting match results can be useful in several domains. The premise is that, if positional information is a major contributor on the outcome of the match, then players can develop strategies incorporating configurations and patterns based on probabilistic results. This can be extended to create a "training engine" that simulates match results, provided there is enough data. Teams could analyze the current situation in real time to determine the next stratagem. Furthermore, artificial agents can be trained to eliminate players based on their position information and other features (e.g. weapon type). These agents may learn how to handle encounters in real-time based on the current results. While somewhat facile, highly accurate models could be extended and employed in real-life police force or military training simulations (with tuning for real-life parameters). This could improve safety in various operations that require teamwork to complete a task.

Besides team side, the betting market for all sort of sports all over the world is worth a huge amount of money, which means the match outcome prediction is rather important to the bookmakers for profit reason. Some bookmakers use sophisticated pricing models to reflect the game results likelihood shown as "odds" [BNR19] for people to make a bet. With using more in-game features, the bookmakers can improve their own predictions for higher accuracy.

1.2 CS:GO

CS:GO is a multiplayer first-person shooter video game. The game has two teams against each other: Terrorists (T) and Counter-Terrorists (CT). Normally, one game contains 30 rounds and one round lasts 2 minutes. After the first 15 rounds, the teams will need to switch sides and the first team that wins 16 rounds wins the game. Also, there will be overtime rounds once the scores are 15-15.

In competitive maps of the game, T-side players must plant the bomb in specific areas on the map and try to eliminating all CT-side players to prevent the bomb from being defused, while the CT players must either prevent the bomb from being planted or defuse the bomb. After the bomb is been planted, the CT players have 40 seconds to defuse the bomb until it explodes. If the bomb explodes, it means T players win the round and CT lose.

From the beginning of the game, both teams have a default amount of money that can be used to buy weapons of different values with varying usage specifications and effective ranges depending on the type. In this case, depending on the position of the players on the map, different weapons have a varying level of usefulness (general case) with respect to winning a match. For example, a knife can be useful within three steps (close proximity) while grenades impact a greater range (area of effect) and can be used further away.

There are several different maps in the game based on real cities in the world. For all the factors above, it would relatively hard to analyze all the maps in this thesis. For this reason, only the map "Mirage" will be used since it is one of the most played ones. A simple 2D Mirage map can be seen in Fig. 1.1 below. In this image, dark areas are not reachable, green parts are the start areas of the two sides and red parts are the areas that are allowed for the Terrorists to plant the bomb.

All the recorded matches can be replayed using the demonstration files which can be obtained from HLTV (Half-Life TV, a website and forum containing news, tournament events, and statistics of professional CS:GO matches). While replaying the demonstration files, all the necessary data can be parsed and manipulated after for further needs.

Before applying machine learning on predicting the results, people tend to make simple pre-match predictions based on historical data. For instance, the historical win percentage for different maps, over all T/CT-side win percentage, how much times both teams has played against each other and the results, etc.

With the higher demand, only with historical data is not sufficient, since it doesn't conclude enough information and the prediction is not accurate enough. So, predicting match results using live game score has come to the market. With the live score, it is much easier to describe the current game situation to make better prediction. For example, it is much easier for a team to win a game with a score of 14 than with 7, regardless the historical win probability.

However, using scores as the only feature is not sufficient, especially when the game scores are close in value (e.g. 14:15). In this situation, what is actually happening in the current round is essential for the results prediction. For example, Team A having four

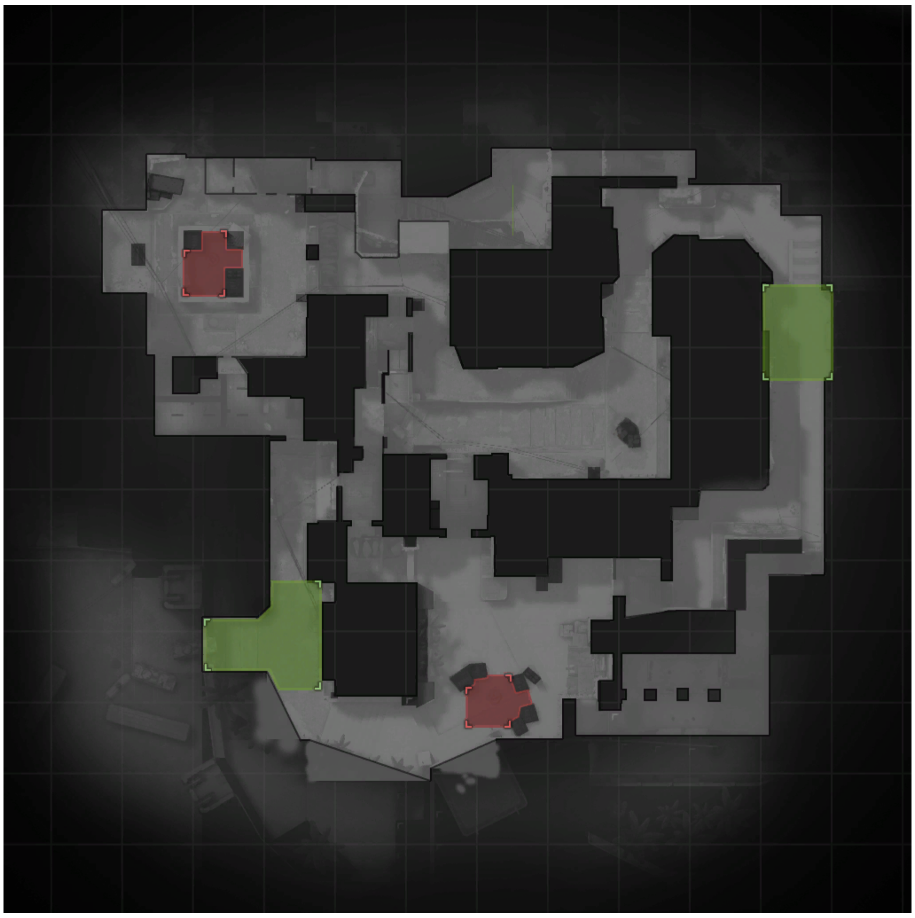


Figure 1.1: Map Mirage

alive players is more likely to win a round than Team B having only one alive player. Therefore, generating the round win-lose prediction using live data is the main goal for this thesis.

Unlike most works in sports prediction only using player and team statistics, spatial data are utilized to analyze their contribution in predicting rounds results since the player movement is critical to win the round.

1.3 Objectives

As stated in last section, this thesis aims to provide an independent experiment for verifying and validating the usage of machine learning algorithms with spatial data for live CS:GO round win-lose prediction. In this thesis, spatial data are used by calculating the distances between the players themselves and to the bomb instead of the players' and bombs' spatial coordinates. The distances are obtained from path planning algorithms while simulating the actual player movements.

The research objectives include:

1. Predicting the winning results of a round based on the position of bombs planted relative to alive players statistical features.
2. Analyzing the impact of common game-based and position-related features on the probability of a team winning a round.
3. Analyzing the impact of distances calculated by different path planning algorithms on the probability of winning a round.
4. Evaluating the performance of different ensemble learning and supervised learning models on round outcomes.
5. Evaluating the impact of various ensemble learning and supervised learning algorithm hyper-parameters on the probability of teams winning an individual round.

1.3.1 Approach

To perform the best prediction, the following features are looped over and combined together as different combinations. Each combination is used to generate a prediction and Logloss then is used to find the best combination for further processes. Feature importance of the best combinations are printed out to analyze Objective 2.

- Bomb-Player distance
- Player-Player distance
- Ratio of Alive Players further than threshold distance
- Number of High-Performing Players
- Number of Alive Players
- Available Flash Bang numbers
- Available Molotov (Grenade) numbers
- Available Smoke Grenade numbers
- Team Total Weapon Cost

Bombs explode after 40 seconds of deployment. In the 40 seconds of the bomb being planted, the round winning probability was predicted at an interval of every 5 seconds with the corresponding best feature combination for Random Forest, Logistic Regression and LightGBM algorithms. Then, the hyper-parameters were tuned to get the final prediction results for evaluation.

Accuracy, Logloss and F1-Score are used as metrics to evaluate the predictions for Objectives 1, 3, 4, 5.

1.3.2 Assumptions

There are a number of assumptions made prior to designing the model architecture and algorithm selection. Several of the assumptions are based on the objectives while others are based on the application of algorithms/models for the project motivation.

- The above objectives are consistent, coherent, and conflict-free compared to the other objectives.
- A trend of individual round winnings will imply long-term player/team performance.
- The ratio of remaining Alive players will have the highest impact on the round results.
- Spatial features will have the second-highest impact on the round results as compared to weapon costs and remaining grenade numbers.
- Models trained on historical data will have a similar performance on real-time rounds (e.g. historical environment behavior does not change).
- Path Planning Algorithms will simulate mostly how the players' actual movement with acceptable error.
- Different players are considered having the same skill level in this work since the data are all from professional games and measuring the skill level is another complicated topic.
- Players are assumed to have the same moving speed. Running, creeping around are ignored in this work.

1.3.3 Out of Scope

This section specifically states aspect that were decided to be outside of the scope of the project. Either the complexity was too high, the assumed impact was considered negligible, or the scope was too broad/non-specific.

- No predictions for whole game results, only valid for round winning predictions in the bomb-planted situation. Other round winning possibilities are not considered here (e.g CT players killed all T players before the bomb is planted). However, the results of this work can be used as a feature for the whole game results prediction.
- The number of remaining grenade-type weapons and total weapon cost is a function of the number of alive players. However, the impact of the relation of those features on the round result is not conducted or handled (high-complexity challenge, time-variant).
- Besides walking or running, players can also jump in this game. However, 3D player movement is not considered in this thesis due to higher complexity in simulating.

1.4 Outline

There are seven chapters in this thesis and a brief introduction of the chapters is given in this section.

Chapter 2 - Background gives a detailed explanation of the related techniques of this work and a comparison of the mentioned techniques.

Chapter 3 - Related Work describes similar works to this thesis by different algorithms and compared the results of the works.

Chapter 4 - Methodology describes the structure of the thesis work, what data and how the related techniques are been used to solve the problem.

Chapter 5 - Implementation describes a more detailed working process of the thesis, including the working environment, how the data been extracted, the settings of the default values for the implementation and visualization of the implemented results.

Chapter 6 - Evaluation explained the metrics used for the prediction and a brief evaluation of the results with graphs and tables.

Chapter 7 - Conclusion contains the discussion and a summary of the whole thesis and lists the future work for improvement.

2 Background

2.1 Machine learning

Machine Learning (ML) is the "scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead" [Wikb]. Generally, machine learning is considered a subset of *Artificial Intelligence* (AI) in which ML algorithms create mathematical models based on training data (samples). The goal is to make accurate predictions or decisions without being explicitly programmed to perform the task [OE16]. The quantity and quality of data is another key aspect for input data as algorithms are highly "data hungry". This means that models occasionally require millions of observations to reach acceptable performance levels. In addition, biases in data collection can substantially affect both performance and generalizability [Sam00]. Over-fitting and under-fitting are common concerns for generalized models, though over-fitting (at the model level) can be acceptable if the model only has a single purpose.

ML is a ubiquitous problem-solving approach used throughout many forms of science dealing with complexity. For example, millions of images acquired via telescopes for the classification and clustering of galaxies would be practically impossible for humans to complete [OE16]. Similarly, ML algorithms are used to predict optimal patient diets in bio-medicine, by analyzing genetic sequences and deficiencies to recommend personalized nutrient consumption. Generally, there are three types of learning: supervised, unsupervised, and reinforcement.

In supervised learning, labeled data are used to train the machine to learn to find the correct patterns. Different supervised learning methods are used in multiple fields. For example, Joslyn et al. uses Supported Vector Regressor and Extreme Gradient Boosting to predict water quality [JL18], Lin et al. used Neural Network to predict earthquake magnitude [LCC18a] and Khatri et al. utilized various classification methods to detect credit card fraud [KAA20].

For unsupervised learning, there are no labels for the data and the machine is trained to find any hidden patterns that suit the situation. It can be used to do image classification, for example, Weber et al. presented an unsupervised learning method from unlabeled and unsegmented cluttered scenes for visual object recognition [WWP00].

Reinforcement learning is concerned with how software agents should take actions in certain environment in order to maximize the notion of cumulative reward [Wikc]. AlphaGo is a successful example for reinforcement learning. Besides that, robotics is an common application field for reinforcement learning. An overview can be seen in [AB09], [DP13], [KP13].

For the purpose of this thesis - predicting eSports outcome, it is more clear to label the features and target feature (win or lose) and using supervised learning to accomplish the prediction. Some of the most popular supervised learning algorithms are: Support Vector Machines, Logistic Regression, Decision Trees, Neural Networks, Random Forests, and Boosting. The performance of these algorithms in the specific domain will be explained in the following section.

2.1.1 Random Forest

Random Forest is an ensemble learning algorithm based on Decision Trees. It combines several randomly selected decision trees and to make the variance impact less for the results, RF uses the bagging to build the decision trees.

For each decision tree in the forest, a bootstrap sample $S^{(I)}$ is selected from S. Then, a random subset (f) of all the features (F) is used, instead of applying all potential splits of the feature set, at all nodes of the tree. This reduces the correlations between the trees, due to the same features are likely to show up multiple times when splitting the bootstrap samples. Next, the nodes split based on the best features of the random subset f. It narrows down the feature set so that decrease the learning time for each tree. The pseudo-code is shown in Algorithm 1 [Ber17].

Algorithm 1 Random Forest Algorithm

Precondition: Initialize both open and closed list A training set $S := (x_1, y_1), ..., (x_n, y_n)$, features F, and number of trees in forest B.

```
function RandomForest($S, F$)
   $H \leftarrow \emptyset$
      for $i \in 1, \ldots, B$ do
         $S^{(i)} \leftarrow$ A bootstrap sample from $S$
         $h_i \leftarrow$ RandomizedTreeLearn($S^{(i)}, F$)
         $H \leftarrow H \cup \{h_i\}$
      end for
      return $H$
end function

   function RandomizedTreeLearn($S, F$)
         At each node:
            $f \leftarrow$ very small subset of $F$
            Split on best feature in $f$
         return The learned tree
   end function
```

2.1.2 Logistic Regression

Logistic regression (LR) is a ubiquitous statistical method for modeling a binomial outcome with one or more explanatory variables. Typically, this binomial outcome is given binary values and the Bernoulli family of distributions is chosen to model conditional distributions. As such, LR is commonly implemented when data consists of a categorical and continuous variable. The basis of logistic regression is derived from the logit (or log-odds ℓ, [Ber44]) where:

$$logit\, p = \ln\left(\frac{p}{1-p}\right) \quad for\, 0 < p < 1 \tag{2.1}$$

Given that p is the probability. With some mathematical manipulation, the log-odds can be applied to linear systems in the form of:

$$\ell = \log_b\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{2.2}$$

Such that ℓ is the log-odds, b is the base of the logarithm, x_i are predictors, and β_i are parameters of the model. Further mathematical manipulation allows for p to be calculated with fixed β_i values. However, these β_i values must be derived via logistic regression.

$$p = \frac{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}}{b^{\beta_0+\beta_1 x_1+\beta_2 x_2}+1} = \frac{1}{1+b^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}}. \tag{2.3}$$

Logistic regression coefficients can be computed via the maximum likelihood estimate. Model fitting in machine learning seeks to bind the value of a random variable between 0 and 1 for experimental data via normalization [Ng20].

2.1.3 Light Gradient Boosting Machine

Light Gradient Boosting Machine (LGBM) is a histogram-based heuristic extension of Gradient Boosting.

Gradient Boosting (GM) is a technique applied to regression and classification problems, typically decision trees, for producing prediction models [Fri01]. GM is driven by a loss function that allows for generalization and optimization around a gradient. In supervised learning described with a joint probability distribution for a training set, an approximation $\hat{F}(x)$ minimizes the expected value $\mathbb{E}_{x,y}$ of a loss function, $L(y, F(x))$. This can be written in the following form:

$$\hat{F} = F \arg\min \mathbb{E}_{x,y}[L(y, F(x))]. \tag{2.4}$$

The method for gradient boosting is then initialized with a constant value for:

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const.} \tag{2.5}$$

where $h_i(x)$ is the weighted sum of functions are in a class of weak learners, $\mathcal{H}$. Next, the approximation attempts to minimize the average loss function value on the training set for the constant function and greedily expands. The process is iterative and computationally exhaustive (optimization is infeasible), resulting in the need for simplification via a heuristic. At this point, functional gradient descent is implemented based on the pseudo-residuals r_{im} as shown below:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n. \tag{2.6}$$

A weak learner is fit to r_{im} and the step length multiplier γ_m is computed via:

$$\gamma_m = \gamma arg\,min \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right) \tag{2.7}$$

Next, the model is updated in the form of:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{2.8}$$

and the result is output.

LGBM takes continuous feature/attribute values from buckets and places them into discrete bins. The motivation behind this is reducing memory usage and speeding up training as the cost of calculating the gain for each split is reduced. Furthermore, this approach also reduces the communication cost for parallel learning.

LGBM uses workers to hold the full amount of data. Workers find the best local split on a local feature set. Then, workers communicate the best local splits to each other and select the best split. Afterward, the best split is performed. Each worker does not communicate the split results for data as each worker knows how to split the data. Therefore, LGBM is a suitable algorithm unless the size of the data is too large. To speed up training, LGBM uses histogram subtraction which determines the percent positive by subtracting the negative control from histogram neighbors.

2.2 Path Planning Algorithms

2.2.1 A*

A* is a best-first search algorithm that focuses on finding a path with the lowest cost. It is an extension of the Dijkstra search algorithm combined with heuristics similar to Greedy Best-First-Search algorithms [HNR68]. The distance from the source node to some node n, *g(n)* is combined with a heuristic function, *h(n)*, that estimates the minimum cost from the current node n to the goal node. The combination of *g(n)* and *h(n)* is the current approximated cost of the shortest path to the goal node going through some node n (given as *f(n)*).

An open list and closed list are initialized and a start node is selected. The algorithm then iterates through the graph/tree using a priority queue on the open set where the node with the lowest *f(x)* value is removed and the neighbors are added to the queue. Once the queue is empty, the final f value can be returned as the shortest path. The general A* algorithm's pseudocode is provided in Algo. 2 [SAL+12].

A* can be used to find the best path of players and the location of the players are viewed as nodes in the graph (Fig. 2.1) [Jon]. Djojo et al. used Dijkstra, A *, and Floyd-Warshall algorithms on the mesh network. Various metrics was used to compare the performance of all three algorithms and it was concluded that A * algorithm has the smallest computational load while Dijkstra performed the best on memory usage [DK13]. In video game path finding, Lawrence et al. used A* and Deep-Q Learning (DQL) to find player path in 2v1 scenario to analyse player strategy in CS:GO, and the results shows that A* generated paths have better success rate and learns faster [LB13].

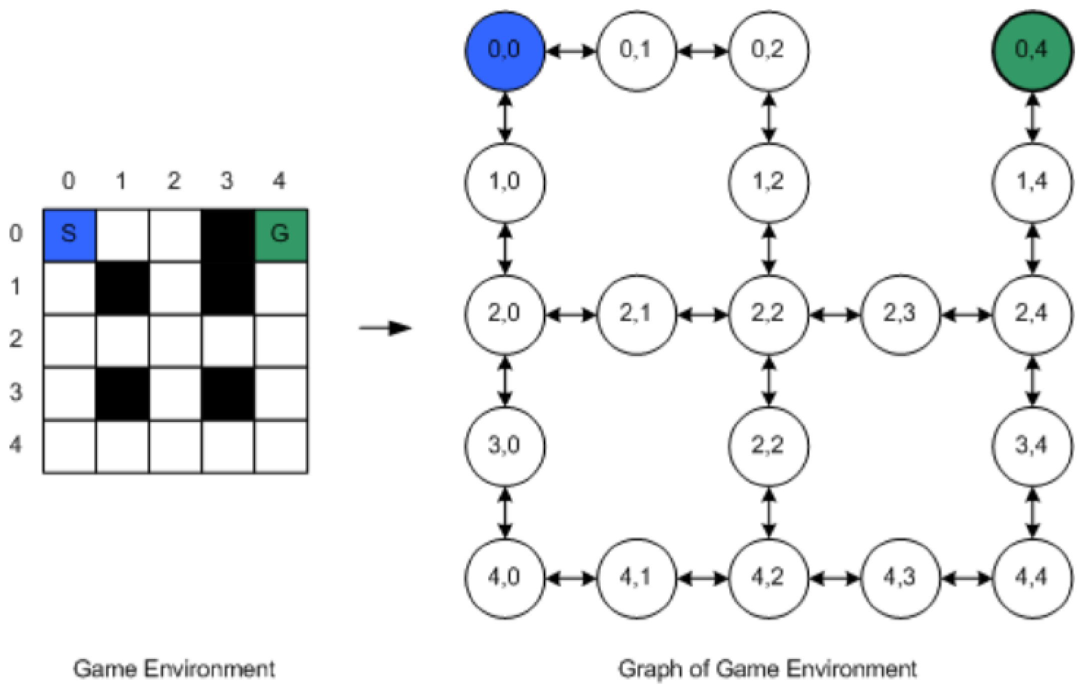


Figure 2.1: A* implements graph search to identify the best path [Jon].

Algorithm 2 A* Algorithm

```
1: procedure A*
2:     Initialize both open and closed list
3:     Add the start node
4:
5:     while open_list ≠ empty do
6:         $m$ = Node on top of open_list, with least $f$
7:         if $m$ == end then
8:             return
9:         remove $m$ from open_list
10:        add $m$ to closed_list
11:        for each $n$ in child(m)
12:        if $n$ is in closed_list then
13:            continue
14:            cost = g(m) + distance(m,n)
15:            if $n$ in open_list and cost < g(n) then
16:                remove $n$ from open_list                ▷ new path is better
17:            if $n$ in closed_list and cost < g(n) then
18:                remove $n$ from closed_list
19:            if $n$ not in open_list and $n$ not in closed_list then
20:                add $n$ to open_list
21:                g(n) = cost
22:                h(n) = heuristic_function(n, end)
23:                f(n) = g(n) + h(n)
24:    return failure
```

2.2.2 Floyd-Warshall

The Floyd-Warshall Algorithm uses graphs with weighted edges to calculate the shortest path by comparing all possibilities of the movement for all nodes [Flo62]. The algorithm recursively iterates through the constructed graph until the optimal estimate is returned. The implementation only returns the path lengths between all pairs of vertices, so a shortest-path tree can be calculated based on a matrix of the discrete values that were derived iteratively. The concatenation of all shortest path lengths results in the shortest cumulative path as being optimal. The Floyd-Warshall algorithm is a suitable choice for dense graphs that have a majority of pairs connected by edges [PS03]. Azis et al. compared Floyd-Warshall and Greedy Algorithm in finding the shortest route and concluded that although Floyd-Warshall algorithm took longer to find the route, it gave the result with the best accuracy [AdMLS18].

Algorithm 3 Floyd-Warshall Algorithm

```
Let: dist be a $|V| \times |V|$ array of minimum distances initialized to $\infty$
 1: procedure FloydWarshallWithPathReconstruction
 2:     for each edge (u, v) do
 3:         dist[u][v] ← w(u, v)                      ▷ The weight of the edge (u, v)
 4:     for each vertex v do
 5:         dist[v][v] ← 0
 6:     for $k$ from 1 to $|V|$ do                   ▷ standard Floyd-Warshall implementation
 7:         for $i$ from 1 to $|V|$
 8:             for $j$ from 1 to $|V|$
 9:     if dist[$i$][$j$] > dist[$i$][$k$] + dist[$k$][$j$] then
10:             dist[$i$][$j$] ← dist[$i$][$k$] + dist[$k$][$j$]
```

2.2.3 Rapidly-exploring Random Trees

The Rapidly-exploring Random Tree (RRT) was designed for searching non-convex, high-dimensional spaces efficiently by investigating sections of a space-filling tree that have been previously unsearched [VFKP09]. In this way, the algorithm seeks to maximize its breadth into configuration spaces with a bias towards the largest Voronoi regions. RRT* is an extension of search algorithm, RRT.

RRTs has succeed on various hard planning problems. For example, planning path under challenging constrains [LaV98] and kinodynamic tasks [EFF02]. Chen et al. used an innovative RRT-based algorithm smooth the path for robot to move along and improved the planning efficiency [CST$^+$18] and Wu et al. used another RRT-based algorithm to plan paths for UAVs and proved it simpler, faster and more accurate than A* [XCY14].

The RRT* algorithm's pseudocode is provided in Algo. 4 [KF11].

Algorithm 4 RRT* Algorithm

```
 1: procedure RRT*
 2:     V ← $x_{init}$; E ← $\varnothing$
 3:       for $i = 1, \ldots, n$ do
 4:       $x_{rand}$ ← SampleFree$_i$
 5:       $x_{nearest}$ ← Nearest(G = (V,E), $x_{rand}$)
 6:       $x_{new}$ ← Steer($x_{nearest}, x_{rand}$)
 7:       if ObstacleFree ($x_{nearest}, x_{rand}$) then
 8:         $X_{near}$ ← Near(G = (V,E), $x_{new}$, min{$\gamma$ RRT*(log(card(V))/card(V))$^{\frac{1}{d}}$, $\eta$})
 9:         V ← V ∪ {$x_{new}$}
10:         $x_{min} \leftarrow x_{nearest}; c_{min}$ ← Cost($x_{nearest}$) + $c$(Line($x_{nearest}, x_{new}$))
11:         foreach $x_{near} \in X_{near}$ do          ▷ Connect along a minimum-cost path
12:           if CollisionFree($x_{near}, x_{new}$) $\wedge$
13:           Cost($x_{near}$) + $c$(Line($x_{near}, x_{new}$)) $< c_{min}$ then
14:             $x_{min} \leftarrow x_{near}; c_{min}$ ← Cost($x_{near}$) + $c$(Line($x_{near}, x_{new}$))
15:         $E \leftarrow E \cup \{(x_{min}, x_{new})\}$
16:         foreach $x_{near} \in X_{near}$ do                        ▷ Rewire the tree
17:           if CollisionFree($x_{new}, x_{near}$) $\wedge$
18:           Cost($x_{new}$) + $c$(Line($x_{new}, x_{near}$)) < Cost($x_{near}$) then
19:             $x_{parent}$ ← Parent($x_{near}$)
20:           $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$
21:     return G = (V,E)
```

2.3 Parameter Tuning

There are three popular method for hyper-parameter tuning: Grid search, Random Search and Bayesian Optimization.

Grid search is a traditional method to perform hyper-parameter by looping over all possible hyper-parameters and look for the best one. **Random Search** selects the combination randomly, instead of looping over all combinations to prevent high cost. However, it is not promised to find the best combination. **Bayesian Optimization** is a method that yields the global optimum for noisy functions and can be applied to hyperparameter optimization. Bayesian optimization iteratively builds a probabilistic model based on the current model, updates the model, and then is evaluated on a validation set to provide as much meaningful information as possible regarding a function (the optimum) [Wika].

2.3.1 Grid Search Cross-Validation

Grid Search is the most commonly used parameter-tuning methods. It loops through all the predefined hyper-parameters doing cross-validation. So that the best parameters from the listed hyper-parameters are selected. Then refit the models on the training set to obtain the final prediction.

The workflow (Fig.2.2) of the method is:

1. Looping through each parameter candidate
2. Randomly divide the training set into K-folds
3. For K-iterations, each fold is considered as testing set and the other K-1 folds are training set
4. Train the models and store the predictions
5. Calculate the loss and find the best parameters
6. Retain the models on all training set with the best parameters
7. Test the models on testing set for the final evaluation

2.4 Comparison of Technologies

In this section, the mentioned technologies are being compared by their advantages, disadvantages. Comparison of the machine learning algorithms are in Table 2.1 and related information are obtained from [Gup], [CFI], [Lin] and [atl]. In Table 2.2, the path planning algorithms are compared with information from [Dji], [Ata] and [Suk]. The hyper-parameter tuning methods comparison is shown in Table 2.3 with information obtained from [flo], [Zho] and [Cap].

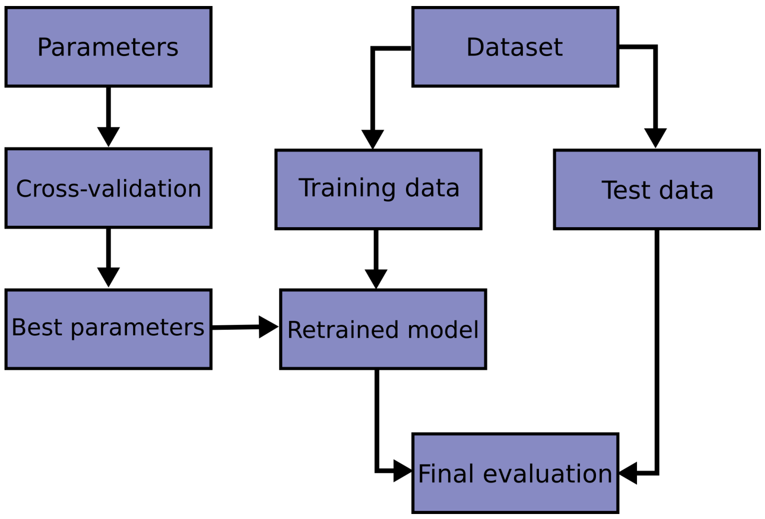


Figure 2.2: Grid Search Workflow[PVG+11]

Algorithm	Advantages	Disadvantages
Logistic Regression	Easy to understand and implement, Lower computational cost	Under-fitting, Lower accuracy
Support Vector Machines	Good for small samples, Good for non-linear classification, Performs well in higher dimension	Higher complexity, Sensitive to missing data
Random Forest	Good for multi-dimension dataset, Strong generalization ability, Good Performance on imbalanced datasets	Over-fitting when classification have higher noise, High complexity and longer training period when there are more trees, Appears as Black Box
Boosting	Easy to read and interpret, Low generalization error, High accuracy, Feature importance visualization	Sensitive to outliers, Hard to scale up, Difficult interpretation
Neural Networks	Strong nonlinear fitting ability, High accuracy, Strong robustness and fault tolerance for noise	Large number of parameters needed, Hard to interpret, Long learning period
Decision Trees	Easy to understand, Easy visualization, Normalization or scaling of data not needed	Lower accuracy, Prone to over-fitting Sensitive to data

Table 2.1: Machine Learning Algorithms Comparison.

Algorithm	Advantages	Disadvantages
A*	Morphable, Always finds the path if it exists, Heuristic,	Fail to perform with many target nodes
Floyd-Warshall	Parallel, Regular data access pattern, Predictable work load, Simple data structure	Complexity in O(n^3)
RRT	Little memory and computation requirement for sparse exploration, Heuristic Quickly finding feasible paths	Solutions require path smoothing, Restricted smoothed path to the same homotopy class

Table 2.2: Path Planning Algorithms Comparison

Methods	ML	DL	Cost	Using history information	Parallel	Suitable situation
Grid Search	Yes	No	High	No	Yes	Few parameters, Multiple models,
Random Search	Yes	Yes	Medium	No	Yes	More parameters, Multiple models, Less information
Bayesian Optimization	Yes	Yes	Low	Yes	Possible, not trivial	High training cost No derivative information

Table 2.3: Parameter Tuning Methods Comparison

3 Related Work

3.1 Machine Learning in Sports

Applying ML methods in sports can provide insights into player behavior or autonomous agent design. In the future, such insights might allow for the development of completely virtual environments that have been designed and optimized based on user-centered principles.

In general, all kinds of sports generates large volumes of random data that require automatic processing. Thus, in this section, related works related to using artificial intelligence and machine learning methods for both traditional sports and eSports are briefly described.

Logistic Regression

Logistic Regression is a classical method that uses a statistical model in its basic form uses a logistic function to model a binary dependent variable. Yun et al. analyzed the usage of logistic regression as a form of balancing gameplay mechanics based on player behavior in online games [Yun11]. The research question surrounding their analysis was considering whether or not tournament winners are actually superior to other players or if the results are based on particular flaws in the design. It is expected that similar developments could be derived from the results of this thesis work (i.e., determining some of the causes for player wins based on metrics and decisions). Prasetio et al. used Logistic Regression to predict football matches results by using only the important features obtained by certain related works rather than guessing and obtained an accuracy of 69.5% [PH16].

Support Vector Machines

Support Vector Machines (SVMs) have been used to analyze large amounts of data [JJP08]. Jiang et al. developed an SVM-based interactive gaming algorithm that can deliver instant responses (to players or other agents) in the prediction of interactive points to assist in winning. They do this by separating the mapped data from the origin to maximize a margin value based on an isolated hyper-parameter. Jain et al. used SVNs and Hybrid Fuzzy-SVM (HFSVM) to predict the outcome of basketball games outcome and having an average 5-fold cross-validation results of HFSVM obtained 88% accuracy with average 391 supported vectors and 86% and 547 supported vectors for SVM [JK17].

Random Forests

Random Forest (RF) is an ensemble learning method for classification, regression, and other tasks based on a multitude of decision trees [Wikd]. It has been used to predict the win probability of NFL games having the mean squared error less than 0.2 for all 4 quarters of the test data [LN14]. Thenmozhi et al. use several machine learning algorithms to predict the results of the sport Cricket with a result that Random Forest performs the best out of all the algorithms they applied with the accuracy mostly over 70% [TMJ+19]. Ani et al. focused on ensemble methods including Random Forest to predict the eSport game, League of Legends, using pre-match and within-match features, with Random Forest having very good accuracy in both pre-match (95.52%) and in-match (98.18%) predictions and a combined accuracy of 99.75% [AHDD19]. Oughali et al. predicted the shooting results of NBA players and RF gave an accuracy of 57% [OBE19]. Johansson et al. compare RF and SVM with RF having a best accuracy value of 88.83% in predicting Dota2 results while SVM failed to perform [FJ15]. Pretorius et al. compared the ability of human prediction and artificially intelligent prediction system, with the results that RF outperforms the human prediciton systems (89.58% vs 85.42%) [PP16].

Boosting

Boosting is an ensemble method used to improve the existing model by training the weak learners and correcting the predecessors to a stronger learner. In [AHDD19], beside Random Forest, Ani et al. also used AdaBoost, Gradient Boosting, and Extreme Gradient Boosting, they all archived a combined accuracy around 97% and more than 96% for the prediction of within-match. However, both of the Gradient Boosting only have 65% for the pre-match predictions and AdaBoost only 57%. In the basketball field, however, Oughali et al. used Grid Search for Extreme Gradient Boosting and increased the prediction result by 8% (from 60% to 68%) and it performs better with 11% higher accuracy than RF [OBE19]. Within all existing boosting framework, Light Gradient Boosted Machine is shown to have better performance on both efficiency and accuracy with lower memory consumption [LMN+17]. Hodge et al. used LGBM as one of the methods to predict Dota2 match results and having an highest accuracy of 77.46% [HDS+19].

Neural Networks

Interestingly enough, *Artificial Neural Networks* (ANN) gained attention throughout various industries after 2012 when *Convolutional Neural Networks* (CNNs) were used to accurately label images and voice recognition (and the developers went on to create DeepMind). Shortly after in 2014, Clark et al. used CNNs to beat expert players at a famously challenging game called Go [CS14]. While the game can be logically considered "trivial" due to the limited number of rules, complexity arises when evaluating the state-space and the results of actions of future outcomes. CNN's have proven very effective at predicting (and pruning) states that would lead to poor results in many games. At the current time, neural networks are being used to train autonomous agents to beat human players in StarCraft and other *Massive Multiplayer Online* (MMO) games based on player behavior in real-time. Björklund et al. chose a feed-forward neural network to predict the outcome of CS:GO games with a 6% higher accuracy (65.11%) than the benchmark prediction (58.97%) [A.B17]. Katona et al. predict the death of Dota2 players within a 5-second window with a large number of features also by a deep, feed-forward neural network, and achieved an average precision of 0.5447 [KSH+19]. Lin et al use CNNs to forecast the NBA (National Basketball Association) game results and achieved an accuracy close to 80% [LCC18b]. McCabe et al. covered several major league sports and used multi-layer perceptron (MLP) and back-propagation to predict the outcome and the best accuracy out of all testing was 75.4% for Super Rugby [MT08].

Decision Trees

Decision Trees have been used to create video games since 1956, in which a Minimax tree for Tic-Tac-Toe was implemented [Jon]. Throughout the early 90s, *rule-based* decision trees and *Finite State Machines* (FSM) were combined in the realm of video games for simple AIs. Decision trees encode decisions to consequences. For example, if a player takes an action, a *Non-Player Character* (NPC) will have a *reactive* behavior depending on the rules and state of the player. As the state space has continued to grow with many potential inputs and stochastic behavior, decision trees have become nearly obsolete in modeling or predicting complex behavior in real-time. An example of the complexity involved in current video games is demonstrated in Fig. 3.1. Tang et al. [TLL+18] applied Decision Trees on Chinese football matches to predict the winner. With a few steps of extracting features, the highest accuracy obtained was 57.7%.

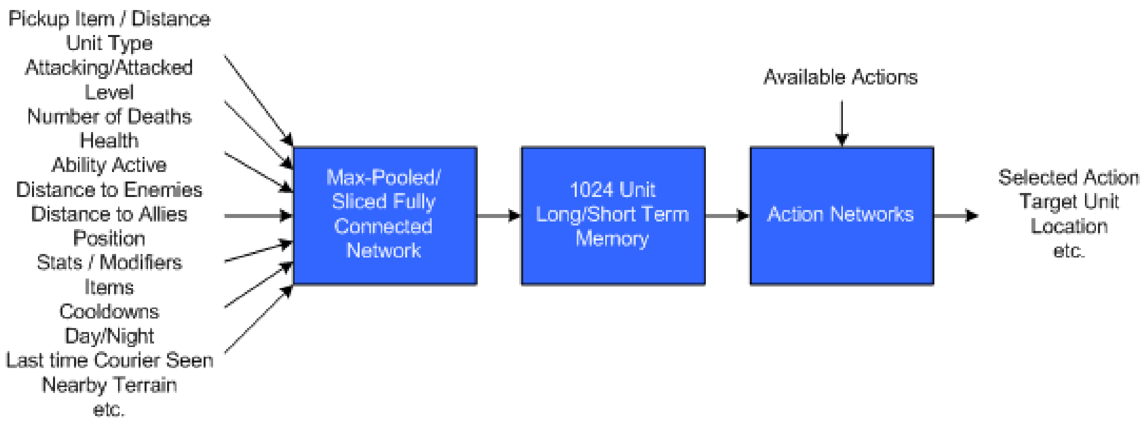


Figure 3.1: Example of Dota 2 hero AI for long horizon learning [Jon].

Others

Yue et al. used a latent factor modeling approach for in-game basketball event prediction and the validation shows that the results are rather accurate [YLC+14]. Dubbs used only in-game scores to predict multiple sports game results, and the predictions only for basketball comes very close to and sometimes outperforms the near-optimal indicator since basketball games have a longer season to have a pre-knowledge about which teams are better [Dub18].

3.2 Counter-Strike Games Specific

To predict the game-winning results, Björklund et al used k-means cluster to decide player roles and parsed player kills, deaths, weapons, location and type of kills, money situation when kills happen, Grenade usage and players active area and position related to others for different roles to predict the final game results using ANNs [A.B17]. Also, Hladky et al. used hidden semi-Markov models to predict opponent positions [HB08].

For predicting round-winning results, Rizani et al. used a *Generalization Ratio* GR as a metric, indicating that the whole game results depend on the players' skills but the result for each round is stochastic and unpredictable [RI18]. However, Makarov et al. used both Logistic Regression and Decision Trees to predict the round winner after the bomb was planted using features: number of players alive, equipment cost, player health, different grenades numbers and a team rating score obtained by a Bayesian skill rating system TrueSkill. The results are acceptable for this work with a highest accuracy of 62% and a lowest Logloss of 0.675 [IMI].

4 Methodology

This section describes the methodology for the thesis. To solve the problem, on a high level, these processes include extracting map information, parsing data from HLTV demonstration files, finding the shortest path (from CS:GO game coordinates), data pre-processing, and modelling.

4.1 Structure

There are five major processes and one supporting process (collecting in-game coordinates) in this thesis (Fig. 4.1). The necessary game stats are obtained from the Data Parser, the related WGS84 positional data are coming from the Map Info Extraction process, which is used together with the positional data obtained from the game itself to perform the Shortest Path Planning algorithms. Before modelling, all necessary data including the results of Path Planning are all been pre-processed for a better result.

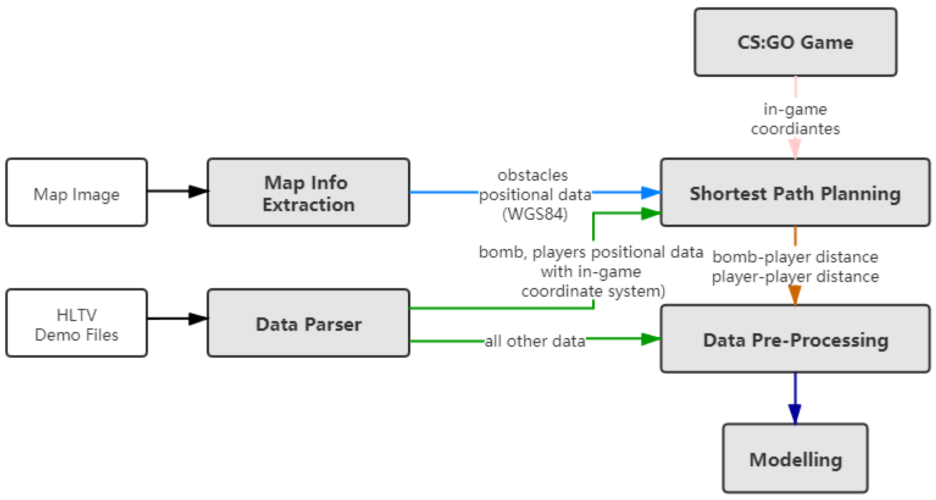


Figure 4.1: Structure

Data Parser and **Data Pre-Processing** processes are explained in Sec. 4.2 for a general explanation of how the data are prepared while **Shortest Path Planning** is explained separately in Sec. 4.3 with **Map Info Extraction** for a clearer explanation about the algorithms and **Modelling** is explained in Sec. 4.4.

4.2 Data Preparation

4.2.1 Data Parsing

To predict the round winning probability, it is important to decide which features are going to be used for training. The sampling rate is every 5 seconds for the 40-second interval, beginning from the bomb being planted until the bomb expires. The input features are listed as follows and will be explained more in the following subsections:

- Round Winner
- Seconds after bomb planted
- Average Bomb-Player distance for T-side
- Average Bomb-Player distance for CT-side
- Team total Weapon Cost for T-side
- Team total Weapon Cost for CT-side
- Average Player-Player distance of both sides
- Ratio of Alive Players further than threshold distance for T-side
- Ratio of Alive Players further than threshold distance for CT-side
- Number of High-Performing Players for T-side
- Number of High-Performing Players for CT-side
- Number of Alive Players category
- Available Flash Bang numbers category
- Available Molotov (Grenade) numbers category
- Available Smoke Grenade numbers category

To obtain the stated features, the: round winner, current time, player positions, bomb position, player kills, player weapon values, player alive status, and grenade numbers are parsed from 142 historical games for the stated sampling rate with a total of 10274 data samples.

4.2.2 General Features

In general, the number of living players for each side decides the basic situation of the game. For example, there is a lower chance of a team winning with a 1:5 ratio (living players) compared to a 3:5 ratio, which implies a higher likelihood of winning (i.e., recovery). Naturally, a 5:5 ratio of living players is the most balanced, leading to the number of living players being less of an indication for the match direction until a player dies.

A feature of players that has a substantial contribution is the number of their kills. A higher value indicates how well a player is at their role and teams that have better players are more likely to win. Weapons are a factor in kills and have a large influence on the end results. For example, a team equipped with AK47s has a larger probability of winning than a team only equipped with knives. Weapons have a numerical value based on their effectiveness (i.e. due to range, power) with higher values representing better weapons. This value is used rather than the weapon's name for quantitative purposes. Grenades are in the category of sub-weapons and the number of grenades for both teams was also considered in the weapon category.

4.2.3 Positional Features

In addition to the general features, positional features are an important factor for the results of this thesis. The only win-lose results that are analyzed are based on CT-side players to wining by defusing the bomb placed by T-side players. In alternative, T-side players need to eliminate as many CT players as possible to prevent them from disarming the bomb. With these two aspects in mind, the distances between the players to the bomb are a significant factor in the end result. Furthermore, players need to eliminate enemies from the opposing team. As such, the distance between players is a necessary factor since target eliminations influence the match result. However, using each distance value as individual features requires a greater amount of time to find the best feature combination, making the process non-ideal for the training process. Therefore, data manipulation was pursued to improve training time.

Instead of every player's individual distance to the bomb, the average bomb-player distance of both sides was used. Also, a ratio of players who are further than a certain distance (set to 200 in this thesis) to the total alive players of both sides was also used as a feature to see if the majority alive players are close to the bomb. Instead of every player-player distance, the average player-player distance is used. Furthermore, to find out if one player has a better chance to eliminate the opponent player, another feature is the number of high-performing players who are close to the opponent players (threshold set to 200 in this thesis) of both sides. High-performing players are defined as having more kills than current average kills (i.e., high-performing players are better at the game).

4.2.4 Categorical Features

To reduce the number of features for faster feature importance detection, rather than individual features for both sides, the alive player numbers, three different types of grenades numbers are combined together as categorical features input, for example, CT_1_T_3 for CT-side 1 alive player and T-side 3 alive players.

All features except the categorical features are numerical (round winner is set as 0 for CT, 1 for T). Thus, it is optimal having numerical values for the categorical features, too.

To obtain the estimated values, the impact of each categorical data distribution situation on the round winner should be estimated. So, the mean value of round winner (0 for CT, 1 for T) of all data sample is calculated as $mean_{all}$ and the mean values of the round winner of the data sample that matches each unique category value are calculated as $mean_{category\_value}$, for instance, $mean_{playerAlive\_CT\_1\_T\_3}$.

Then the weight of the sample data of certain category value is needed (if the category situation happens rarely, it should have a smaller weight). The sample size of the category value (N) and a factor value are applied in Equation 4.1 to get the weight. The weight value lies in 0-1 and the equation is obtained by trial and error. A lower factor value will result in less sample size on categorical transformation weighting higher. In this thesis, the factor value is set to 150.

Finally, the estimated values can be obtained by Equation 4.2 with the weight (W) of the category value and the values are used instead of real categorical data.

$$Weight = \left(\frac{1}{1 + 10^{-\frac{N}{factor}}} - 0.5\right) \cdot 2 \tag{4.1}$$

$$EstimatedValue = mean_{all} \cdot (1 - W) + W \cdot mean_{category\_value} \tag{4.2}$$

4.2.5 Pre-Processing

Only a small amount of pre-processing needed in this thesis, since the data have been manipulated already in the feature generating subsections.

In this part, firstly, all samples that have a null value should be dropped. Also, it is important to make sure all the features are standardized since they are normally a common requirement for many machine learning estimators: they might not have optimal behavior if the individual features are not in a standard normal distribution (e.g. Gaussian with 0 mean and unit variance) [PVG+11].

4.3 Shortest Path Planning

To obtain the distances stated in Sec.4.2.3, simply using the Euclidean distance is not sufficient since players do not move with linear trajectories (straight lines), and not every point on the map is accessible. Thus, some path planning algorithms are used to calculate the distances.

4.3.1 Map Info Extraction

To perform path planning, the necessary coordinates need to be extracted, which are start point, goal point, and obstacles. In this thesis, start and goal points are bomb positions or player positions. All coordinates are parsed from the game replay file mentioned in Sec.4.2.1. However, the positions of the obstacles are not obtainable from the replay files directly.

To solve this problem, a geographic information system (GIS) application is used to obtain the positions of the obstacles (in WGS84) since it is designed to process position related data. However, the in-game positions are not under the WGS84 coordinate system. With knowing the map only has a different scale and translation in two systems, the following equation is used to get transformed coordinates, where a indicates how much the scale of the in-game coordinates from WGS84 and b indicates the translation.

$$x_{wgs84} = a \cdot x_{game} + b \tag{4.3}$$

Instead of exact coordinates of the obstacles, a grid of the map is created and the positions of the centroids of the grid cells are used as the obstacles coordinates if over 50% of the cell is filled with obstacle objects. In this case, although there are still minor errors from the transformation due to the reason that the WGS84 coordinate system is not flat, the errors can be ignored.

To have an acceptable result, the grid size is an important factor. It should not be too large or too small so that the result will not end up with low accuracy or too much time consumption. In this thesis, since the map area is not huge, it is better to have a relatively smaller size for more accurate planning results. So, after analyzing the in-game coordinates and the map, one step of the player movement (roughly 10 units on the map) is set as the grid size.

4.3.2 Algorithm Selection

Since path planning and identifying the optimal path is not the primary goal for this thesis, and for each data sample, 35 distances (10 player-bomb distances and 25 player-player distances) were calculated. That is, rather than absolute accuracy in deriving an optimal path, the time complexity is a more important factor in choosing the right path planning algorithm. For this goal, approximating an optimal path is satisfactory without impacting the results of the match prediction.

In this case, the more "basic" algorithms, which include A* and the Floyd-Warshall algorithm, were used for minimizing time complexity. On the other hand, a more complex algorithm (RRT*) was also utilized for higher accuracy. The inclusion of basic and complex algorithms is to analyze trade-offs between accuracy and time complexity. The algorithms are introduced in 2.2.

However, these algorithms cannot find the best path in every situation due to different types of errors. For instance, due to the grid granularity, players may be considered staying in a not reachable grid if they hide right behind the wall. Effectively, the algorithm considers the player as a "part of the wall." In this case, the **Manhattan Distance** (Equation 4.4) is used for distance calculation when the algorithms fail to perform correctly.

$$Distance_{Manhattan} = |x_1 - x_2| + |y_1 - y_2| \tag{4.4}$$

4.4 Modelling

From all the methods mentioned before, Logistic Regression, Random Forest, and Light Gradient Boosting Machine are chosen to to the modeling. It is due to that Logistic Regression is very easy to understand and have lower cost, Random Forest performs the best in all papers mentioned (in [AHDD19] RF performed a combined accuracy of 99.75%), and according to [LMN+17], Light Gradient Boosting Machine is shown to outperform on all boosting algorithms when boosting in general is known to have higher accuracy. According to Table 2.1, Neural Networks is not used in this thesis for the reason that it has a longer training period and need relatively more parameters, SVMs are not considered for the high complexity reason, and since two tree-based algorithms are used and Decision Trees have relatively lower accuracy, it also not used in this thesis.

For better performance, the features stated in Section 4.2.1 are been selected by looping through all feature combinations as input and then trained for all three algorithms with library default parameters to find the best feature combinations for the algorithms and the prediction results.

The dataset is divided into a training set (70%) and a testing set (30%) by the parsed time to prevent over-fitting. Accuracy, Logloss, and F1-Score are used as scoring metrics to evaluate the prediction results of the testing set. A detailed explanation of the metrics are in Sec.6.1

4.4.1 Parameter Tuning

Hyper-Parameters for models

Hyper-parameters are parameters that are not directly learned within estimators [PVG+11]. For the different models, different hyper-parameters are applied.

For RF, the number of trees and the maximum depth is important. In theory, the more trees are used, the better is the learning result. However, a huge number of trees increase training time. The maximum depth indicates the maximum depth of each tree. With deeper trees, the model can convey more information from the data since there are more splits, but the trade-off here is also the training time.

For LR, it is essential to have the penalty and solver defined. The penalty prevents the model from over-fitting by limiting the complexity of the model in order to keep the complexity and performance in balance. L1 and L2 are two commonly used regularization methods for the penalty parameter. The key difference between them is that L1 calculates the absolute value of magnitude while L2 measures the squared magnitude. For different choice of penalty, the solver is different, *liblinear* solver can be used for both L1 and L2 while *newton-cg*, *lbfgs*, *sag* and *saga* can only be applied on L2.

For LGBM, there are a lot of parameters for different aspects, for example, *bagging fraction* for faster training and *learning rate* for higher accuracy. Since LGBM already performs relatively faster, in this thesis the main focus will be on the accuracy. For also the tree-based LGBM model, the number of trees, and the maximum depth are also important. Additionally, the learning rate and the number of leaves make a large impact on the accuracy, too. However, leaves numbers too high will cause over-fitting.

Thus, to improve the prediction results, hyper-parameter tuning is commonly required to find a balance between better performance and the trade-offs and the traditional parameter tuning method Grid Search is utilized since there are not a lot of parameter combinations. At last, the retrained results of three models are compared with the results obtained in Section 4.4.

5 Implementation

This section describes the implementation for the thesis, including the working environment and detailed explanation of each steps and the example implementation results.

5.1 Process

The more detailed processes of the whole thesis is shown in Fig. 5.1 with all the steps of the implementation. Implementation steps for **Data Parser** and **QGIS** are explained in Sec. 5.3, steps for **Shortest Path Planning** and acquiring **In-Game Reference Points** are explained in Sec. 5.4 and **Pre-Processing** in Sec. 5.5, **Modelling** in Sec. 5.6.

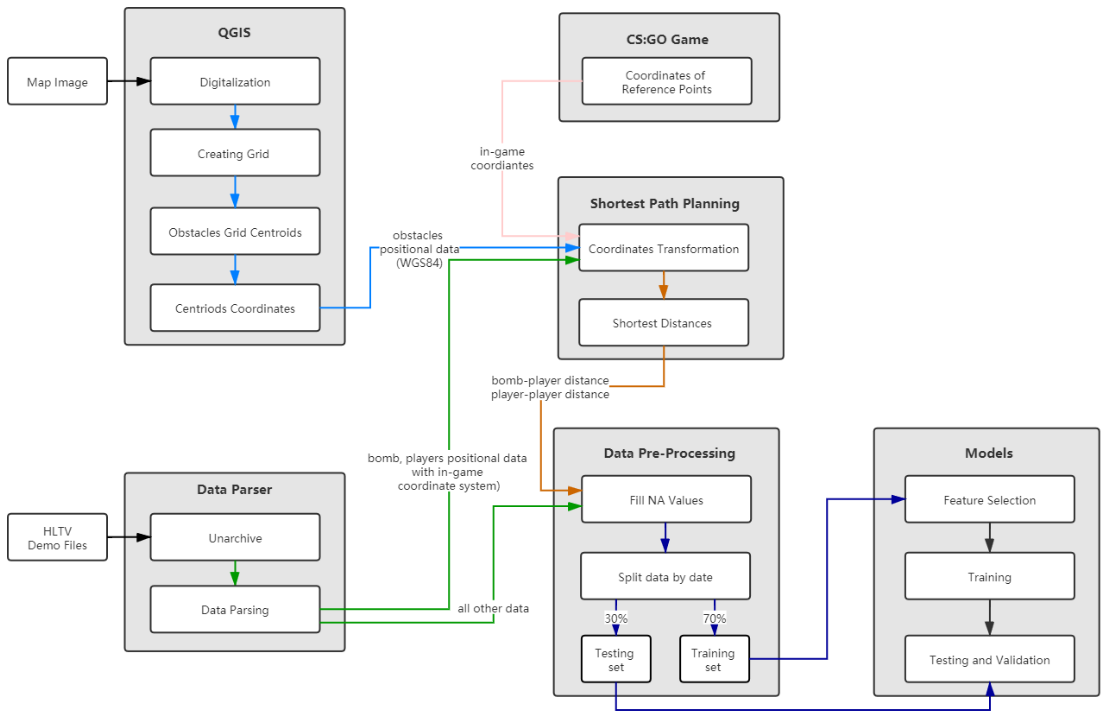


Figure 5.1: Process

5.2 Environment and Programming Language

The implementation was completed using a 64-bit Windows 10 operating system device with the following software and libraries:

Software:

- JetBrains GoLand 2019.3
- PyCharm Community Edition 2019.3.2
- QGIS3

Libraries:

- LightGBM (`https://lightgbm.readthedocs.io/en/latest/`)
- NumPy (`https://numpy.org/`)
- Pandas (`https://pandas.pydata.org/`)
- scikit-learn (`https://scikit-learn.org/stable/`)
- SciPy (`https://scipy.org/scipylib/`
- `https://github.com/mholt/archiver`
- `https://github.com/markus-wa/demoinfocs-golang`
- `https://github.com/AtsushiSakai/PythonRobotics/tree/master/PathPlanning`

There are two programming languages are used in this thesis.

Go is utilized for data parsing, for the reason that Go is a very fast language, considering the amount of data being parsed, slower language is not considered. Besides, Go is relatively easy to learn and use.

The pre-processing and modelling parts are done with Python. Although Python is slower in computation, it has its own advantages. It is very easy to use and develop, has all kinds of libraries, especially for machine learning related works.

For path planning algorithms, although it will be faster to use Go to do the calculation, the distances need to be used as features in the training process. So, Python is used for this part, too.

5.3 Data Extraction

To complete the prediction, the chosen game stats as well as the map information are essential. Software Goland is used to parse the game stats using the programming language Go and QGIS is used to obtain the related map information. 10274 round samples from 142 games are extracted.

5.3.1 Game Stats

The game stats are parsed from the replay demo files, which are obtained from GRID eSports (originally downloaded from HLTV) in RAR format. To unarchive the files for data extraction, an open-source archiver (`https://github.com/mholt/archiver`) has been used. Then the unarchived files are used as input to get game stats using an open-source parser (`https://github.com/markus-wa/demoinfocs-golang`). These following raw data are extracted from the unarchived replay files:

For current game status:

- map name
- round number
- winner side
- bomb position
- seconds after the bomb is planted
- number of alive player for both sides

For each alive player:

- name
- side
- position
- current active weapon
- current weapon value
- current money left
- velocity
- kills
- grenades numbers

Beside unarchiving, the implementation contains three major parts: register events, get required event snapshot data, and output.

When a event is registered, the data of the registered moment are obtained by a snapshot function, then all the needed information is stored in maps in the following format and output:

Listing 5.1: Data Storage Format

```
    map[roundNumber]map[timeAfterPlant]map[playerName]statType
```

Note: the data type for first three map are int, time.Duration and string, the name above is for showing storage format. The last value can be different type of data depending on the information type.

Snapshot is the function for getting all the necessary data at the very exact moment. When an event is registered, all data at the moment of the registered event can be obtained in the parser.GameState() and the data are stored in the the type of map[*playerName*]*statType* in List 5.1. For example code, see the Appendix List 1.

To get data of a certain time during the game, three events are registered:

BombPlanted, registered when the bomb is planted for the round. The position of the bomb can be easily obtained. The current time is set to lastPlantTime and lastSnapshotTime for further comparison. All information needed are obtained by function Snapshot as stated before and store in the the type of map[*timeAfterPlant*]map[*playerName*]*statType* in List 5.1 with timeAfterPlant is 0 meaning bomb is just planted. For example code, see Appendix List 2.

FrameDone, registered when each frame of the game is done. To get data every 5 seconds after the bomb being planted, this event is needed to check if the current time is the required time. This is where the lastPlantTime and lastSnapshotTime are needed. When bomb is planted and the time is required time, the data can be obtained from Snapshot function as stated before and store in the the type of map[*timeAfterPlant*]map[*playerName*]*statType* in List 5.1 with timeAfterPlant set to how many seconds after the bomb is planted. Example code see Appendix List 3.

RoundEnd, registered when the current round ends. Mainly for getting how many rounds are played and store all the obtained data in the format of map[*roundNumber*]map [*timeAfterPlant*]map[*playerName*]*statType* in List 5.1. Example code see Appendix List 4.

To output the data in JSON, three types are created:

- **RoundInfo** contains basic round information: mapName, roundNumber, winnderSide, bombPosition and a list of PostPlantStatus.
- **PostPlantStatus** contains post bomb plant information: how many seconds after the plant, alive players of both sides, and a list of AlivePlayers.
- **AlivePlayers** contains all necessary data of the alive players.

Listing 5.2: Example Game Stats Output in JSON

```
{
  "mapName":"de_mirage",
  "num":1,
  "winnerSide":"T",
  "bombPosition":[1877.96875,692.96875,160.03125],
  "postPlantStatus":[
    {
        "second":0,
        "TsideAlive":1,
        "CTsideAlive":0,
        "alivePlayers":[
        {
            "Name":"RpK",
            "Side":"T",
            "Position":[2610.46044921,−137.0937042236,138.0312],
            "Weapon":2,
            "WeaponValue":1250,
            "Money":450,
            "Velocity":[−3.2613320350647,−16.569658279419,0],
            "Kills":1,
            ...
        },
    },
    ...
```

5.3.2 Map Information

To perform the Shortest Path Planning, it is necessary to have the coordinates of the obstacles on the map. In this thesis, QGIS is used for extracting coordinates of the obstacles and no-go areas on the map.

To obtain the coordinates, the features firstly need to be digitalized. Based on the map image (Figure 1.1), three different shapefiles are created for reachable areas (blue), obstacles (yellow) and no-go areas (white), see Fig. 5.2.

To reduce the size of the graph, a 72×84 grid map is created based on the Minimum Bounding Box of the map instead of adding every point on the map to the graph. Grid size is chosen to be 10 due to the actual gaming experience to represent the step size of the players. See Fig. 5.3, in real game, the width can contain approximately 3 steps.

With the help of QGIS Geoprocessing Tool- UNION and INTERSECTION, the layers of no-go areas and obstacles are united, then the new layer is intersected with the grid layer to obtain the new grid layer of all the areas that players are not allowed to go. Finally, in Fig. 5.4, the central points of all grid are extracted with QGIS Geometry Tool - CENTROIDS, together with the coordinates of the centroids under the WGS84 coordinate system.

The coordinates of all 3593 centroids are stored in text file:

Listing 5.3: example coordinates in text file

```
112.282571991937,−802.528426890445,5
112.282571991937,−792.528426890445,5
112.282571991937,−822.528426890445,5
112.282571991937,−812.528426890445,5
112.282571991937,−842.528426890445,5
...
```

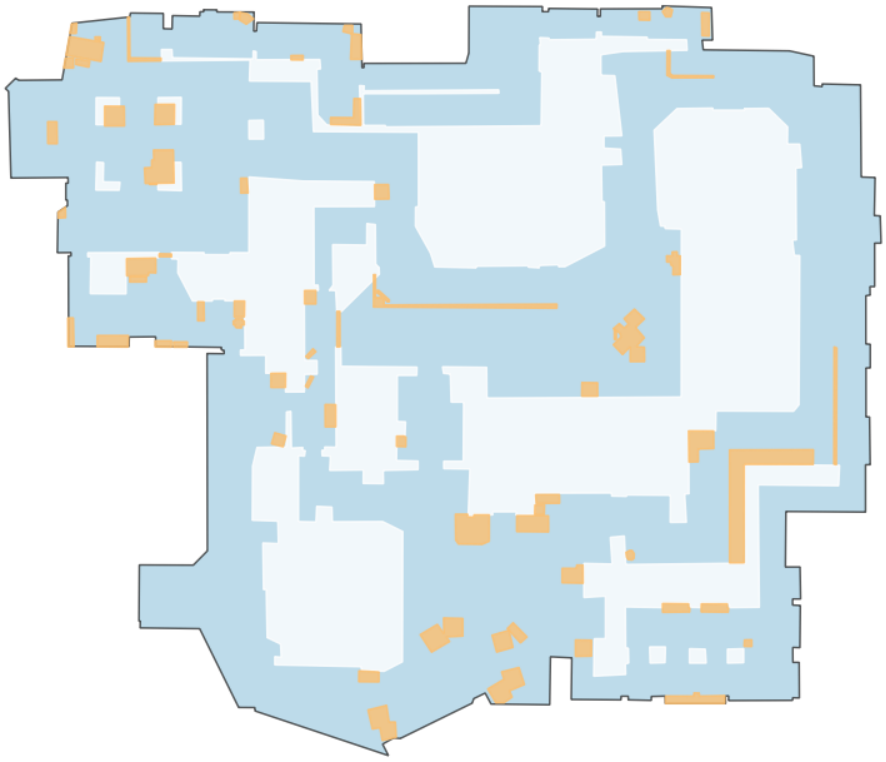


Figure 5.2: Digitalized Map

5.4 Shortest Path Planning

With a different coordinate system, the coordinates obtained from the demo file need to be transformed to the WGS84 system to perform path planning algorithms. Equation 4.3 explained in Section 4.3.1 is used to get transformed coordinates.

Four point pairs (Fig. 5.6) are chosen for the transformation. The in-game coordinate points are obtained by using the command "cl_showpos 1" in the game console, then go in-game mode and move to the points shown in Fig. 5.6, the coordinates of current positions are shown on the screen (Fig. 5.5, both location and coordinates are in the left up corner). The coordinates obtained here should be the same as the ones in the demo files.

With obtained coordinates, by applying two point pairs in the equation, the coefficients a, b can be calculated. Then, two other pairs are used for testing the results. If both the difference between $x_{calculated}$ and x_{real} are less than 10 (grid size), then the two coefficients can be used for further transformation calculation.

Three algorithms are used in this section: Floyd-Warshall, A*, and RRT*.

Floyd-Warshall algorithm is implemented in the SciPy library. Before applying the Floyd-Warshall algorithm, the distance matrix should be generated with a size of 6048 (the number of the total centroids). To obtain the distance matrix, a 72×84 obstacle

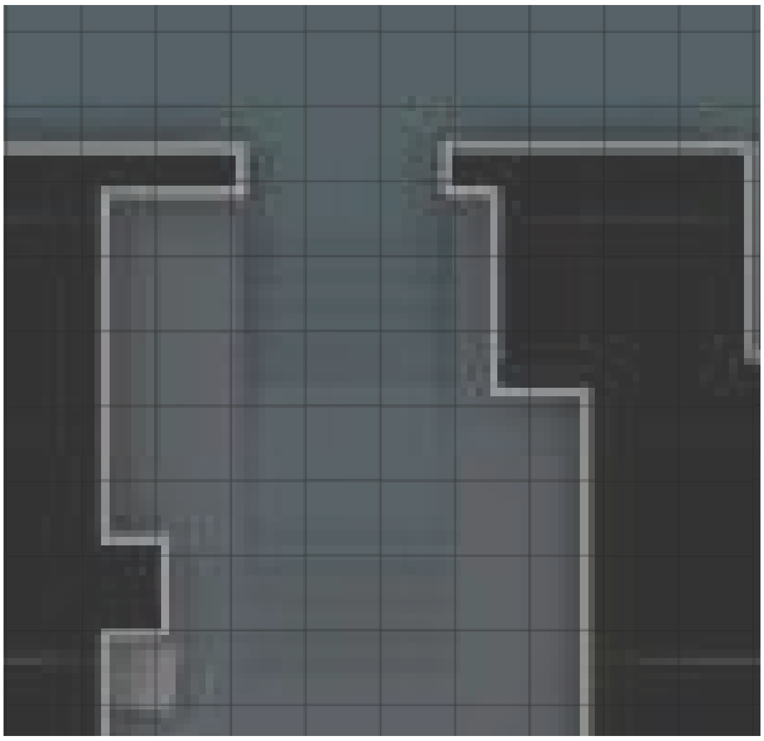


Figure 5.3: Stairs in the map under grid size 10

matrix is generated with the values are only TRUE and FALSE indicating if the grid cell is an obstacle cell from the obstacles text file obtained in Sec. 5.3.2. Elements on the distance matrix diagonal are set to 0, all other elements of the distance matrix are set to 10 (reachable) or 999999 (not reachable) according to the obstacle matrix.

It takes the algorithm approximately four minutes (plus or minus ten seconds) to update all elements with the shortest distance between corresponding points.

A* and RRT* are both implemented in an open-source GitHub project (`https://github.com/AtsushiSakai/PythonRobotics/tree/master/PathPlanning`). To perform path planning, the obstacles need to be stored in a Python list. Player motion is set as 1 and the moving radius is set to the size of the grid. Then, with the size of the grid, transformed start, and goal coordinates, the shortest path will be found and the cost of the path is used as the distance between the start and goal points.

Due to the size of data, the time consumption is the major consideration for the path planning algorithm. RRT* fails to generate one path within 10 minutes while A* finishes all the calculations for more than 10000 samples within 12 minutes (Example A* planning result in Fig. 5.7). So, only distances obtained by Floyd-Warshall and A* are used in the training process.

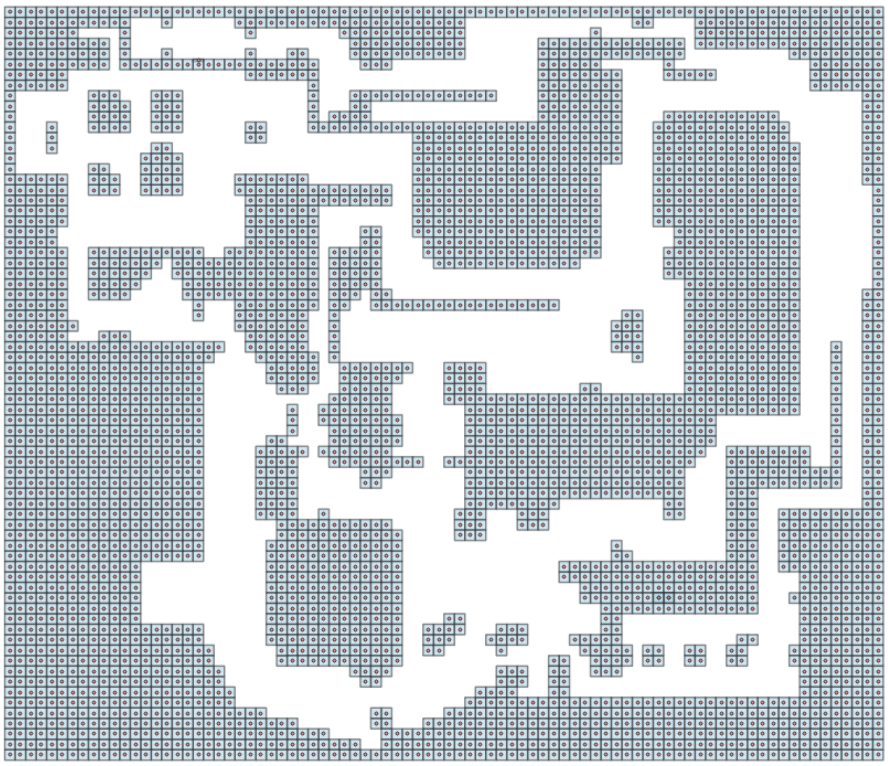
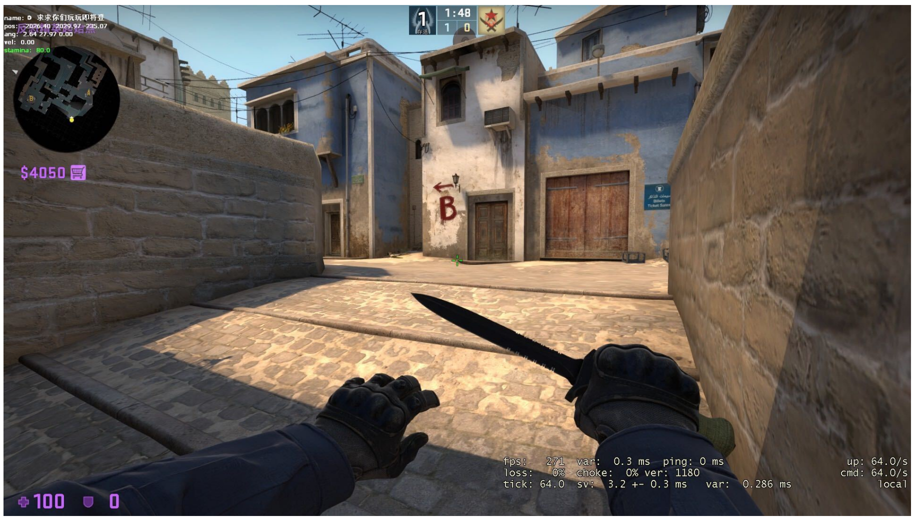


Figure 5.4: Grid Centroids of obstacles and no-go area



```
name: 求求你们玩玩即将登
pos: -2026.40 -2029.97 -235.07
ang: 2.64 27.97 0.00
vel: 0.00
stamina: 80.0
```

```
fps:    271  var:  0.3 ms  ping: 0 ms                     up: 64.0/s
loss:    0%  choke:  0% ver: 1180                        cmd: 64.0/s
tick: 64.0  sv:  3.2 +- 0.3 ms   var:  0.286 ms              local
```

Figure 5.5: Example in-game point position

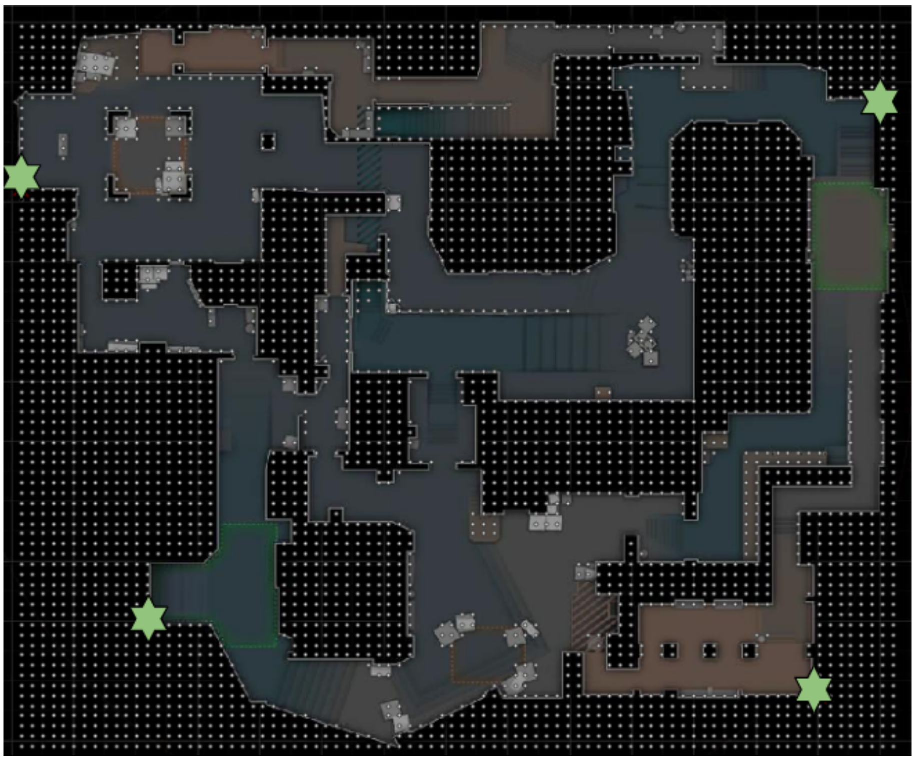


Figure 5.6: Points Pairs Position

5.5 Pre-Processing

For the features needed for training, the extracted data need to be processed.

First, instead of the actual winner side, 0 is used for CT and 1 is used for T to avoid categorical data.

To get the average value of the distance, all the N/A distance values are replaced with 0. A threshold value 200 is set for the distance between player and player or player and bomb to get the ratio of the players who are too far from the bomb or opponent team players. For category features, for example, player alive numbers, they are saved in the column for both sides in the format *"CT_number_T_number"*. Other features are calculated using simple calculation, for example, total team weapon value for both sides are simply just obtained by adding up all alive players weapon value for each side.

After all the calculation, the rows still have NA values are dropped, and the data are split by parsed timestamp to make sure the training dataset and testing dataset are not related.

At last, sklearn.preprocessing.StandardScaler() is used on both training and testing set for standardization for the better prediction performance.

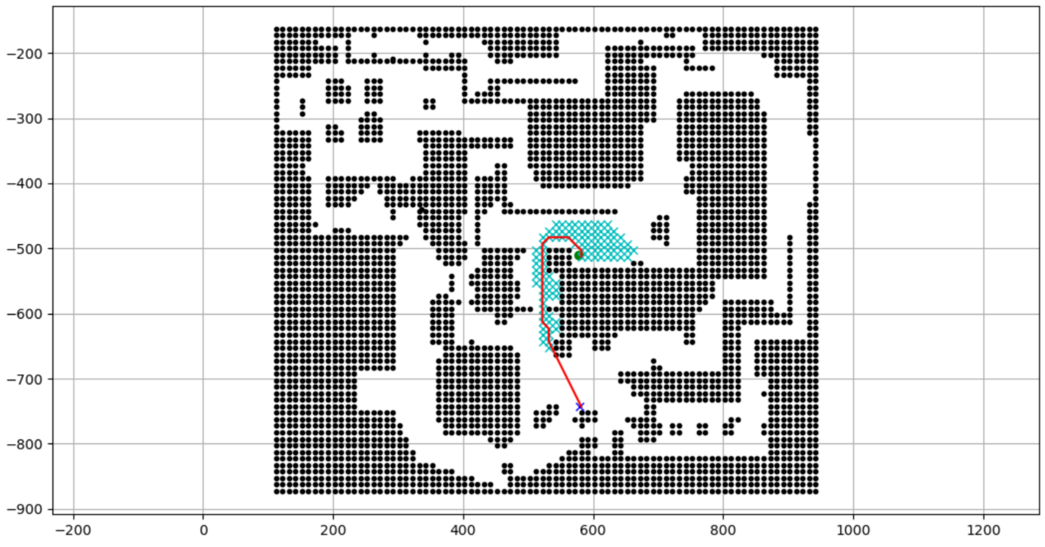


Figure 5.7: Example A* Planning Result

5.6 Modelling

5.6.1 Feature Selection and Training

Before using the extracted features in the training process, it is important to decide which features are actually useful for predicting the results.

To achieve the feature importance detection, all standardized features are combined with other standardized features as different combinations. For example, three features [A, B, C] have seven different combinations [A], [B], [C], [A, B], [A, C], [B, C], [A, B, C]. With all the input features, there are 114688 unique feature combinations.

For both A*-distance and Floyd-Warshall-distance datasets, all possible combinations of features are iterated over in training for RF, LR, and LGBM separately, the feature combinations having the smallest Logloss value are considered as the best feature combinations for the corresponding ML algorithm and those features are used for further optimization. The factor value stated in Sec. 4.2.4 is set to 150 based on row numbers of each categorical feature.

With low computational cost, LR successfully completed the training process within 22 minutes with the selected feature combinations and the evaluation metrics while both RF and LGBM failed to process all the 114688 combinations within 8 hours. Therefore, another method of feature selection is applied to RF and LGBM.

Since both models can give the feature importance during the training process, the automatic looping over the feature combinations are done manually instead. First, all features are used to train the model and the feature importance are printed out. Then, retrain the model with the least important features removed. Repeat the step before and compare all the metrics until there are no more improvements. At last, the final selected features are used in further optimization.

5.6.2 Parameter Tuning

Hyper-parameters play really important roles in the training process, suitable hyper-parameters can improve the training results and save training time. For hyper-parameters for different models stated in Sec. 4.4.1, the default values are set as Table 5.1. To find the best hyperparameters for the three models, all the parameter candidates are looped by grid search cross validation implemented by scikit-learn. Logloss is used as the scoring metric and 5-fold-cross-validation is applied.

After looping over all the parameters, the models are retained with the best parameters. Then, the best model parameters and the importance of each feature are printed out.

Model	n_trees	max_depth	penalty	solver	learning_rate	n_leaves
RF	40, 100, 150, 250	3, 5, 10, 20				
LR			l1, l2	liblinear		
LGBM	40, 100, 150, 250	3, 5, 10, 20			0.05, 0.1, 0.25	30, 50, 80

Table 5.1: Default Parameters for Models

5.6.3 Testing and Evaluation

With the best parameters, Accuracy, Logloss, and F1-Score of the testing dataset predictions are used to find out how good the models perform and find the best model for the situation. All of the metrics can be calculated using scikit-learn. In this thesis, the F1-Score is needed for both of the class prediction but the scikit-learn function only calculates the F1-Score for one class. So, the Confusion Matrix is used to obtain the necessary values to calculate F1-Score for both classes.

6 Evaluation

6.1 Scoring Metrics

To evaluate the prediction results, choosing the scoring metrics is important. This section describes the three metrics are used to evaluate the prediction results.

6.1.1 Accuracy

The most common metric is **Accuracy**, it measures how many samples are predicted correctly with the actual results. In binary classification, there are positive and negative classes, in this case 1 stands for T-side (the positive class) and 0 stands for CT (the negative class). There are four different situations for the prediction results: true-positive (TP), false-positive (FP), true-negative (TN), false-negative (FN), respectively means correctly classify positive class, incorrectly classify positive class, correctly classify negative class and incorrectly classify negative class. Both true-positive (TP) and true-negative (TN) are considered as accurate, so the accuracy can be calculated in Equation 6.1).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.1}$$

But with only accuracy cannot represent the performance of the model in the case that the samples are not formed with roughly equal amount of positive and negative data. For example, one model has 90% accuracy for predicting T-side winning and only 10% for CT winning, it will still show high accuracy if most winner side of the data is T.

To prevent this from happening, Logloss and F1-Score are introduced in the thesis to evaluate the model too.

6.1.2 Logloss

Logloss measures how much the prediction varies from the actual result without considering positive class or negative class. The value of Logloss lies between 0 and 1, the smaller the value is, the better the model is. Logloss is calculated in Equation 6.2, with N is the number of samples, y indicates if the sample is correctly classified using 0 or 1, p is the model's predicted probability.

$$\mathtt{Logloss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i) \tag{6.2}$$

6.1.3 F1-Score

The **F1-Score** (Equation 6.3) is the harmonic mean of the **Precision** (Equation 6.4) and **Recall** (Equation 6.5). It is noticeable from the equation that F1-Score focuses mainly on correctness of true-positive classification, but in this case, both T-side and CT-side winning prediction are important to the result. Hence, they are separately been used as "positive" to calculate the corresponding F1-Score.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.3}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.5}$$

6.2 Feature Selection

After running through all possible 114688 feature combinations for LR and manual feature selection for RF and LGBM with distance data calculated by A* and Floyd-Warshall algorithm, Table 6.1 shows the best feature combinations for different cases with the feature ranks (1 indicates the most important) and Fig. 6.1 shows the percentage of the importance of each feature. Features are shown as letters:

a) Round Winner
b) Seconds after bomb planted
c) Average Bomb-Player distance for T-side
d) Average Bomb-Player distance for CT-side
e) Team total Weapon Cost for T-side
f) Team total Weapon Cost for CT-side
g) Average Player-Player distance of both sides
h) Ratio of Alive Players further than threshold distance for T-side
i) Ratio of Alive Players further than threshold distance for CT-side
j) Number of High-Performing Players for T-side
k) Number of High-Performing Players for CT-side
l) Number of Alive Players category
m) Available Flash Bang numbers category
n) Available Molotov (Grenade) numbers category
o) Available Smoke Grenade numbers category

Modelling Alg.	Path Plan Alg.	a	b	c	d	e	f	g	h	i	j	k	l	m	n
RF	A*	10	3	8	2	6	4	13	7	12		1	5	11	9
RF	Floyd-Warshall	8	3	10	2	5	4					1	6	9	7
LR	A*	6		3	2	4	7	8	10	11		1	9	5	
LR	Floyd-Warshall	8			2	3	7	10	5	6	11	1	9	4	
LGBM	A*	3	6	5	1	2	7					4	8		
LGBM	Floyd-Warshall	3	10	6	1	2	5	12	11			4	9	8	7

Table 6.1: Best Feature Combinations

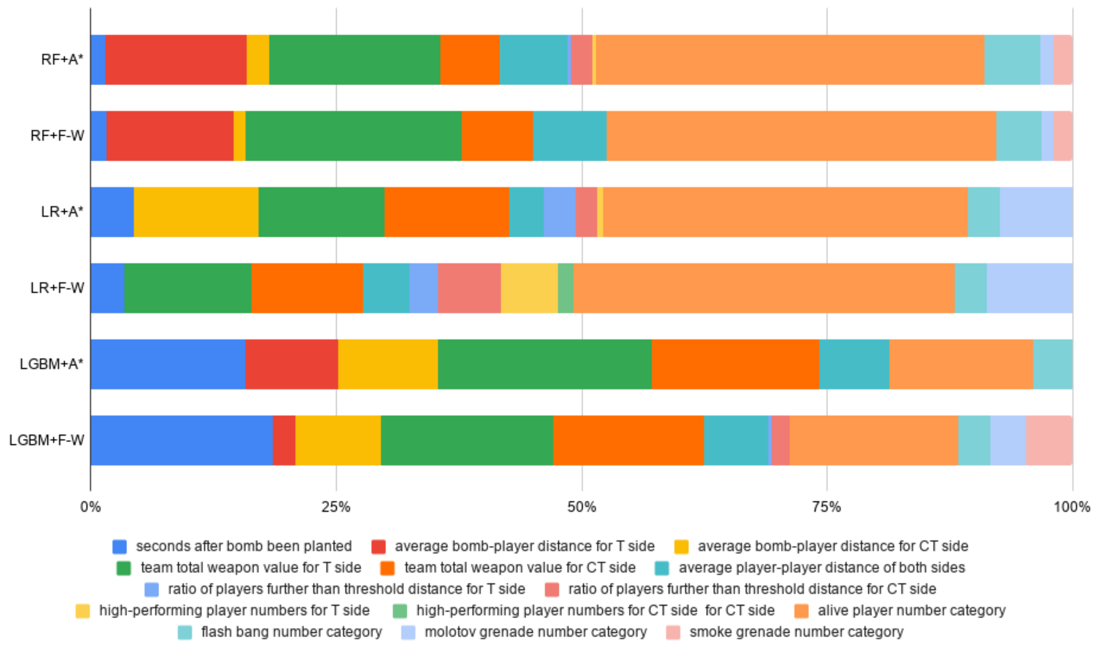


Figure 6.1: Feature Importance

From both the table and chart, the weapon values are the most important features for round winning prediction following with the alive player numbers of both sides and average player-player distance.

From Machine Learning algorithms aspect, the alive player numbers are not as important for LGBM as they are to the other two algorithms, while the average bomb-player distance of T-side is more essential for RF than for the other two. Even though the average bomb-player distance for T-side plays an important role for RF and the average bomb-player distance for CT-side contributes lots for LGBM, both of the distances are not impacting the results in most cases for LR. Also, it is noticeable that the high-performing player numbers within certain distance are not making big impact to the prediction.

From Shortest Path Planning algorithms aspect, A* and Floyd-Warshall perform generally the same, except for bomb-player distance for CT-side calculated by A* contributes more for all three predictions.

6.3 Modelling

6.3.1 Prediction Results

For each modelling and path planning algorithm combination, their best feature combination are used as input and the three scoring metrics are used to evaluate how the algorithms perform.

In Table 6.2, the accuracy, Logloss, F1-Score for both class are listed for different algorithm combinations before parameter tuning with the best values for each metric in bold. All of the metric values are also visualised in a line chart (Fig. 6.2). To make the visualisation more clear, $1-Logloss$ is used in the chart instead of Logloss due to the reason that smaller value indicates better results for only Logloss.

Modelling Alg.	Path Plan Alg.	Accuracy	Logloss	F1_T	F1_CT
RF	A*	0.868	0.320	0.905	0.765
RF	Floyd-Warshall	0.873	0.308	**0.913**	0.771
LR	A*	0.839	0.343	0.891	0.695
LR	Floyd-Warshall	0.822	0.364	0.882	0.640
LGBM	A*	**0.875**	**0.282**	**0.913**	**0.774**
LGBM	Floyd-Warshall	0.873	0.288	0.912	0.767

Table 6.2: Models Performance before Parameter Tuning

For the general performance, all models can successfully predict the round winner with acceptable results. Before hyper-parameter tuning, with the highest accuracy, F1-Score for both class and lowest Logloss, it can be clearly seen that LGBM with distance calculated by A* performs the best out of all the combinations.

For the three modelling algorithms, LGBM performs slightly better than RF, while LR is the worst out of the three models with 2% lower accuracy, 5% higher Logloss, 2% lower F1-Score for T-side prediction and 5% lower F1-Score for CT-side prediction.

For path planning algorithms, prediction results with A* distance are slightly better than those with Floyd-Warshall distance for LR and LGBM, while Floyd-Warshall distance works better with RF.

Although all models perform well in predicting the results, it can be noticed from the clear gap between the two F1-Scores (T-side around 0.9, CT around 0.7) that all models are better predicting the when the winner is the terrorists.

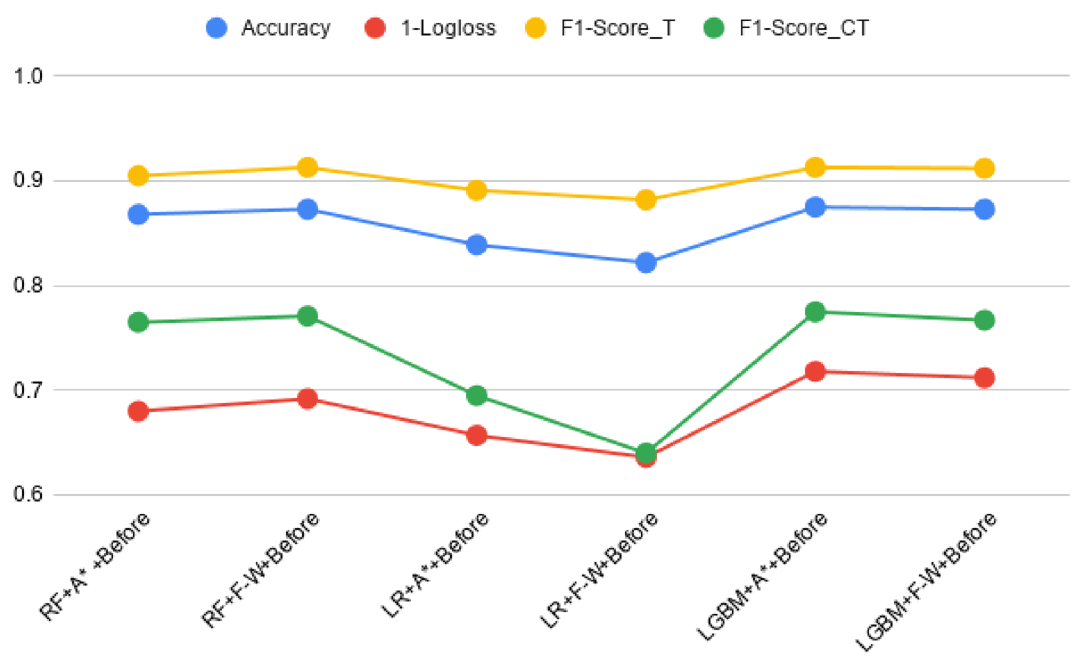


Figure 6.2: Algorithm Combination Performance Before Parameter Tuning

6.3.2 Parameter Tuning Results

The models performance after tuning is shown in Table 6.3 and the metric values are also visualised in the line chart (Fig. 6.3).

Modelling Alg.	Path Plan Alg.	Accuracy	Logloss	F1_T	F1_CT
RF	A*	0.877	**0.274**	0.916	0.775
RF	Floyd-Warshall	0.879	**0.274**	**0.917**	0.777
LR	A*	0.859	0.319	0.905	0.728
LR	Floyd-Warshall	0.853	0.326	0.900	0.721
LGBM	A*	0.878	**0.274**	**0.917**	0.774
LGBM	Floyd-Warshall	**0.881**	0.285	**0.917**	**0.787**

Table 6.3: Models Performance after Parameter Tuning

After the hyper-parameters been tuned, all three ML models preform better (Fig. 6.4), especially for LR with 3% higher accuracy , 4% lower Logloss, 2% higher F1-Score for T and 3% higher for CT. However, it is still the worst of all three. Interestingly, although LGBM still perform slightly better than RF, the best model after tuning is with distance calculated by Floyd-Warshall rather than A*.

The best parameters for the models are shown in Table 6.4. In general, path planning

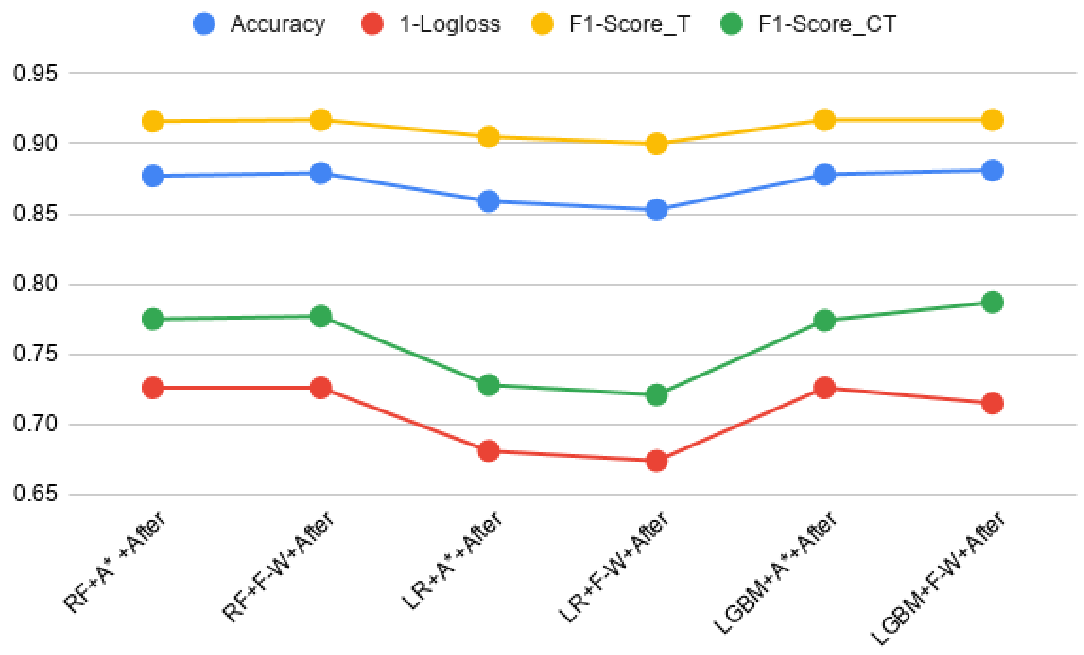


Figure 6.3: Algorithm Combination Performance after

algorithms do not effect the hyper-parameters except for LGBM.

For RF, the maximum default number of trees are applied while the model performs the best when maximum depth of the tree is at 10 rather than the max default value. That is to say, even though more data information is conveyed to the model when the tree is deeper, it doesn't equal to the better the prediction result will be.

For LR, L2 regularization works better since L2 is more sensible for outliers.

For LGBM, with distances from Floyd-Warshall algorithm, the large number of trees are not needed to obtain even better prediction results with the same number of leaves and maximum tree depth but slower learning rate. Also, comparing LGBM with RF, with maximum depth only of 3, LGBM performed 1% better on Accuracy and F1-Score for CT winning.

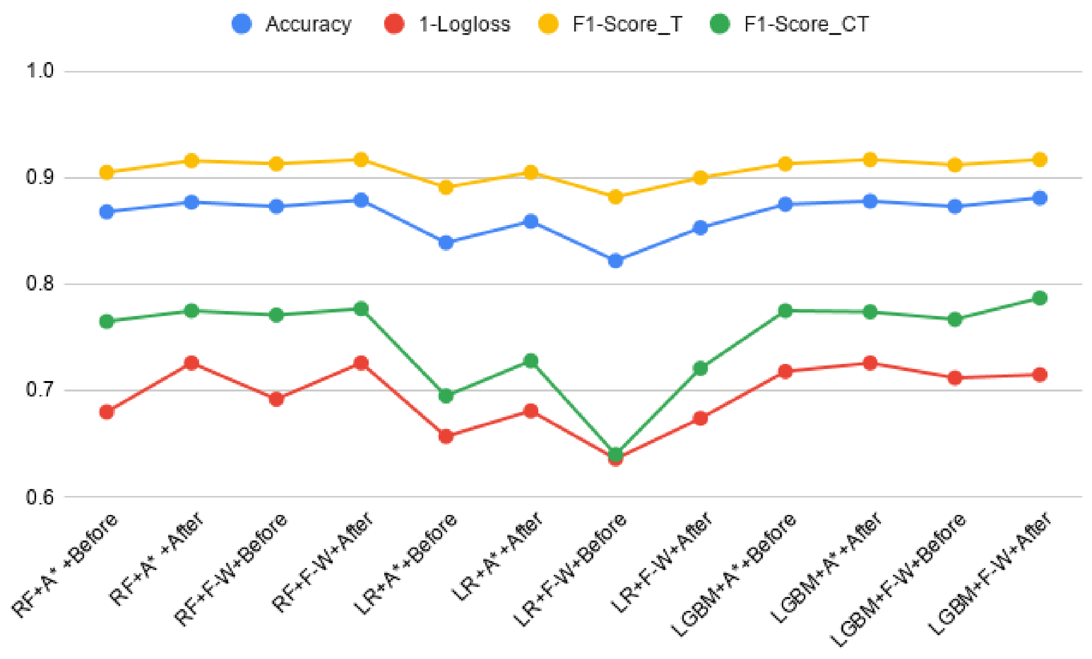


Figure 6.4: Algorithm Combination Performance

Model	n_trees	max_depth	penalty	solver	learning_rate	n_leaves
RF + A*	250	10				
RF + F-W	250	10				
LR + A*			l2	liblinear		
LR + F-W			l2	liblinear		
LGBM + A*	250	3			0.05	30
LGBM + F-W	40	3			0.2	30

Table 6.4: Best Parameters for Models

7 Conclusion

7.1 Discussion

For the main problem for this thesis - predicting eSports result using positional data, it can be seen in Sec. 6.3 that the results can be successfully predicted and the results are better than those work using only player skills as features. This proves that positional data is also one of the important features in the eSports field.

However, the positional features did not play the most important role in the prediction, especially when the man-set thresholds are involved. In this thesis, the value 200 is set as the maximum distance to distinguish "closer" and "further" players. The value is obtained from analyzing the map since 10 is set as one step of the player movement. But within 20 steps may not be close in the actual situation in the game depending on the used weapons of the players. For example, in knife rounds, a player within 5 steps may consider as "close" while for heavy weapon rounds, even 30 steps can still be considered as "close".

With the results obtained in Sec. 6.2, it is noticeable that all the models are only using part of the features to get the best performance, it shows that in predictions, more features are not technically related to better performance. For category data, encoding is a necessary step to extract the importance of the features. In this thesis, numeric encoding is used for category features, and seen from Fig. 6.1, it does a good job in converting feature importance and it can be seen that player alive numbers players a huge role in all models. Although they are indeed important for the final prediction, more players alive normally means higher total weapon values. With alive player numbers and weapon values related, they may impact the final results in an unknown way.

From the results from Sec. 6.3, it is noticeable that the F1-Score for T-side winning is much higher (around 20%) than CT winning. This may due to that after the bomb is planted, it is technically halfway for T-side to win since the T-side players will only have to eliminate the CT players to make sure the bomb explodes. In this case, the features that have been used can basically describe the whole situation for T players. However, CT players who need to defuse the bomb while killing the other side players, it is much harder to cover the game situation for CT to predict the results with current feature combinations. To solve this problem, more specific features will be needed, for example, combine player moving pace, player-bomb distance, and round time left will be useful to see if players have enough time to defuse the bomb, and the player numbers from each side within a certain distance around the bomb to see if there are CT players take covers for the player defusing the bomb.

From the hyperparameter aspect, it is clear from the results that LGBM is more sensitive to the parameters. With distances obtained from two path planning algorithms as the only different features, the hyper-parameters are totally different and change the performance of LGBM while the other two algorithms stay the same performance from the path planning algorithm aspect.

This result can be an inspiring work for future research in this topic for using positional data since currently, most works are still focusing on player skills or team rating in general to predict the results.

It can also be used in more Artificial Intelligence fields, especially computer games domain. For player behavior analysis for example, by analyzing the professional players to make better decisions to make NPCs or even robots. Also, this is helpful for all the sports related people to learn better gaming strategy.

7.2 Summary

In this thesis, multiple techniques are been used to predict CS:GO round winning results.

The programming language GO is utilized to obtain all the game data while the positional data are from QGIS, and the programming language Python is applied for further analysis.

To do the prediction, machine learning algorithms Logistic Regression, Random Forest, and Light Gradient Boosting Machine are used. For all three algorithms, the best accuracy after hyperparameter tuning are: 88.1% for LGBM, 87.9% for RF, and 85.9% for LR. For prediction results from the game sides aspect, the models do better work for T-side predicting than CT-side for a general F1-Score of 0.9 for T winning and 0.7 for CT winning.

Distances are needed for the training features and 4 methods are used to obtain the distances: A*, Floyd-Warshall, RRT*, and Manhattan Distance. A* and Floyd-Warshall perform fairly well in planning the path of players while RRT* fails to perform due to high time complexity. Manhattan Distance is used for both A* and Floyd-Warshall as back up distance calculator. In general, with the same machine learning algorithms, Floyd-Warshall distances can give better prediction results with proper tuning for hyper-parameters for tree-based algorithms and A* is better for Logistic Regression.

The tuning method has been used in this thesis is Grid Search with 5-fold cross-validation. It is proven that tuning hyper-parameters can indeed improve the model performance (in this thesis for LR especially), however, parameter tuning does not influence the performance for RF that much.

For the parameters, it can be concluded that L2 regularization works better in this thesis than L1 for the reason that L2 is sensitive to outliers. For tree-based algorithms, a deeper tree does not always transform to better results. Furthermore, the parameter performance for LGBM cannot be concluded for this thesis yet since the parameters make big impact to the distance features. Thus, some future work for LGBM parameter tuning is needed.

Unlike Rizani et al. stated that the round winning prediction is stochastic and unpredictable [RI18], this thesis proves that the round winning prediction can be generally successfully generated. Furthermore, it has an overall better performance than the results (highest accuracy of 62% and lowest Logloss of 0.675) presented by Makarov et al. [IMI].

7.3 Future Work

Although the results are relatively satisfying, there is still room for improvement.

- As stated in Sec. 7.1, both *ratio of players are far away from the bomb* and *number of high-performing players are close to opponent players* are not contributing enough when doing feature selection, while they actually play relatively important roles in real games. Therefore, more distance thresholds can be set to obtain more values for both features to analyze how the distance affects the outcome.

- As mentioned in Sec. 6.3, the F1-Score for predicting CT-side winning is not as good as the F1-Score for T-side winning, for the reason that having the bomb planted has already increased the chance of T-side winning and makes it harder to scale the situation for CT-side. More features to describe the CT situation may help with getting better F1-Scores for CT winning.

- More default hyper-parameters should be given for LGBM for better performance since the parameters are impacting how path planning methods contribute.

- It can be seen in Sec. 6.2 that encoding category data is fairly important. More encoding methods can be applied on category data to find out how they impact the results.

- More features, in general, would impact the outcome, too. In this thesis, some data are extracted but didn't been used for final modelling because of the complex pre-calculation of the features. For example, with player velocity and moving direction, the player movements can be predicted with other techniques, for instance, Kalman Filter. With this information, the distances will be more accurate and prevent the situation that two players are very close to each other but moving towards a different direction. Also, more weapon related features, for example, player kills with their weapons names, can be combined together and having an estimated value (similar to the categorical features) to reduce the training time.

- Due to the time limit, the game map is used as a 2D map to calculate the distances. With enough time and more research, better Path Planning Algorithms can be used in the future to obtain distances from 3D maps.

- Because of two coordinate systems, there are minor errors of the transformation between WGS84 and in-game positions. A more accurate interpolation method can be applied instead of the simple linear one.

- With less consideration about time consumption, the grid size can be reduced to the size of the wall in the game, so that the players hiding behind the wall are not considered standing in a not reachable area by the algorithms.
- More data is always a good way to improve the predicting results. If live data is available, the best models and parameters can be saved with more data coming through. Some Automated Machine Learning techniques, TPOT for example, can be applied. This technique can also generate new features within learning for better results.
- The results of this work can be further used as a feature for the game overall win-lose results prediction.

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
CS:GO	Counter Strike: Global Offensive
CT	Counter-Terrorists
DQL	Deep-Q Learning
FSN	Finite State Machines
GB	Gradient Boosting
GIS	Geographic Information System
GR	Generalization Ratio
HLTV	Half-Life Television
LGBM	Light Gradient Boosting Machine
LR	Logistic Regression
ML	Machine Learning
MLP	Multi-layered Perceptron
MMO	Massive Multiplayer Online
NPC	Non-Player Character
RF	Random Forest
RRT	Rapidly-exploring Random Trees
SVM	Support-Vector Machines
T	Terrorists

Bibliography

[AB09] ARGALL, B. D., CHERNOVA S. VELOSO M. and B. BROWNING. 57(5), 2009.

[A.B17] A.BJÖRKLUND, F.LINDEVALL, P.SVENSSON W.J.VISURI: *Predicting the outcome of CS:GO games using machine learning.* 2017.

[AdMLS18] AZIS, H., R. D. MALLONGI, D. LANTARA and Y. SALIM: *Comparison of Floyd-Warshall Algorithm and Greedy Algorithm in Determining the Shortest Route.* In *2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT)*, pages 294–298, 2018.

[AHDD19] ANI, R., V. HARIKUMAR, A. K. DEVAN and O. S. DEEPA: *Victory prediction in League of Legends using Feature Selection and Ensemble methods.* In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 74–77, 2019.

[Ata] ATANASOV, NIKOLAY: *ECE276B: Planning Learning in Robotics Lecture 8: Sampling-based Planning.*

[atl] ATLEADAI: *Overview of the advantages and disadvantages of each algorithm and application scenarios in supervised learning.*

[Ber44] BERKSON, JOSEPH: *Application of the Logistic Function to Bio-Assay.* Journal of the American Statistical Association, 39(227):357–365, 1944.

[Ber17] BERNSTEIN, MATTHEW N.: *Random Forest Notes.* 2017.

[BNR19] BEAL, RYAN, TIMOTHY J. NORMAN and SARVAPALI D. RAMCHURN: *Artificial intelligence for team sports: a survey.* The Knowledge Engineering Review, 34:e28, 2019.

[Cap] CAPTAIN: *Summary of common methods for deep learning tuning.*

[CFI] CFIBOOSTING: *Boosting.*

[CS14] CLARK, CHRISTOPHER and AMOS STORKEY: *Teaching Deep Convolutional Neural Networks to Play Go.* 12 2014.

[CST+18] CHEN, L., Y. SHAN, W. TIAN, B. LI and D. CAO: *A Fast and Efficient Double-Tree RRT*-Like Sampling-Based Planner Applying on Mobile Robotic Systems.* IEEE/ASME Transactions on Mechatronics, 23(6):2568–2578, 2018.

[Dji] Djidjev, Hristo N: *Efficient Shortest Path Computations on Multi-GPU Platforms.*

[DK13] Djojo, M. A. and K. Karyono: *Computational load analysis of Dijkstra, A*, and Floyd-Warshall algorithms in mesh network.* In *2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems*, pages 104–108, 2013.

[DP13] Deisenroth, M. P., Neumann G. and J. Peters. 2, 2013.

[Dub18] Dubbs, Alexander: *Statistics-free Sports Prediction.* IOS PRESS, page 173 â 181, 2018.

[EFF02] E. Frazzoli, M. A. Dahleh and E. Feron: *Real-time motion planning for agile autonomous vehicles.* 25(1):116–129, 2002.

[FJ15] F. Johansson, J. Wikström: *Result Prediction by Mining Replays in Dota 2.* 2015.

[flo] *Practical Guide to Hyperparameters Optimization for Deep Learning Models.*

[Flo62] Floyd, Robert W.: *Algorithm 97: Shortest Path.* Commun. ACM, 5(6):345, June 1962.

[Fri01] Friedman, Jerome H.: *Greedy Function Approximation: A Gradient Boosting Machine.* The Annals of Statistics, 29(5):1189–1232, 2001.

[Gup] Gupta, Shailaja: *Pros and cons of various Machine Learning algorithms.*

[HB08] Hladky, S. and V. Bulitko: *An evaluation of models for predicting opponent positions in first-person shooter video games.* In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 39–46, 2008.

[HDS+19] Hodge, V., S. Devlin, N. Sephton, F. Block, P. Cowling and A. Drachen: *Win Prediction in Multi-Player Esports: Live Professional Match Prediction.* IEEE Transactions on Games, pages 1–1, 2019.

[HNR68] Hart, P. E., N. J. Nilsson and B. Raphael: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

[IMI] Ilya Makarov, Dmitry Savostyanov, Boris Litvyakov and Dmitry I. Ignatov: *Predicting Winning Team and Probabilistic Ratings in Dota 2" and Counter-Strike: Global Offensive" Video Games.*

[JJP08] Jiang, Yang, Jianmin Jiang and Ian Palmer: *Computerized Interactive Gaming via Supporting Vector Machines.* Int. J. Computer Games Technology, 2008, 12 2008.

[JK17] JAIN, S. and H. KAUR: *Machine learning approaches to predict basketball game outcome.* In *2017 3rd International Conference on Advances in Computing,Communication Automation (ICACCA) (Fall)*, pages 1–7, 2017.

[JL18] JOSLYN, K. and J. LIPOR: *A Supervised Learning Approach to Water Quality Parameter Prediction and Fault Detection.* In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2511–2514, 2018.

[Jon] JONES, M.TIM: *Machine Learning and Gaming.*

[KAA20] KHATRI, S., A. ARORA and A. P. AGRAWAL: *Supervised Machine Learning Algorithms for Credit Card Fraud Detection: A Comparison.* In *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 680–683, 2020.

[KF11] KARAMAN, SERTAC and EMILIO FRAZZOLI: *Sampling-based Algorithms for Optimal Motion Planning*, 2011.

[KP13] KOBER, J., BAGNELL J. A. and J. PETERS. 32(11), 2013.

[KSH+19] KATONA, A., R. SPICK, V. J. HODGE, S. DEMEDIUK, F. BLOCK, A. DRACHEN and J. A. WALKER: *Time to Die: Death Prediction in Dota 2 using Deep Learning.* In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019.

[LaV98] LAVALLE, S. M.: *Rapidly-exploring random trees a new tool for path planning.* 1998.

[LB13] LAWRENCE, R. and V. BULITKO: *Database-Driven Real-Time Heuristic Search in Video-Game Pathfinding.* IEEE Transactions on Computational Intelligence and AI in Games, 5(3):227–241, 2013.

[LCC18a] LIN, J., C. CHAO and J. CHIOU: *Determining Neuronal Number in Each Hidden Layer Using Earthquake Catalogues as Training Data in Training an Embedded Back Propagation Neural Network for Predicting Earthquake Magnitude.* IEEE Access, 6:52582–52597, 2018.

[LCC18b] LIN, S., M. CHEN and H. CHIANG: *Forecasting Results of Sport Events Through Deep Learning.* In *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 2, pages 501–506, 2018.

[Lin] LIN, NPEIN: *A brief summary of the advantages and disadvantages of various classification algorithms.*

[LMN+17] LUCCHESE, CLAUDIO, CRISTINA MUNTEAN, FRANCO NARDINI, RAFFAELE PEREGO and SALVATORE TRANI: *RankEval: An Evaluation and Analysis Framework for Learning-to-Rank Solutions.* pages 1281–1284, 08 2017.

[LN14] Lock, Dennis and Dan Nettleton: *Using random forests to estimate win probability before each play of an NFL game.* Journal of Quantitative Analysis in Sports, 10, 06 2014.

[MT08] McCabe, A. and J. Trevathan: *Artificial Intelligence in Sports Prediction.* In *Fifth International Conference on Information Technology: New Generations (itng 2008)*, pages 1194–1197, 2008.

[Ng20] Ng, Andrew: *CS229 Lecture Notes.* CS229 Lecture Notes:16–19, 2020.

[OBE19] Oughali, M. S., M. Bahloul and S. A. El Rahman: *Analysis of NBA Players and Shot Prediction Using Random Forest and XGBoost Models.* In *2019 International Conference on Computer and Information Sciences (ICCIS)*, pages 1–5, 2019.

[OE16] Obermeyer, Ziad and Ezekiel Emanuel: *Predicting the Future - Big Data, Machine Learning, and Clinical Medicine.* The New England journal of medicine, 375:1216–1219, 09 2016.

[PH16] Prasetio, D. and D. Harlili: *Predicting football match results with logistic regression.* In *2016 International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*, pages 1–5, 2016.

[PP16] Pretorius, Arnu and Douglas A. Parry: *Human Decision Making and Artificial Intelligence: A Comparison in the Domain of Sports Prediction.* In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '16, New York, NY, USA, 2016. Association for Computing Machinery.

[PS03] Pemmaraju, Sriram and Steven Skiena: *Algorithmic Graph Theory*, page 321â374. Cambridge University Press, 2003.

[PVG+11] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay: *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011.

[RI18] Rizani, M. N. and H. Iida: *Analysis of Counter-Strike: Global Offensive.* In *2018 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pages 373–378, 2018.

[SAL+12] Sharma, Harshita, Alexander Alekseychuk, Peter Leskovsky, Olaf Hellwich, R.s Anand, Norman Zerbe and Peter Hufnagl: *Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics.* Diagnostic pathology, 7:134, 10 2012.

[Sam00] Samuel, Arthur: *Some Studies in Machine Learning Using the Game of Checkers. II-Recent Progress.* IBM Journal of Research and Development, 3:206 – 226, 02 2000.

[Suk] Sukaj, Endi: *dijkstra-s-algorithm*$_{v}s_{a} - algorithm$.

[TLL+18] Tang, X., Z. Liu, T. Li, W. Wu and Z. Wei: *The Application of Decision Tree in the Prediction of Winning Team.* In *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 239–242, 2018.

[TMJ+19] Thenmozhi, D., P. Mirunalini, S. M. Jaisakthi, S. Vasudevan, V. Veeramani Kannan and S. Sagubar Sadiq: *MoneyBall - Data Mining on Cricket Dataset.* In *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, pages 1–5, 2019.

[VFKP09] Vonásek, Vojtěch, Jan Faigl, Tomáš Krajník and Libor Přeučil: *RRT-path – A Guided Rapidly Exploring Random Tree.* In Kozłowski, Krzysztof R. (editor): *Robot Motion and Control 2009*, pages 307–316, London, 2009. Springer London.

[Wika] Wiki: *Wikipedia: Hyperparameter Optimization.*

[Wikb] Wiki: *Wikipedia: Machine Learning.*

[Wikc] Wiki: *Wikipedia: Reinforcement learning.*

[Wikd] Wikipedia: *Random Forest.*

[WWP00] Weber, M., M. Welling and P. Perona: *Unsupervised Learning of Models for Recognition.* In *Computer Vision - ECCV 2000*, pages 18–32, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[XCY14] Xinggang, W., G. Cong and L. Yibo: *Variable probability based bidirectional RRT algorithm for UAV path planning.* In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 2217–2222, 2014.

[YLC+14] Yue, Y., P. Lucey, P. Carr, A. Bialkowski and I. Matthews: *Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction.* In *2014 IEEE International Conference on Data Mining*, pages 670–679, 2014.

[Yun11] Yun, Hyokun: *Using Logistic Regression to Analyze the Balance of a Game: The Case of StarCraft II.* Computing Research Repository - CORR, 05 2011.

[Zho] Zhongxiang, Dai: *Bayesian Optimization.*

Appendix

Listing 1: Example code for getting player positions

```go
func Snapshot(p *dem.Parser) map[string]r3.Vector {
    position := make(map[string]r3.Vector)

    for _, pl := range p.GameState().Participants().Playing() {
        if pl.IsAlive() == true {
            position[pl.Name] = pl.Position
        }
    }
    return position
}
```

Listing 2: Example code for registering BombPlanted event

```go
p.RegisterEventHandler(func(e events.BombPlanted) {
    isBombPlantActive = true
    lastPlantTime = p.CurrentTime()
    lastSnapshotTime = p.CurrentTime()

    bombPosition = p.GameState().Bomb().Position()
    currentPPPos = make(map[time.Duration]map[string]r3.Vector)
    currentPPPos[0] = Snapshot(p)
})
```

Listing 3: Example code for registering FrameDone event

```go
p.RegisterEventHandler(func(e events.FrameDone) {
    const snapshotFrequency = 5 * time.Second
    now := p.CurrentTime()

    if isBombPlantActive &&
        (lastSnapshotTime+snapshotFrequency) < now {
        lastSnapshotTime = now
        currentPPPos[now-lastPlantTime] = Snapshot(p)
        bombPosition = p.GameState().Bomb().Position()
    }
})
```

Listing 4: Example code for registering RoundEnd event

```
p.RegisterEventHandler(func(e events.RoundEnd) {
    if !isBombPlantActive {
        return
    }
    isBombPlantActive = false
    roundsPlayed = p.GameState().TotalRoundsPlayed()
    postPlantPositions[roundsPlayed] = currentPPPos
    bombPositions[roundsPlayed] = bombPosition
})
```