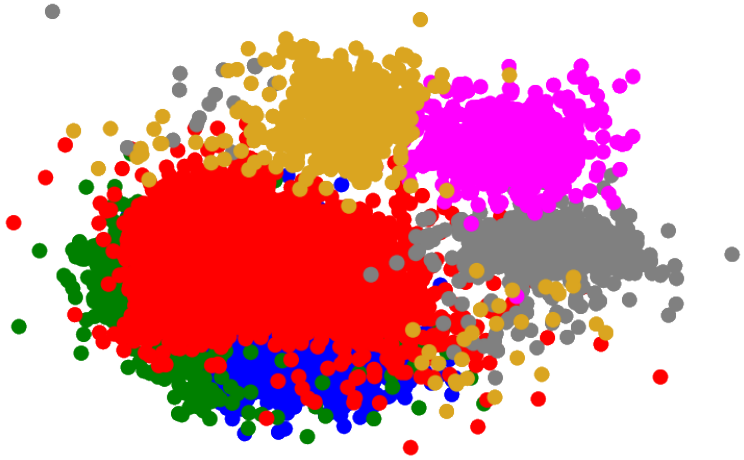
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Clustering and Classification of Time Series in Real-Time Strategy Games

A machine learning approach for mapping StarCraft II games to clusters of game state time series while limited by fog of war

Bachelor's thesis in Software Engineering

Olof Enström,
Fredrik Hagström,
John Segerstedt,
Fredrik Viberg,
Arvid Wartenberg,
David Weber Fors

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Bachelor's thesis 2019:81

Clustering and Classification of Time Series in Real-Time Strategy Games

A machine learning approach for mapping StarCraft II games to clusters of game state time series while limited by fog of war

OLOF ENSTRÖM
FREDRIK HAGSTRÖM
JOHN SEGERSTEDT
FREDRIK VIBERG
ARVID WARTENBERG
DAVID WEBER FORS

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

DATX02-19-81

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Clustering and Classification of Time Series in Real-Time Strategy Games
- A machine learning approach for mapping StarCraft II games to clusters of game state time series while limited by fog of war

Department of Computer Science and Engineering
Chalmers University of Technology
06-2019

Supervisor:
Alex Gerdes, Department of Computer Science and Engineering

Examiners:
Wolfgang Ahrendt, Department of Computer Science and Engineering
Morten Fjeld, Department of Computer Science and Engineering
Sven Knutsson, Department of Computer Science and Engineering

Bachelor's Thesis 2019:81
DATX02-19-81
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of game state time series from StarCraft II game states in two dimensions.

Clustering and Classification of Time Series in Real-Time Strategy Games
- A machine learning approach for mapping StarCraft II games to clusters of game state time series while limited by fog of war

Department of Computer Science and Engineering
Chalmers University of Technology
06-2019

Abstract

Real-time strategy (RTS) games feature vast action spaces and incomplete information, thus requiring lengthy training times for AI-agents to master them at the level of a human expert. Based on the inherent complexity and the strategical interplay between the players of an RTS game, it is hypothesized that data sets of played games exhibit clustering properties as a result of the actions made by the players. These clusters could potentially be used to optimize the training process of AI-agents, and gain unbiased insight into the gameplay dynamics. In this thesis, a method is presented to discern such clusters and classify an ongoing game according to which of these clusters it most closely resembles, limited to the perspective of a single player. Six distinct clusters have been found in StarCraft II using hierarchical clustering over time, all of which depend on different combinations of game pieces and the timing of their acquisitions in the game. An ongoing game can be classified, using neural networks and random forests, as a member of some cluster with accuracies ranging from 83% to 96% depending on the amount of information provided.

Keywords: Classification problem, Cluster analysis, Hierarchical clustering, Machine learning, Neural network, Random forest, Real-time strategy, StarCraft II, Time series

Clustering and Classification of Time Series in Real-Time Strategy Games
- A machine learning approach for mapping StarCraft II games to clusters of game state time series while limited by fog of war

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
06-2019

Sammandrag

I realtidsstrategispel har spelaren en stor mängd möjliga handlingar och inkomplett information om motståndaren, vilket innebär långa träningstider för att AI-agenter skall kunna uppnå samma skicklighetsnivå som mänskliga expertspelare. Baserat på spelets komplexitet och det strategiska samspelet mellan spelarna hypotetiseras det att datamängden bestående av spelade matcher har klustertendenser som ett resultat av de handlingar spelarna gör under matchernas gång. Dessa kluster kan potentiellt användas för att optimera träningen av AI-agenter, och för att få opartisk insikt i spelens dynamik. I den här rapporten presenteras en metod för att utläsa sådana kluster och klassificera klustertillhörighet av pågående matcher, baserat på inkomplett information. Sex distinkta kluster har hittats i StarCraft II med hjälp av hierarkisk klustring över tid vilka alla beror på olika kombinationer av spelpjäser och tidpunkterna då de införskaffades. En pågående match kan klassificeras som en medlem av något av dessa kluster med hjälp av random forest och neurala nätverk med en träffsäkerhet mellan 83% och 96%, beroende på hur mycket information som används.

Acknowledgements

The authors would like to thank Alex Gerdes for his guidance, support and feedback. Furthermore, we would also like to thank Miguel González Duque for his open source software sc2reaper [1] and his support, without which this thesis would not have been possible. Lastly we thank Morteza H. Chehreghani for his feedback and advice on our cluster analysis.

Olof Enström

Fredrik Hagström

John Segerstedt

Fredrik Viberg

Arvid Wartenberg

David Weber Fors

Chalmers University of Technology
Gothenburg
June, 2019

Glossary

- **Action space**: the set of available actions at any given time in an *RTS*-game.
- **Baseline**: The lowest possible accuracy of a model, in this thesis corresponding to the size of the largest cluster. The baseline is always equal to or larger than chance.
- **Fog of war**: The area of the game board which a player does not currently have vision of.
- **Game piece**: Defined in this thesis as either a unit, building, or upgrade in the game StarCraft II.
- **Game state**: All game pieces each player controls at a certain point in time.
- **Match-up**: A certain race in StarCraft II versus another. For example one match-up would be Protoss versus Protoss (PvP).
- **Race**: There are three different races to play in StarCraft II: Terran (T), Protoss (P) and Zerg (Z)
- **RTS**: *Real Time Strategy.* An RTS game such as StarCraft II where players can take actions continuously throughout the game, unlike turn-based strategy games such as chess.
- **StarCraft II**: A real-time strategy game released by Blizzard in 2010. More information on the game can be found in Section 1.1.1 and Section A.1.
- **Time series**: The development of a set of variables over time. In this thesis, game state time series denotes the development of a game's state over time.

Terminology defined here is italicized at their first use.

Contents

1

Introduction

This chapter presents an introduction to the thesis with a background, purpose, scope, research questions, and contributions.

1.1 Background

AI research has made rapid progress in recent years, entering new fields of gradually increasing complexity. In 1997, a chess-playing computer developed by IBM defeated the world chess champion, Garry Kasparov [2]. In 2017, the computer program AlphaGo Master, developed by Google DeepMind, defeated the highest ranked player Ke Jie in the game GO [3]. Advances in AI are even making its way into everyday life in many ways, including autonomously driving cars [4], computer programs capable of understanding human speech [5] and identifying images [6] with superhuman capacity.

Computer games in particular are a common test bed for AI because of their complexity in regards to their *action space* and their strategic depth, thereby providing an ample challenge and potential application for AI research, making them a natural next step after having conquered chess and GO. Further reasons for the large amount of AI research being conducted on computer games include the clearly defined rules and limitations of the games, the large amounts of available training data, and the potential to show the decisions a trained AI is making in a visual way [7].

1.1.1 Introduction to StarCraft II

StarCraft II is an *RTS* game released by Blizzard in 2010. The game features various different game modes, but the main competitive one-versus-one mode is the only one discussed in this paper. The gameplay of StarCraft II features elements of producing buildings and units, while also having players handling resource management and planning long-term strategy [8]. Adding strategic depth to the game, each player has limited vision of the other player's actions, buildings and units, which is called the *fog of war*. A player wins when their opponent either surrenders or no longer controls any buildings [8]. Furthermore, the game features three distinctively different playable *races*, each with their own arsenal of buildings and units, all of which have their own strengths and weaknesses.

Hundreds of thousands of games of StarCraft II are played each day online [9] and during last year alone professional full-time players competed in different tournaments across the globe for a total prize pool of over $4.5 million. [10].

Further information on, and informative figures representing gameplay of, StarCraft II can be found in Appendix A.1.

1.1.2 AI development in StarCraft II

StarCraft II has as of early 2017 entered the center stage of AI development after the release of its machine learning oriented API [11]. In early 2019, DeepMind unveiled their latest accomplishment: managing to produce and train a neural network which defeated a top-ranked StarCraft II player in five out of five games [12]. However, the validity of this match is questionable as AlphaStar, DeepMind's AI, displayed superhuman speed and reflexes [13] [14], something the DeepMind team specifically stated they wanted to avoid [12].

Professional players of StarCraft II are still far away from being outclassed by AI, as in the earlier projects of GO and such. When the problem domain grows larger, it becomes increasingly difficult to train neural networks to make the right decision given a certain set of inputs [15]. Efforts are being made to improve the structure and training methods in order to reduce the time needed for training [16].

The lengthy training times are areas of potential improvement within the existing StarCraft II machine learning agents. When Google trained their AlphaStar agents, they reported that each agent trained for an equivalent of up to 200 years of real time [12]. If the goal of AI research in general is to be reminiscent of human learning, such lengthy training times are problematic. There are professional players that achieve a similar ability in far less real time, indicating that perhaps there are ways of improving upon the learning process and making it more efficient.

1.1.3 Divide and conquer in AI training

One hypothetical and interesting approach to the problem of lengthy training times is to train multiple agents on specific subsets of replay data of StarCraft II, instead of training one agent on all available data. By finding ways of dividing the available training data into distinct clusters, one could perhaps parallelize the training process by training agents on each of these clusters simultaneously. Perhaps even more importantly, by allowing each agent to train on one specific data cluster, the variance in the training data is reduced, which could lead to more efficient problem solving in general. With this method, one could perhaps more efficiently utilize the rapid initial growth in win rate with respect to time spent training.

This approach contrasts to indiscriminately training one agent on all available data, where the variance is likely to be high. The hypothesized result of this type of training would be that one would have several optimized agents at one's disposal, each specialized in gameplay similar to that of its designated cluster, much like a skilled player has several different strategies or reactions at his or her disposal ready to be deployed at any time.

The envisioned application of these ideas would consist of a system that can switch between a multitude of AI agents. Each of these agents should be trained on unique subsets of games separated into different precalculated clusters of *game state time series*, i.e. the development of a game's state over time. Game state in this context refers to the collected information about all *game pieces*, i.e. buildings, units, and upgrades, controlled by both players at a certain point in time during the game.

However, producing such a system in its entirety is beyond the scope of this project. This Bachelor's thesis focuses solely on the cluster identification and game state classification parts of the envisioned system.

Beyond the scope of StarCraft II, methods and results presented in this bachelor thesis may translate well when considering closely related problems, such as other RTS games.

A recent study by Hayes and Beling [17] employs unsupervised hierarchical clustering to classify similar build orders as a strategy in StarCraft I, StarCraft II's predecessor. A search through relevant literature reveals no previous research that use clustering in a similar fashion in StarCraft II. This thesis will investigate clustering to classify similar game state time series in StarCraft II.

1.2 Purpose

The purpose of this Bachelor's Thesis is to find distinguishable clusters of game state time series from replay data of StarCraft II games, and to test the feasibility of using machine learning methods to classify game state time series as members of the clusters, while limited to the perspective of one player.

Firstly, the replay files containing information of past games will be translated into game state time series, which will later be grouped into clusters. These clusters will be constructed through finding similarity between subsets of data using an agglomerative hierarchical clustering method. In this thesis, a game state is defined as all game pieces each player controls at a certain point in time where a game piece is defined as either a unit, building, or upgrade in the game. To later validate success in finding distinct clusters, both mathematical evaluations and feature analysis will be performed.

Secondly, classifier models will be constructed to identify which cluster a game currently is in, given only the information that is available to one player. The challenge is that the clusters are formed using perfect information, knowledge of the entire game state, whilst the classifiers will only be given the information of one player's game pieces and those of its opponent that the player has spotted during the game. This presents a classification task where the models are to learn, through training, to match these incomplete game state time series with the pre-calculated clusters.

1.3 Scope

This project will be conducted in regards to the following constraints:

Only data from compatible versions of the game will be used, as this assures that the features are as distinct as possible. When StarCraft II gets updated, which happens regularly, gameplay may differ for different game versions which is to be avoided.

A single *match-up* will be considered: Protoss versus Protoss. This restriction will be done to minimize the complexity of data by reducing the variance, as game states from different match-ups are non-compatible and thus, they wont exist in shared clusters.

This thesis does not aim to examine the absolute optimal means of clustering of game states. Instead, a few select clustering methods will be trialed and the one that delivers the most distinct clusters will be chosen.

This thesis proposes the hypothesis that the total training time of an AI agent for StarCraft II can be reduced whilst still producing an agent with equal or better performance if training data is split into subsets, where the subsets are distinctly different. Observe that this thesis does not aim to prove the hypothesis. The aim is solely to identify the subsets of games needed in the hypothetical system, in addition to examining if classification of full game state clusters is possible by using partial game state information. In essence, the results of this thesis will be a segment of the envisioned system.

1.4 Research Questions

The aim of this Bachelor's thesis is two-fold since the purpose consists of two phases. The purpose suggests an initial phase of data analysis, followed by the development and testing of different types of classifiers.

The objective of the data analysis phase is to investigate the feasibility of using clustering methods to find distinct partitions of game state time series in StarCraft II replay data. The following questions were composed to serve as a guideline to achieve this objective:

- Is it possible to find meaningful clusters of game state time series using replay data from StarCraft II?
  - If so, what is an appropriate clustering method?

The main objective for the classification phase is to identify classification methods capable of successfully categorizing which cluster a game state time series belongs in given incomplete information. The research conducted during this phase will aim to answer the following questions:

- Which game pieces are relevant for effectively distinguishing between these clusters?

- Are neural networks and/or random forests feasible methods of classifying clusters of game state time series using incomplete information?
  - If so, how accurate are the methods?

1.5 Contributions

This thesis contributes an initial view of clustering with respect to game state time series in StarCraft II. It provides a method by which one can find clusters in such data. This method builds upon the work of Hayes and Beling [17], who performed a similar analysis but on single player strategies in the original StarCraft game, by expanding the perspective and studying the combined game states of both players in a game.

In addition to the clustering, this thesis provides a study of the feasibility of using artificial neural networks and random forests to classify ongoing games, with limited information, to predefined clusters. It is shown that both neural networks and random forests can be used for this purpose, with the latter method also providing a way of analyzing the cluster contents in terms of game pieces.

The methods presented in this thesis are not completely dependent on the environment of StarCraft II. As such, it is probable that these methods can be applied to other RTS games, producing similar results.

2

Theory

This chapter aims to provide the sufficient theoretical background required for readers of this thesis, including topics such as cluster analysis, and machine learning methods of classification.

2.1 Cluster analysis

Cluster analysis is a method of data analysis by which one can find natural partitions on a data set. Given a data set D, one partitions the data set based on some similarity measure into a set of clusters $C = \{C_i : 1 \leq i \leq K\}$, where K is the number of clusters, such that $C_i \cap C_j = \emptyset, i \neq j$, i.e. no data point belongs to several clusters at once. The goal is to find such partitions which represent differences between data points, so that each cluster contains data points which are similar to each other and dissimilar to data points not in that cluster in some meaningful way.

In relation to StarCraft II specifically, and in the context of this study, a data point could correspond to the state of the game at a particular point in a replay, containing numerical data of certain attributes of the gameplay before being handed to the clustering algorithm. The algorithm itself then groups the data together based on various kinds of similarities which often depend on the clustering method used.

2.1.1 Distance functions

In order to perform a cluster analysis of a data set, one must have a definition of the distance between the data points. Based on this distance, it is possible to determine how the data points are to be partitioned in the clustering. Such distances can be defined by distance functions, also called metrics. Familiar examples of distance functions include Manhattan distance and the Euclidean distance.

Given a set C, a metric [18] on C is a function $d : C \times C \to [0, \infty)$ such that $\forall x, y, z \in C$, it fulfills the following criteria

$$\begin{aligned} &1. \quad d(x,y) \geq 0 \\ &2. \quad d(x,y) = 0 \text{ if and only if } x = y \\ &3. \quad d(x,y) = d(y,x) \\ &4. \quad d(x,z) \leq d(x,y) + d(y,z) \end{aligned} \tag{2.1}$$

Intuitively, one may think of the distance function in terms of familiar everyday measures of distances on the set of spatial coordinates. However, any function d on a set of arbitrary types of members which satisfies the criteria above is by definition considered a distance.

Consider two data points represented by the vectors $\mathbf{x} = (x_1, ..., x_n), \mathbf{y} = (y_1, ..., y_n)$. The Manhattan distance is defined as

$$d_{\text{manhattan}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|,$$

which derives it's name from the fact that the metric corresponds to distances as measured on the gridlike street geography of Manhattan. The Euclidean is defined as

$$d_{\text{euclidian}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2},$$

which familiarly reduces to Pythagoras theorem if $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$.

The cosine dissimilarity $d_{\cos}$ is defined as

$$d_{\cos}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} = 1 - \cos(\theta), \tag{2.2}$$

where θ is the angle between the vectors $\mathbf{x}$ and $\mathbf{y}$. An important fact is that $d_{\cos}$ violates the fourth criterion in 2.1, and as such it is not a proper distance function. Distances which satisfy only the first three criterion are usually called dissimilarities [19]. However, for the intents and purposes of this cluster analysis, this bears no great importance, since hierarchical clustering, which will be introduced later, can use dissimilarities as input [19].

When performing cluster analysis, the cosine dissimilarity can be used if one represents data points as vectors pointing from the origin. The cosine dissimilarity is very different from the Euclidean and Manhattan distances in that it measures the angle between vectors representing data points, and disregards their relative magnitude. Parallel vectors are considered identical, while orthogonal vectors are considered to be spaced far apart, in positive space, i.e. where all vector components are positive. This has the effect of placing greater importance upon the internal proportions of the vector components rather than the total magnitudes of the vectors, e.g. the vector $(1, 1)$ is considered identical to the parallel vector $(10, 10)$, and it is considered maximally different to the orthogonal vector $(0, 1)$.

The cosine dissimilarity can also be shown to be related to the Euclidean distance under certain conditions [18]. For two normalized vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we have that

$$(d_{\text{euclidean}}(\mathbf{x}, \mathbf{y}))^2 = 2d_{\cos}(\mathbf{x}, \mathbf{y})$$

This means that for two normalized vectors the square of their Euclidean distance is related to the cosine dissimilarity between the vectors by a factor of 2. For proof, see Appendix A.2.

2.1.2 Agglomerative hierarchical clustering

Consider a set of n observations $O = \{o_i\}_{i=1,...,n}$, where each observation contains m numerical attributes ($o_i \in \mathbb{R}^m$). Hierarchical clustering [19] is a method of grouping such a set into clusters $C_j, j = 1, .., k$ by linking together the observations with respect to some linkage criterion. The linkage criterion is a function of the distance metric or dissimilarity on the data points which determines how to recalculate distances between the clusters when two clusters have been merged, and as such is a measure of the distances between the clusters. An example of such a criterion is the single linkage criterion. Given two clusters X and Y, the single linkage method defines the distance between clusters X and Y as

$$\delta_{X,Y} = \min_{x \in X, y \in Y} d(x, y),$$

where d is the distance metric between the data points. The two closest data points between the clusters.

In the agglomerative hierarchical clustering method [20], each data point starts out as a cluster by itself, then the clusters are merged recursively according to an increasing linkage tolerance. At each step, the clusters with the smallest linkage between them are merged, following which the linkages between the newly formed cluster and all other clusters are calculated. When all data points have been merged into a single cluster, the algorithm terminates. As the algorithm proceeds, a natural hierarchy which specifies how data points and clusters are related to each other arises. An example of the agglomerative clustering method is illustrated in Figure 2.1.

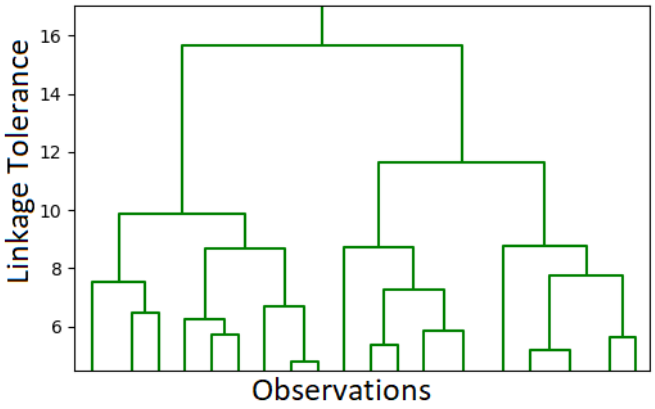


Figure 2.1: Hierarchical dendrogram for toy data using ward linkage.
Note that the dendrogram starts out with all observations separated and in their own respective cluster, and gradually combines them into clusters, represented as lines, as the tolerance for the linkage criteria grows. To identify k clusters, one would draw a line that crosses k vertical lines.

As such, hierarchical clustering requires no predetermined amount of clusters as input, but does not automatically generate the number of clusters in a data set either, and since it requires very little knowledge about the data prior to clustering, it is a prominent clustering method [19]. Given some metric or dissimilarity determining the distance between data points and a linkage criterion determining how clusters are to be linked in each step, the algorithm merely determines how data points are related (which can be visualized via a dendrogram as seen in Figure 2.1). One can then specify a tolerance determining where to draw a horizontal line through the dendrogram, which determines the number of clusters in the data set by the amount of vertical lines this tolerance line crosses. For example, had one drawn a straight horizontal line in Figure 2.1 corresponding to a linkage tolerance of 14, one would have found 2 clusters represented by the sub-dendrograms connected to each of the crossed vertical lines.

Ward's method [21], which was used to create the dendrogram in Figure 2.1, is a linkage method used for hierarchical clustering which at each step chooses clusters to be merged so that, after having been merged, the value of an error function is minimized. The choice of error function depends on the goal of the clustering. Ward's minimum variance method [21] is a special case of Ward's method, which uses an error function based on minimizing the increase in variance within clusters when merging. Let C_i and C_j be two clusters being merged to form the new cluster C_{i+j}. Ward's minimum variance method updates the distance between the newly formed cluster and every other cluster C to

$$\delta_{C_{i+j},C} = \sqrt{\frac{A}{T}d(C,C_i)^2 + \frac{B}{T}d(C,C_j)^2 - \frac{|C|}{T}d(C_i,C_j)^2},$$

where $T = |C| + |C_i| + |C_j|$, $A = |C| + |C_i|$, $B = |C| + |C_j|$, and d is the Euclidean distance function. As such, Ward's minimum variance is best suited for data sets on which one uses the Euclidean distance metric. However, as discussed in Section 2.1.1, the squared Euclidean distance is equivalent to the cosine dissimilarity for normalized vectors up to a scaling factor.

2.1.3 Dunn's index

Many clustering methods require either a predetermined number of clusters, or a tolerance for some linkage criteria to be provided to the algorithm. This means that the methods don't explicitly return an optimal clustering, and instead either require the sought amount of clusters be provided as a parameter or simply provides a hierarchical linkage structure of the data. Hierarchical clustering is an example of an algorithm which only returns a linkage structure of the data, leaving it up to the individual to determine how to interpret this linkage structure in terms of the number of clusters.

When utilizing such clustering methods, it is natural to have some index by which one can evaluate the resulting clusterings. By comparing this index for different clusterings containing different amount of clusters, one can determine that the amount which optimizes this index is the natural clustering of the data. One such index is Dunn's index, which provides a measure of the compactness of the clusters in a clustering in relation to how separated the clusters are from each other. By maximizing this metric, one arrives at an optimal amount of clusters for a given data set.

Let $C = \{C_i\}_{i=1,\dots,K}$ be a set of $K \geq 2$ clusters. Let Δ_{C_i} be the general compactness measure of cluster C_i and let $\delta(C_i, C_j)$ be the general separation measure between cluster C_i and cluster C_j. Dunn's Index for a clustering C clusters is defined [22] as

$$DI_K = \min_{1 \leqslant i \leqslant K} \left\{ \min_{1 \leqslant j \leqslant K, j \neq i} \left\{ \frac{\delta\left(C_i, C_j\right)}{\max_{1 \leqslant k \leqslant K} \Delta_{C_i}} \right\} \right\}.$$

This means intuitively that Dunn's index evaluates a clustering by dividing the separation measure of the closest two clusters by the compactness measure of least compact cluster. By finding a clustering that maximizes this measure, the clusters in that clustering would, ideally, be maximally compact and separated. Note that the semantic and numerical meanings of the concept of compactness are inversely correlated, meaning that when one talks about cluster with maximal compactness, the corresponding compactness measure is minimized.

The compactness measure Δ_{C_i} of a cluster C_i may be defined in several ways, depending on context. For example,

$$\Delta_{C_i} = \max_{x,y \in C_i} d(x, y)$$

takes the maximum distance found between two data points within cluster C_i as its compactness measure, as originally defined by Dunn [22].

The separation of clusters C_i and C_j may be defined similarly to the compactness measure of a single cluster, with

$$\delta_{C_i, C_j} = \min_{x \in C_i, y \in C_j} d(x, y),$$

which takes the minimum distance found between two data points within cluster C_i as the

separation of the two clusters, as originally defined by Dunn [22].

However, Dunn's Index handles data sets containing outliers badly, resulting in outliers having a disproportionate effect on the index, and therefore the clustering evaluation process. This prompts other definitions of the compactness and separation of the clustering [23]. For example, one may define the compactness measure of cluster C_i as

$$\Delta_{C_i} = \frac{1}{|C_i|(|C_i|-1)} \sum_{x,y \in C_i, x \neq y} d(x,y) \tag{2.3}$$

which measures the average distances between the points in cluster C_i, and the separation of clusters C_i and C_j similarly as

$$\delta_{C_i} = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d(x,y). \tag{2.4}$$

Hayes and Beling [17] successfully use a index, denoted here by $\widetilde{DI}$, similar to Dunn's index in their cluster analysis of single player strategies in the original StarCraft, with separation and compactness measure defined as averages. However, they extend the modification a step further by also defining their metric as the average compactness of the clusters divided by the average separation of the clusters, instead of the minimum and maximum functions employed by Dunn. This index therefore becomes

$$\widetilde{DI}_m = \frac{\frac{1}{|C|} \sum_{1 \leq k \leq m} \Delta_{C_k}}{\frac{1}{|C|(|C|-1)} \sum_{i,j \leq m, i \neq j} \delta_{C_i,C_j}},$$

where Δ_{C_k} and δ_{C_i,C_j} are defined as 2.3 and 2.4, respectively. Hayes and Beling noted that this index tends to evaluate clusterings containing a large number of small clusters more favorably than desired, necessitating a limit to the minimum acceptable size of a cluster. This limit is expressed as a percentage of the total amount of data points. Hayes and Beling used 10%.

2.2 Random forest

For the purposes of this thesis, classifying a game as a member of a cluster given information limited by the perspective of one player is necessary. This is a task that is non-trivial when the original clustering is done on complete information, since a new data point can't be classified simply by comparing it to clusters. For this reason, a separate classifier is required.

A random forest is a model that uses a multitude of decision trees for constructing a classification or regression model [24, p. 587]. To go into further detail, we must first properly introduce what a decision tree is, as well as a concept called bagging.

A decision tree consists of a root node, branching out into child nodes [24, p. 305]. Child nodes in their turn can also have children, and so forth. Nodes that have no children are called leaf nodes. When regarding classification, an input is passed to the root node. Each node then holds a test that is performed on an attribute in the input. A certain input will take a path through the tree until it reaches a leaf node. The leaf node then holds a class label, which is assigned to the input if it reaches that leaf node. An example of this is illustrated in Figure 2.2. The illustration means to intuitively explain how a decision tree makes its decisions.

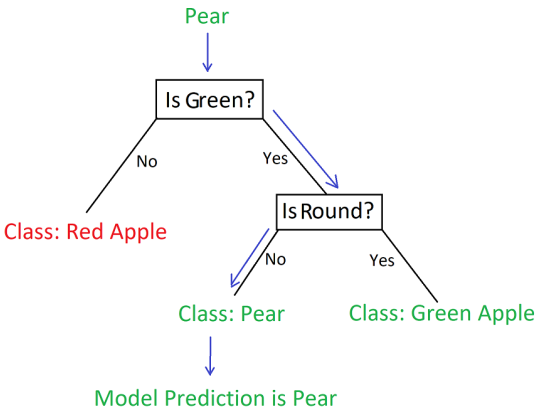


Figure 2.2: Illustration exemplifying how a decision tree works.
In this case there are three classes: red apples, green apples and pears. When the tree gets passed a pear as input, it tests its attributes to classify the input as a pear. The arrows illustrate what decisions the tree makes.

Bagging revolves around training a set of prediction models $\hat{f}_1(x), .., \hat{f}_B(x)$ on B separate training sets [24, p. 282-283]. Since data is often limited, splitting the data set into a multitude of subsets is generally not a practical approach. Therefore the training sets are typically produced by bootstrapping, which creates new data sets by sampling observations from a probability density function which is approximated from the given data. Random subsets of the bootstrapped data sets are used to create a variety of decision trees.

In the context of classification, the random forest model maps observations x to an output space consisting of classes k_i. The random forest relies on bagging to build decision trees $T_1(x), ..., T_B(x)$ from a training set $\mathcal{T}$. For an input x the output of the model is the class that received the majority vote across all the decision trees. This input-output relation is shown in Equation 2.5.

$$\hat{k} = \arg\max_{1 \leq i \leq B} \hat{f}_i(x) \tag{2.5}$$

An illustration of how a random forest makes classifications is illustrated in Figure 2.3, where n trees vote for classes A, B, C, D when given an input.

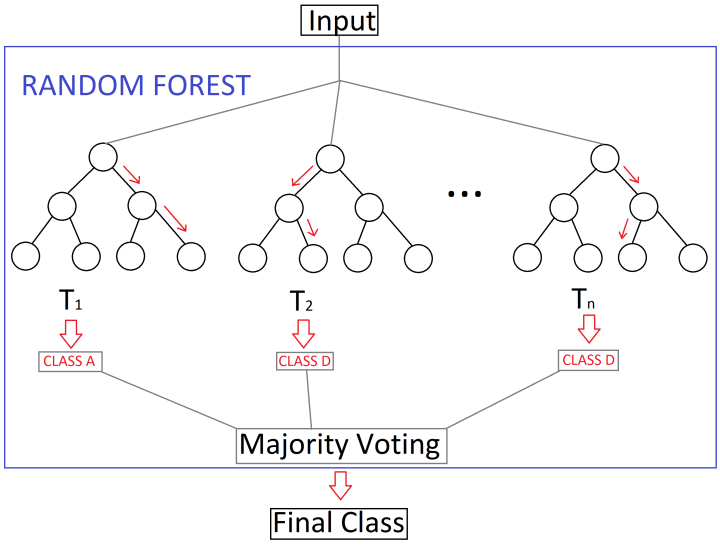


Figure 2.3: Illustration of how RF classification works.
Here, trees $T_1, ..., T_n$ each vote for classes A, B, C, D. The final output is the class which had the majority vote. Note that the input takes different paths through different trees as they are generated and trained in a way such that the correlation between the decision making- between trees is low.

When building a random forest classifier with the goal of solving a task where the input has parameters p, each tree only considers a random subset m of the parameters p, hence the name random forest. Different trees looking at different parameters aims to reduce correlation between the trees. Out of the parameters in m, only one is used for each specific split. Typically $m \approx \sqrt{p}$, i.e. the number of parameters considered in each tree is approximately the square root of the number of parameters. Each split aims to minimize some impurity function. One example of this is the Gini impurity, which is shown in equation 2.6 [24, p. 309].

$$I_G(p) = 1 - \sum_{i=1}^{K} p_i^2 \tag{2.6}$$

Here K classes are considered, and p_i are the fractions of labels in class k_i. The purpose of minimizing the Gini impurity function is to maximize information gain for each split, i.e. in the case of classification, separating classes with respect to one feature with minimum impurity.

By analyzing the decision trees that constitute a random forest it is possible to estimate the importance of features in terms of predictive contribution [25, Ch. 6]. In a decision tree, the nodes are organized as descending from the root based on the feature impurity score. Meaning, the splits performed by the highest nodes give the greatest decrease in impurity. Thus, the higher a feature is placed in a tree, the larger impact it will have on the final prediction. By averaging the feature importance for all trees, an estimated relative measure of feature importance can be obtained [26, Section 1.11.2.5.], [27].

2.3 Artificial neural networks

As with the random forest classifier described in Section 2.2, classifying a cluster is non-trivial due to the limitations set by the perspective of one player. To further investigate the possibilities concerning classification of games, a neural network can be employed.

An artificial neural network, (ANN), is a model that builds on a network of interconnected nodes that, given an input, produces an output in a way that is inspired by the way the human brain works. What is special with this model is that it can be trained to imitate any arbitrary function. The way this is done is by providing the model with examples of input-output relations from the target function that the model learns from. Once the model has been trained on this data, it may be capable of predicting correct outputs for inputs it has not seen before.

2.3.1 Neural network framework

To be more specific, an ANN consist of an input layer, followed by hidden layers and finally an output layer. The layers consist of nodes, and layers are connected by edges between their respective nodes. This framework is illustrated in a simplified manner in Figure 2.4.

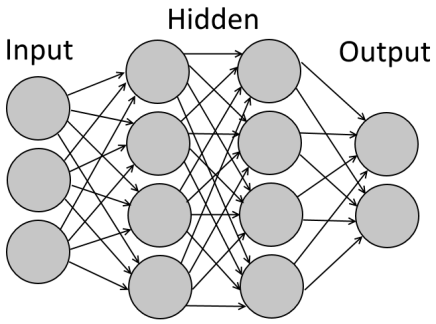


Figure 2.4: Illustration of an artificial neural network.
Here, the input layer consists of three nodes, the two hidden layers have four nodes each, and the output layer has two nodes. Each layer is fully connected to the next. Note that the model can have an arbitrary amount of hidden layers and nodes in each layer.

When a node receives inputs corresponding to the outputs from its predecessors, it computes a weighted sum from these inputs where each connection has an assigned weight. This weighted sum is then passed to an activation function to produce a new output from the current node, which is then passed to the next layer through the node's connections.

A popular choice of activation function for hidden layers is ReLU, which is defined in Equation 2.7. For output layers, the choices are more limited depending on the problem to be solved. For the purpose of classification, it is common to use the Softmax function, which is a layer-wide function that normalizes and converts the outputs from the previous layers into a probability distribution with a probability for each class to be the correct answer. The

class with the highest probability is chosen as the predicted class [28].

$$f(x) = \max(0, x) \tag{2.7}$$

2.3.2 Training the model

The basic idea behind training an ANN is that, given a specific input, the weights in the network are adjusted with regard to the error in the output relative to the ground truth.

To elaborate, consider an ANN solving a task that maps inputs $x \in X$ from a feature space to targets in an output space Y. Let us assume the following about the problem:

- There is an untrained model f containing some amount of layers, each with some amount of nodes.
- f is initialized with weights $W = \{w_{ij}^l\}$, corresponding to the weighted connections between node i in layer l and node j in layer $(l-1)$.
- There is a training set $\mathcal{T}$, consisting of input-output pairs (x_i, y_i), corresponding to true observations from the target function $\tilde{f}$.

To quantify the error, i.e. how wrong the model output is for a specific input, a cost function $C : F \longrightarrow \mathbb{R}$ is defined such that the optimal model $\tilde{f}$, with regard to solving the task, minimizes the cost function [29]. It follows that

$$C(\tilde{f}) \leq C(f), \forall f \in F.$$

The learning process proceeds by adjusting the weights w_{ij}^l, with the goal of minimizing $C(f)$ with regard to our training set $\mathcal{T}$. This is done by iteratively updating the weights in a way that reduces $C(f)$, a procedure known as backpropagation [29]. Mathematically, the way a weight w_{ij}^l is updated to ${w_{ij}^l}^*$ is

$${w_{ij}^l}^* = w_{ij}^l + \eta \frac{dC}{dw_{ij}^l}$$

where η is a parameter that tunes the learning rate, i.e. how much the weights should be adjusted [30]. Each sample in the training set is used for updating the weights once, until all samples have been used. The entire process, called an epoch, is then repeated a predetermined number of times [29].

Ideally, iteratively backpropagating the cost $C(f)$ and adjusting w_{ij}^l to reduce $C(f)$ by inputting $\mathcal{T}$ into our model will fit the model in a way that $C(f)$ is minimal and f is a good approximation for solving the task at hand.

2.3.3 Batch size and early stopping

The backpropagation algorithm utilizes steepest descent to find the minimal value for the cost function [29]. This is a method commonly used to minimize a function, where each variable is iteratively updated proportionally to the function derivative with respect to the variable. The update is done in the direction that lowers the function according to the derivative. This process is repeated until the derivatives approach zero, at which point a minimum value is found. Steepest descent inherently guarantees that the function never increases, leading to a risk of becoming stuck in a state that does not correspond to the global minimum of the function, as seen in Figure 2.5. A steady state like this is called a local minimum, meaning that the function increases in all directions around a point but the function still reaches a lower value somewhere else.

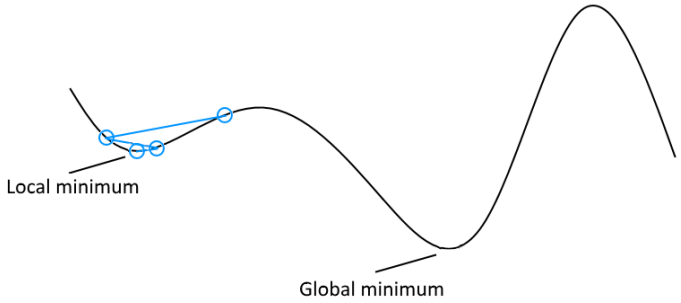


Figure 2.5: Illustration of steepest descent.
Local and global minima are shown in the figure. The blue circles and lines denote a path that could be taken by the steepest descent algorithm, leading to the process becoming stuck in a local minimum.

This risk can be mitigated by adding a stochastic element to the process, in order to remove the constraint of never increasing the cost function with an update [30]. The most common way to achieve this is to update the weights of the model with respect to a small number of randomly chosen data points at a time, instead of the entire data set. This method is referred to as batch training, and is almost always applied in neural networks [31]. The size of the groups of data points, called mini-batches, can be configured before the training process begins. When batch training, the gradients are therefore calculated with respect to a few data points, instead of the entire data set. As such, the steepest descent term is not guaranteed to lower the value of the cost function for each batch of data points.

Another issue when training neural networks is knowing when to stop the training process. Assuming that the accuracy of the model approaches its practical limit with every iteration, or fluctuates around some value due to the learning rate being too large, running the model for a previously set amount of epochs may not be necessary. To avoid training the model for longer than needed, early stopping can be implemented [32]. Early stopping is a method for stopping the training process when the performance of the model is seemingly stagnant between iterations. This algorithm has two parameters: the delta below which an improvement is regarded as stagnant, and the number of stagnant epochs required to terminate early. These parameters are usually denoted as stopping delta and stopping patience.

2.3.4 Hyper-parameter optimization

For any given neural network problem, finding the network layout that maximizes model accuracy can be a difficult task. Selecting between the various types of network, as well as the number of layers and type of each layer, are examples of non-trivial choices that need to be made during the process of constructing a neural network. Beyond the layout of the model, several other internal parameters have an effect on model performance [33]. These parameters, referred to as hyper-parameters, include, but are not limited to:

- Learning rate
- Batch size
- Number of epochs, i.e. training time
- Activation function for each layer
- Early stopping delta and patience

Finding the optimal hyper-parameters can be done by advanced search algorithms, but may take extensive training time as each configuration of hyper-parameters corresponds to a model that needs to be trained and evaluated individually in order to perform comparisons [33].

2.3.5 K-fold validation

When training a statistical model such as a neural network, the data set is usually split into a training set and a validation set [34]. The model is trained using only the training set, and the predicted real world accuracy is calculated by classifying the validation set with the trained model. In neural networks, hyper-parameters are often tuned to maximize the validation accuracy for each generated model [35].

To further validate the resulting model, a test set can be used. This data set should be completely separate from the training and validation sets, and is used to predict the real world accuracy with a finalized model. The predicted accuracy for the test set will represent the real world performance of the model more accurately than the training or validation sets, since the test set accuracy is measured but not optimized towards [35].

In a limited size data set, splitting the data into training, validation, and test sets can result in data sets that are too small to accurately represent the distribution of the full set. This can be mitigated by using k-fold cross validation, a process during which the original data set is divided into k equally sized subsets. The subsets are then iterated over, with each subset being used as a validation set once, with the rest of the data set used as training data during that iteration. One model is trained per subset, resulting in a set of k different models. The accuracy of these models can then be averaged to produce an approximate prediction of the real world performance of the network, without the need for a separate test set [34].

3

Methods

In answering the core questions specified in the problem statement, the method is structured the following way: data retrieval, cluster analysis, classification, and further classification analysis.

3.1 Retrieval and preprocessing of data

The first step of conducting the data analysis of StarCraft II games was to gather and preprocess large amounts of replay data from Blizzard's official replay hub for scientific research [36]. Each replay contains all user inputs throughout a single game of StarCraft II. sc2reaper [1], utilizing pysc2 [37], was used to generate game states of parsed games. To represent a game state, all game pieces of both players were stored at each 30 second interval.

Additionally, the vision each player has of their opponent's pieces was also parsed every half second. For the clustering process the entire game state was used, but for the classification process one player's half of the game state in combination with that player's vision was used. This combination is equivalent to the player's perspective.

3.1.1 Replay filtering criteria

Before any parsing was done, certain criteria were filtered upon. In order to be a part of the final data set, a replay needed to meet all of the following conditions:

The game length of a replay must be between one and 33 minutes. These game lengths were decided upon as games outside of this interval were deemed as not representative of regular gameplay of StarCraft II. Games shorter than one minute were considered to have been interrupted due to extraordinary reasons and therefore not relevant for this study. Similarly, games longer than 33 minutes were judged as being statistical outliers. See Appendix A.3 for a plot of games remaining over time.

The replay must feature the match-up Protoss versus Protoss and have exactly two players on separate teams where neither of them is an AI actor. Additionally, one of the players is required to have a matchmaking rank roughly equivalent to the top 5% of StarCraft II players [38]. The reasoning behind only requiring one player to have an adequate rank is

to cover the case where one of the two players is playing their first game of the season and therefore is yet to have a rank.

The replay must be of a game version within the accepted patch interval: 4.2.1 to 4.6.2. This version interval was chosen as it is recent, 2018 [39], and as it spanned a relatively long time interval without introducing large enough game design changes that they could meaningfully affect gameplay [39].

3.1.2 Vision parsing

For each player, vision of opposing game pieces was parsed. Two different methods of vision parsing were performed, which in this thesis are referred to as the weak and strong vision parsing.

The partial game states of both the vision parsing algorithms were calculated using the currently visible units attribute within the data produced from sc2reaper. Every half second, and from the view of each player, the system recorded the currently visible game pieces. At each 30 second interval, at which point the complete game states were saved, the corresponding vision from each player was also compiled. How the two vision algorithms differ lies in how they gather and transform the vision data between these 30 second intervals.

The weaker parsing algorithm, for each player, keeps track of the number of game pieces that have been seen under the opponent's control. Those numbers are calculated by recording the largest amount of each game piece under the opponent's control a player has ever seen at a singular point in time.

Subsequently, the stronger parsing algorithm uses the built-in identification numbers of each individual game piece in order to determine whether an observed game piece is a new individual or a previously seen one. Consequently, it is able to assess the number of each game piece controlled by the opponent more accurately than the weak algorithm. This algorithm also subtracts from this number whenever a game piece controlled by the opponent is destroyed.

3.1.3 Mathematical and statistical preprocessing

Initially, each game G_i was numerically represented as a set of vectors $O_i^t = \{x_1^t, x_2^t, ..., x_n^t\}$ consisting of the numbers of the game pieces x_j^t, $j = 1, ..., n$, controlled by both players combined at time t. In other words, each vector O_i^t represented the game state of game i at time t. Since the game state was sampled every 30 seconds, each game G_i was represented as a set of game state vectors so that

$$G_i = \{O_i^t\}_{t=0,30,...,T_i},$$

where T_i is the final time at which the game state for game G_i was sampled.

In order to represent the time development of a game G_i up to time t as a single vector, the observation vectors $\{O_i\}_{i=0,30,...,t}$ were concatenated such that each game was represented at each time t by

$$g_i^t = (O_i^0, O_i^{30}, ..., O_i^t),$$

$\forall t \leq T_i$. This way, the time development of each game was represented by those concatenations for all sampled time points up to and including the end time of the game. Such a vector is called a time series of a game, and each game was represented by a set of such time series $\forall t = 0, 30, ..., T$ s. Therefore, each time t has an associated set of such time series for all games surviving to that time point, which are the data sets being clustered upon, resulting in a distinct clustering for each time $t = 0, 30, ..., T$ s. For an illustrative example of this, see Appendix A.4.

A consequence of this method of concatenating observation vectors to form time series was that the number of observations in each data set decreased with time, since games terminate and drop off as time increases. In contrast, as the number of components in each time series increased, one extra time instance of each game piece was added to every time series for each time point. Therefore, these time series were only computed up to and including $t = 870$ s, as the size of the vectors became too big in comparison to the amount of games that survived past that point.

For each such set of time series, each component M of each time series vector $\tilde{G}$ was scaled according to

$$M' = \frac{M - \min(M)}{\max(M) - \min(M)},$$

where $\max(M)$ denotes the maximum value of component M across all time series in the data set, and $\min(M)$ denotes the minimum value of component M across all time series in the data set. This resulted in each component being scaled so that they take values between 0 and 1, rendering all the game pieces adapted to the same scale. This was done in an effort to ensure that no single game piece dominates the game vector purely by attaining higher absolute values than other pieces, and so that the overall shape of the time evolution of each piece is the primary information conveyed by the time series vector rather than the specific values they attained.

3.2 Cluster analysis

Since cluster analysis can be considered an exploratory science in that it almost always requires some experimentation, the method arrived at in this study was the accumulation of knowledge acquired by such experimentation over time. Including a description of every non-successful clustering method that was tried was deemed to be of little interest to this report, since the research question posed in a previous section was explicitly stated to merely find an appropriate method of clustering as opposed to comparing several different methods and arriving at an optimal method. Sci-py's [40] implementation of Hierarchical clustering with Ward's minimum variance linkage was chosen as the final method of clustering, since it was deemed appropriate based on the results it produced. It offered several advantages over other attempted clustering methods, such as the ability to choose linkage types, distance metrics, and the fact that it can be utilized on a distance matrix as opposed to the full data points, as well as dissimilarities.

3.2.1 Method of clustering

For each time $t = 0$ s, 30 s, ..., 870 s cluster analysis was performed on the set of time series up to that time, as detailed in Section 3.1.3. In this way, clustering was performed on the complete evolution of each game up to that time point. Sci-py's implementation of agglomerative hierarchical clustering as introduced in Section 2.1.2 was used with Ward's linkage criterion, shown in Equation 2.1.2.

3.2.2 Determining the number of clusters

Since hierarchical clustering does not by itself compute the optimal amount of clusters in a data set, one must evaluate several clusterings for and somehow determine the optimal clustering. This was done via a modified version of Dunn's index. For a clustering C containing m clusters, the modified Dunn's index is defined as

$$\widetilde{DI}_m = \frac{\frac{1}{|C|(|C|-1)} \sum_{i,j \leq m, i \neq j} \delta_{C_i,C_j}}{\frac{1}{|C|} \sum_{1 \leq k \leq m} \Delta_{C_k}}.$$

In this index, the average separation δ_{C_i,C_j} betweenf all clusters is divided by the average compactness measure Δ_{C_i} of all the clusters, in accordance with Hayes and Beling [17]. To find a clustering which maximizes this index the clusters must be maximally compact and separated with respect to the index, which makes the index a suitable method of evaluating clusterings.

The compactness measure Δ_{C_i} of a cluster C_i was chosen as

$$\Delta_{C_i} = \frac{1}{|C_i|(|C_i|-1)} \sum_{x,y \in C_i, x \neq y} d_{\cos}(x, y)$$

and separation of two clusters C_i and C_j was defined as

$$\delta_{C_i,C_j} = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d_{\cos}(x, y),$$

where $d_{\cos}(x, y)$ is the cosine distance metric expressed in Equation 2.2. This modification of Dunn's index was also made in accordance with the index utilized by Hayes and Beling [17], the only difference being that the cosine dissimilarity was used in place of the Manhattan distance. This modification was made in an effort to achieve a more fair evaluation of clusterings where the clusters are not completely separated but still should be considered distinct clusters. Since the original Dunn's index used minimum and maximum distances, it may not fare well when dealing with more overlapping clusters.

3.2.3 Minimum cluster size

To counteract the tendency of the modified Dunn's index to favor clusterings containing larger amounts of smaller clusters, it was necessary to determine a minimum acceptable size

of each cluster. This limit was set to 5% of the size of the total data set, i.e. if a clustering resulted in one or more clusters containing less than 5% of the total data points it was discarded. This also sets the highest possible number of clusters in one clustering to 20. As a result, for the set of time series at each point in time, clusterings containing 1 to 20 clusters were formed. For each time point, the clustering which fulfilled the minimum cluster size criterion and resulted in the maximal modified Dunn's index was determined to be the optimal clustering for that time point.

3.2.4 Choice of distance metric

For the clustering, the cosine dissimilarity was used as a means of reducing the susceptibility of conventional distance metrics, such as the Euclidean distance, to break down in high dimensional spaces [41]. Additionally, this was done to also focus the clustering to the general shape of the time series of the vector components in an effort to cluster with respect to game state time series, rather than specific values at certain individual times. It was assumed that it mattered less which specific value a certain game piece attained in a game than that the game piece was found at all in that game, to which the cosine dissimilarity lends itself.

3.3 Development of classifiers

The classification step of the process proceeded in two directions: artificial neural networks and random forests. The goal of this step was to train and develop classifiers able to classify which earlier defined clusters of game state time series a new game should belong to.

Both artificial neural networks and random forests were studied as the models differ in terms of which problem domains they perform well in. The random forest classifier performs better in problem domains where data is limited and the structure in the data is weak [42]. If there is a definite structure, however, artificial neural networks can achieve superior performance [42].

Evaluation of the classifiers, with the purpose of finding suitable configurations, was done based on several optimal clusterings found in the cluster analysis step. The absolute performance was deemed irrelevant at this stage, and the configurations were instead chosen based on relative performance.

3.3.1 Development of the neural network classifier

To implement a neural network, the Python library Keras [43] was used together with the TensorFlow [44] backend. A few different configurations for the neural network were tested in an arbitrary manner to find a suitable but not necessarily optimal configuration for solving the problem.

A deep neural network layout with two hidden layers was decided upon, consisting of 8 and

4 nodes respectively. The hidden layers used the ReLU activation function, and the output layer used the Softmax activation function. The remaining hyper-parameters of the model, and training process, were chosen as follows:

- Initial learning rate of 0.0005
- Batch size of 64
- A maximum of 200 epochs.

The model was also defined to use early stopping, with a stopping delta of 0.02 and a patience of 2. The exact layout of the model, as well as the configuration of hyper-parameters, was arrived at through an exploratory process. During this process, several configurations as well as types of networks were investigated. The final decision was based on accuracy and simplicity of the model. Further motivation can be found in Section 5.2.3.

Following the initial model setup, training was done using k-fold cross validation. The number of folds chosen for the training process was 6, in order to produce a result deemed statistically secure while keeping the test data at a large enough size. With a total data set of 8000 replays for the chosen clustering, this led to a training set size of around 6600 and a validation set size of around 1400. The 8000 replays consisted of the replays that were at least 270 seconds long and part of the top $10,000$ replays.

3.3.2 Development of the random forest classifier

For the random forest classifier, the scikit-learn [26, Chapter 1.11.2.] [27] implementation was used. The Gini impurity, which was described in Section 2.2 and expressed in Equation 2.6, was used as a measure of impurity for making splits when building the trees.

To determine a suitable number of trees for the random forest, the model classification error as a function of the number of trees was studied, as shown in Figure 3.1. Here, the model classification error was defined as $1 - A$, where A is the model accuracy. The number of trees was chosen such that the model classification error was low and stabilized. For this, the complete information game state time series data was used for the classification error estimation.

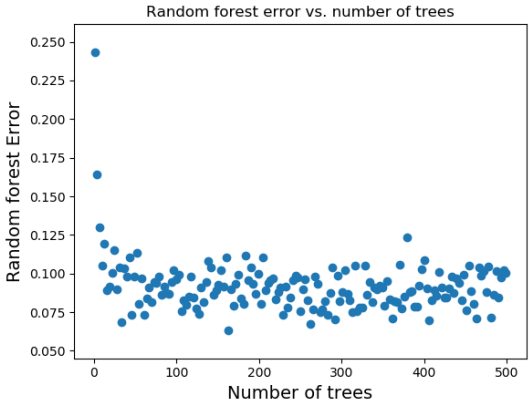


Figure 3.1: Scatterplot of number of trees in Random Forest.
The relation between number of trees in the Random Forest when using the Gini impurity from Equation 2.6, and its error when classifying on the data with the labels assigned from the clustering. Observe that the error flattens after approximately 100 trees. As stated earlier, which specific clustering was used to generate this graph bears no relevance, as it is only used to decide upon a suitable number of trees for the classifier.

Observing that the model error flattens out around about 100 trees in Figure 3.1, 300 trees was chosen as the model parameter to have good margins without making the model unnecessarily complex.

3.4 Analysis of classifier performance

The trained and finalized classifiers were developed to accurately classify provided data containing only the information available to one player up to a specific point in time. A crucial part of understanding the results of the performance of the classifiers was therefore to analyze the accuracies of the classification models, i.e. how often the models made the right prediction in a certain situation. To conduct this analysis, visualized testing data was generated using the methods described below.

3.4.1 Classification with different data sets

To present and analyze the overall performance and classification accuracies of the models, as well as to understand how much information was contained in the different data sets, the performances of the models were tested on data sets which contained different information.

The accuracies were produced by averaging several k-fold cross validations, explained in Section 2.3.5, for the neural network, and by averaging multiple random partitions of 75% training data and 25% validation data for the random forest. The accuracies were measured using complete data of one player's pieces along with varying information of the opponent's

game pieces: no data, weak vision, strong vision, and complete data. The final models were used to generate these accuracies.

3.4.2 Classification with different clustering times

The classification accuracies of our models over different clustering times were studied. This was done in relation to the optimal number of clusters at each point in time determined by the cluster analysis.

Each accuracy A was adjusted in relation to the *baseline* B, corresponding to the proportion of data in the largest cluster, and mapped to $[0, 1]$ by scaling as shown in Equation 3.1. The baseline is considered as a floor for the classifiers, because the training process generally guarantees that if the accuracy of the classifier given the available information is below the baseline, the best strategy is to always predict the largest class.

$$A_{adjusted} = \frac{A - B}{1 - B} \tag{3.1}$$

This measure of accuracy describes how well the model can classify in relation to baseline. The adjusted accuracies were plotted as a function of up to which point in time the clustering was performed. To evaluate the models, the complete game state time series data was used.

3.4.3 Classification with partial data

In order to evaluate how well the classifiers could assign games to their respective clusters, the classifiers were trained and validated on different subsets of the full data set. These subsets corresponded to the concatenation of the sampled game states up until some time, i.e. the classifiers were provided with the time series of the game state, until some cut-off. This was done for both the complete information data, as well as data limited by the player's perspective.

For each cut-off, the classifiers' accuracies were validated by classifying on the labels provided by the previously determined optimal clustering time point. Validation was done using k-fold cross validation for the neural network, and by averaging multiple random partitions of 75% training data and 25% validation data for the random forest. The final cut-off was chosen to be later in time than the time point at which the optimal clustering was performed. To visualize the results, the adjusted classifier accuracies were plotted as a function of the last time point included in the data provided to the classifiers.

3.5 Feature importance analysis of clusters

In addition to being an accurate prediction tool for classification, random forests can be employed to evaluate which features are the best predictors [25, Ch. 6]. Thus, the random

forest classifier was also employed as a black box metric to investigate feature importance of the clustering.

The feature importances for the random forest classifiers were extracted from the model using a standard method from scikit-learn [26, Chapter 1.11.2.5.], [27]. In order to distinguish inter-cluster differences in terms of important features, where each feature corresponds to the number of a certain game piece at a particular time, each individual cluster was evaluated compared to a single cluster composed of all other clusters combined. Essentially, this corresponded to using binary labels indicating whether or not a game belonged to the cluster being studied. This method produced the feature importances for each game piece, at each time point. It thereby implicitly provided information regarding which features primarily differentiate the clusters from each other.

This type of labeling and evaluation was performed for each cluster. Since the feature importances were specific to particular time points, it was necessary to sum the feature importance over time for each game piece, resulting in an overall importance of each game piece. Furthermore, the feature importances at each time over all game pieces were summed, resulting in a measure of the overall importance of each time point. Note that these importances were only compared on a relative basis, and that the absolute values of the importances were of no interest.

In order to analyze these importances further, the average value at each time point of the game pieces which were deemed most important according to their summed feature importances were plotted for each cluster. This resulted in a set of plots which showed how these game pieces on average developed over time for each cluster, enabling an analysis of exactly how they act to differentiate the clusters from each other.

4

Results

This chapter presents the results of this project, using the following structure: results of clustering, results of classification, and results of the feature importance analysis.

4.1 Results of clustering

For each time $t = 0, 30, ..., 870$ s, hierarchical clustering with Ward's minimum variance method was performed on the set of time series $\{g_i^t\}$ of the games up to that time t, as defined in section 3.1.3.

The result of clustering for times $t = 0, 30, ..., 870$ s and choosing the optimal clustering with regard to the modified Dunn's index is shown in Figure 4.1. The graph illustrates the number of clusters found for the optimal clustering and the corresponding modified Dunn's index, where both lines are functions of the clustering time.

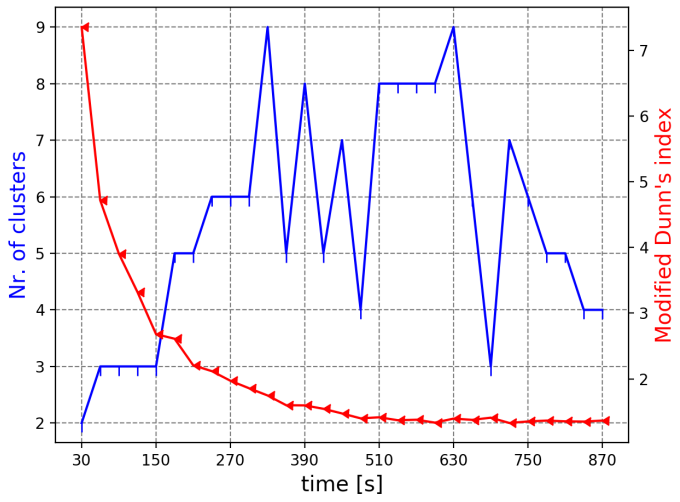


Figure 4.1: Optimal number of clusters and modified Dunn's index over time. *Recall that a greater value for the modified Dunn's index corresponds to a better clustering. The red triangle dotted line shows the modified Dunn's index for the optimal clustering at that time. The blue dashed line illustrates how many clusters are in this optimal clustering.*

Denote the optimal number of clusters at time t by m_t. At $t = 0$ s no clustering reached the criterion of each cluster containing at least 5% of the data, due to the fact that all games start out identically, leaving a single cluster containing all data points and m_0 undefined. As t increases, one can see that the optimal number of clusters generally increases from $m_{30} = 2$ to $m_{300} = 6$, after which m_t seems to behave in a more unpredictable manner, before receding to $m_{870} = 4$. The corresponding modified Dunn's index also decreases with time, starting at $\widetilde{DI}_2^{t=30\text{ s}} = 7.35$ to $\widetilde{DI}_4^{t=870\text{ s}} = 1.37$.

Figure 4.2 shows the modified Dunn's index for different clusterings at different times. The clusterings are presented as a heat map, and all clusterings shown fulfill the criterion of containing at least 5% of the data.

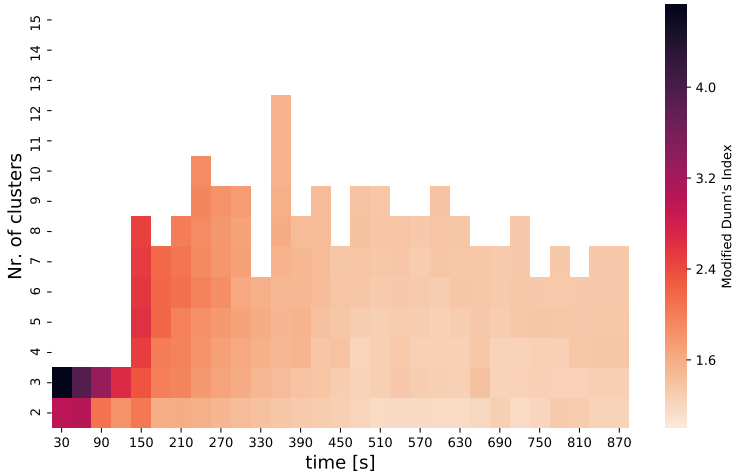


Figure 4.2: Heat map of the modified Dunn's index for each clustering over time. *The clusterings included in this graph are those that fulfill the criterion of each cluster containing at least* 5% *of the data. Note that the index decreases for all clusterings as time progresses.*

Observe that the optimal clustering, corresponding to the highest modified Dunn's index at a certain time, is more distinct for earlier time points. This is indicated by the greater difference in modified Dunn's index between different clusterings. This implies that the most notable divergences in game states occur in the earlier part of the game. For later time points, the optimal number of clusters becomes more arbitrary, as there is no substantial difference between the index for different clusterings at these times. In contrast to the earlier clusterings, this indicates that the games tend to become more homogenized towards later time points.

Judging by the trend in optimal number of clusters observed in Figure 4.1, combined with the motivation regarding homogenization of games at later time points as seen in Figure 4.2, an appropriate time point for an extended analysis of the clusters is determined to be $t = 270$ s.

Figure 4.3 shows a dendrogram of the clustering deemed optimal at time $t = 270$ s by the modified Dunn's index. The horizontal line corresponds to the tolerance for which the

clustering is extracted.

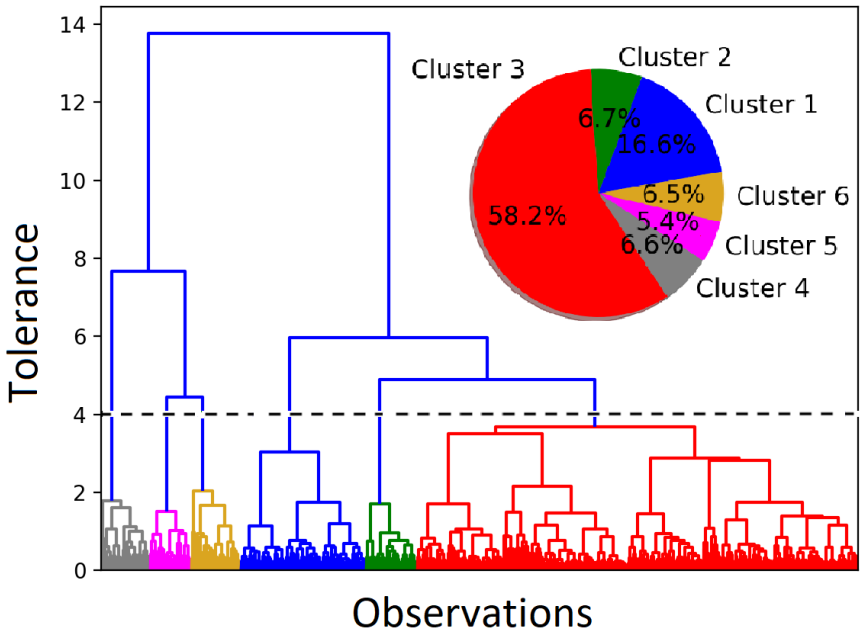


Figure 4.3: Dendrogram of the optimal clustering at $t = 270$ s, coupled by a pie chart indicating the relative sizes of the clusters.
The dashed line shows the linkage tolerance which resulted in the highest modified Dunn's index, $\widetilde{DI}_6^{t=270\ s} = 1.97$. *As a result of the dashed line crossing 6 vertical lines, the data set is clustered into 6 clusters, indicated by different colours.*

Observe that the horizontal line crosses 6 vertical lines, corresponding to the clusters that are identified. Also, note that cluster 3 contains a majority of all games, while the other five clusters are much smaller in terms of size.

Figure 4.4 shows a visualization of the optimal clustering from time $t = 270$ s, generated via multidimensional scaling, (MDS). In this case, MDS visualizes the high-dimensional data by trying to place the games in a three dimensional space. This is done in a manner such that the computed cosine dissimilarities are optimally preserved with respect to the true distances [45, p. 1]. The MDS was implemented via Scikit-Learn [46]. Observe that the clusters are distinct and somewhat well separated. This is an indication of the quality of the clustering, but one must have in mind that MDS cannot guarantee that the true distances are preserved fully. However, since MDS is used only as a visualization tool in the report, this fact bears no importance to the results of the report.

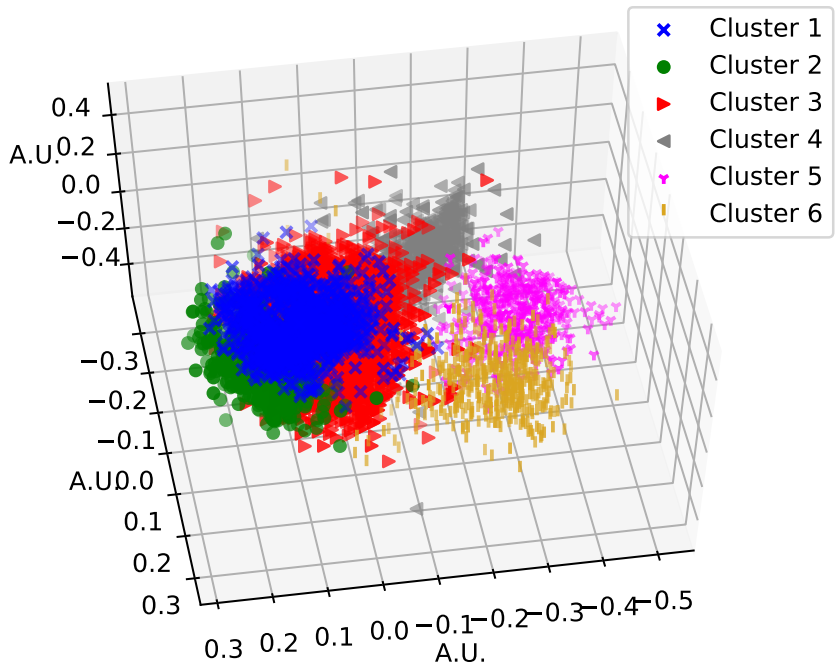


Figure 4.4: Optimal clustering at $t = 270$ s.
Further investigation at $t = 270$ s in particular is warranted via the modified Dunn's index. The data is visualized using multidimensional scaling, (MDS). A.U. indicates that the axis units are arbitrary.

4.2 Performance of classifiers

In this section, the respective performances of the neural network and the random forest classifiers are presented.

4.2.1 Classification on different clusterings

The classifier accuracies adjusted for the baseline are presented in Figures 4.5 and 4.6. Here, the target labels correspond to the optimal clusterings for times $30, ..., 870$ s, and the classifiers are provided complete information on both players. Note that the different optimal clusterings have varying numbers of clusters.

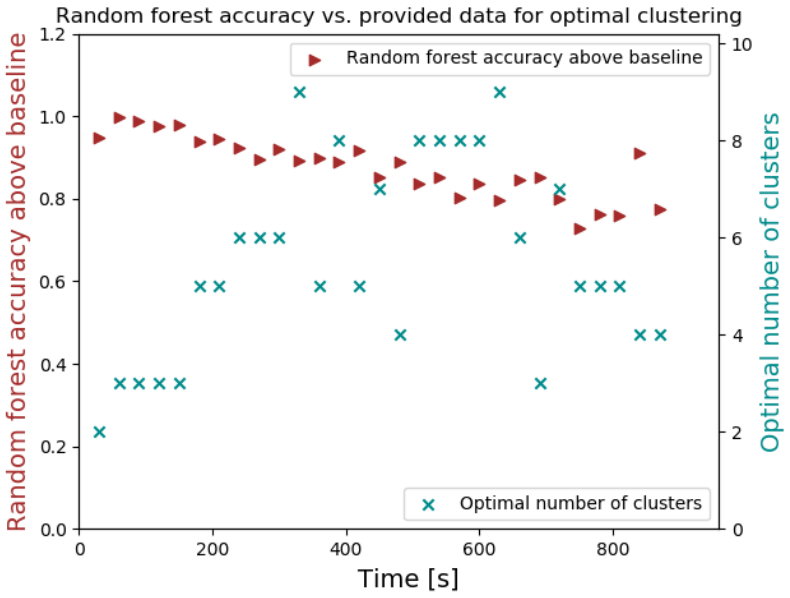


Figure 4.5: Scatterplot of random forest classification accuracies for optimal clusterings at times $t = 30, ..., 870$ s.
The classification accuracies are adjusted for the baseline accuracies, for different clusterings, marked with red arrows. The blue x:s show the number of optimal clusters at the discrete times, corresponding to the results presented in Figure 4.1.

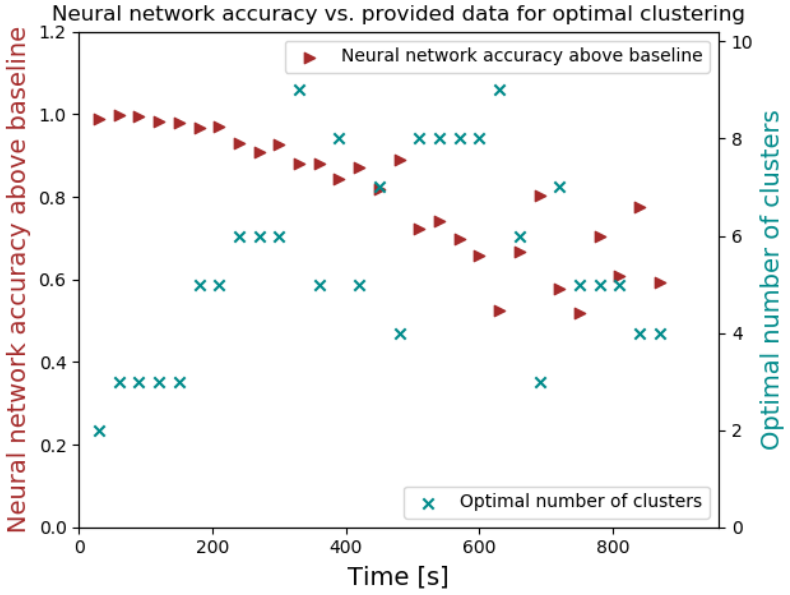


Figure 4.6: Scatterplot of neural network classification accuracies for optimal clusterings at times $t = 30, ..., 870$ s.
The classification accuracies are adjusted for the baseline accuracies, for different clusterings, marked with red arrows. The blue x:s show the number of optimal clusters at the discrete times, corresponding to the results presented in Figure 4.1.

It is apparent that the adjusted accuracies of both classifiers decrease as time progresses. Another observation from the Figures 4.5 and 4.6 is that the accuracy of the neural network classifier is less stable in comparison for times beyond 400 seconds. This is additional motivation for the choice of an early clustering time for further analysis.

4.2.2 Classification of optimal clustering

The accuracies of the classifiers when trained and validated on the clustering from time $t = 270$ s with different combinations of full and imperfect information for the respective players is presented in Table 4.1. The items in the table represent mean values from ten runs, coupled with an uncertainty of one standard deviation.

	1	1+V	1+V*	2
ANN	0.8229 ± 0.0120	0.8950 ± 0.0082	0.8959 ± 0.0085	0.9552 ± 0.0049
RF	0.8547 ± 0.0006	0.8960 ± 0.0008	0.9080 ± 0.0004	0.9627 ± 0.0006

Table 4.1: Table of classifier accuracies using different data sets.
1 and 2 denote complete information on one and two players respectively, V denotes weak vision information and V denotes strong vision information. 1+V and 1+V* indicate complete information of one player's game pieces and the partially complete information from vision of the other player's game pieces.*

From the results presented in Table 4.1 it is evident that the complete information carries more information than the vision information. Also, observe that vision information improves the classification accuracy in relation to only providing complete information about one player. Additionally, the accuracy using strong vision is potentially slightly more accurate than using weak vision, but the margins are slim. Complete information about both players leads to the best classification, but has no applicability in a live game as only one player's perspective is available.

4.2.3 Classification accuracy with partial data

The performance of the classifiers as a function of how much of the data in terms of time development the model is provided with is presented in Figure 4.7. Here, complete information is provided, and labels are assigned in accord with the clustering from time $t = 270$ s.

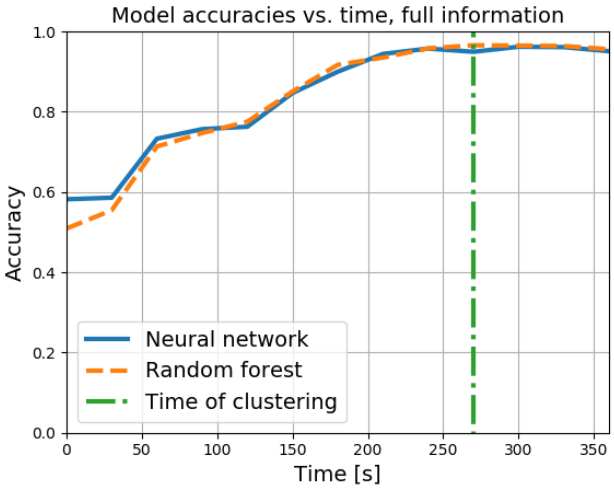


Figure 4.7: Illustration of classifier accuracies on complete information data over time.
The fully drawn blue line and the dashed orange line correspond to the neural network and random forest classification accuracies respectively. The green dash-dot-line indicates the time, $t = 270$ s, of the clustering which provided the labels. Note that both classifiers reach their peaks around the time where the clustering is performed.

Note that the accuracies of the models in 4.7 start much higher than chance even though no information is provided. This is because the largest cluster contains a majority of the games. The models, given no further training data, should always classify games as members of the largest cluster, resulting in accuracies above chance. Note, however, that the random forest classifier performs worse than the baseline for the earlier time points.

It is also evident that the model accuracies for both the random forest and the neural network increase well before the time at which the clustering is performed, and then top out around the time of clustering. This has the implication that the clusters are meaningful in the sense of being formed over time, and not just describing the game state at a specific time point, while future information doesn't provide any good predictive features as the accuracy ceases to increase.

The results of validating the performances of the classifiers with imperfect information in the same composition as above are shown in Figure 4.8. To clarify, the information provided is $1 + V^*$, i.e. complete information of one player's game pieces and partial information from the strong vision algorithm of the other player's game pieces.

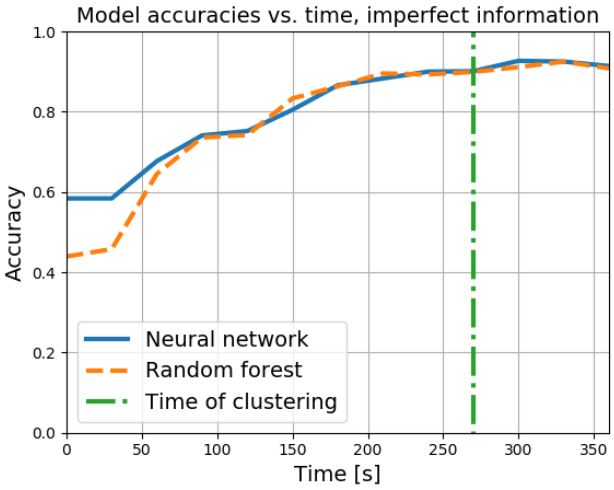


Figure 4.8: Illustration of classifier accuracy on data from a player's perspective over time.
The fully drawn blue line and the dashed orange line correspond to the neural network and random forest classification accuracies respectively. The green dash-dot-line indicates the time, $t = 270$ s, of the clustering which provided the labels. Note that both classifiers slightly increase their classification accuracies even after the time of clustering, unlike what is seen in Figure 4.7.

In Figure 4.8, one can observe a similar evolution of the classification accuracies as a function of time as in Figure 4.7, but two differences are noted. Firstly, the highest classification accuracies of the models are lower than they are with the complete information, as assessed in Section 4.2.2. This is a result of the vision information not being as strong of a predictor as the complete information. Secondly, the accuracies for both classifiers seem to continue increasing slightly even after the time of clustering. This can be explained by the possibility of players observing game pieces after the time at which the clustering is performed, granting further predictors for determining what cluster a game belongs to.

4.3 Feature importance analysis

In this section, the results from the feature importance analysis for the clusters are presented. This includes results showing which game pieces are the distinguishing pieces for each clusters, as well as an analysis of how they develop on average in the games of each cluster. The in-game roles of the specific game pieces within StarCraft II that are mentioned in this section are not important for this analysis. What is important is the fact that clusters are differentiated from each other by different game pieces, and at which point in time they are acquired by the players. If additional information about a specific game piece is required it is presented in the text. The results presented in this section correspond to the clustering at time $t = 270$ s.

4.3.1 Feature importance analysis of clusters

In Figures 4.9 - 4.14, the results of summing the feature importances for different game pieces over time are illustrated. Recall that there is a distinction between feature importance and game piece importance. Feature importance is the random forest importance of a single vector component, i.e. the numerical value of a game piece at a particular point in time. As such, it can be considered a measure of how important a certain vector component is for the classification made by the random forest classifier. Game piece importance is defined as the total summed feature importance of the vector components belonging to a certain game piece, i.e. the sum of the importances of the game piece at particular points in time. The game piece importances within each cluster are presented as bar graphs. Each bar graph shows the ten game pieces with the highest importances for that cluster.

The top ten importances of game pieces for clusters 1, 2, and 3 are shown in 4.9, 4.10, and 4.11, respectively.

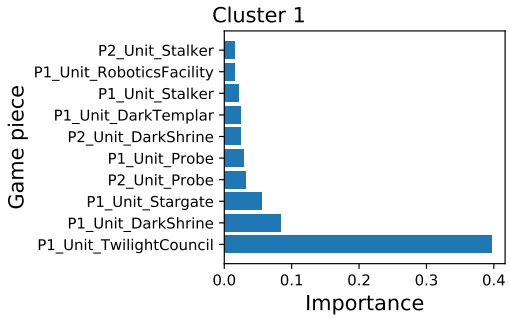


Figure 4.9: Plot of most important game pieces in cluster 1 at $t = 270$ s.

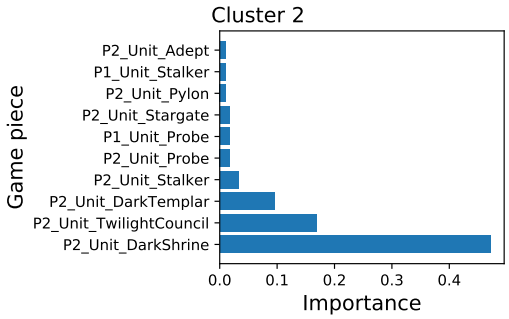


Figure 4.10: Plot of most important game pieces in cluster 2 at $t = 270$ s.

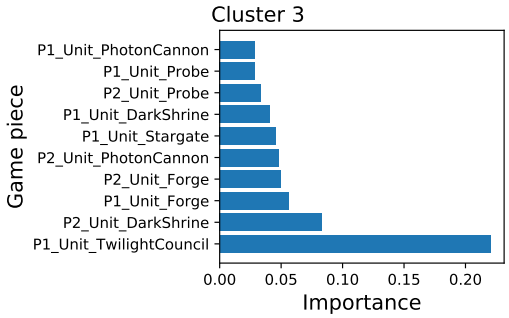


Figure 4.11: Plot of most important game pieces in cluster 3 at $t = 270$ s.

Observe that the importance of Twilight Council controlled by player 1 dominates the game piece importances of both cluster 1 and cluster 3. For cluster 2, the game piece Dark Shrine controlled by player 2 is the dominating game piece, with Twilight Council controlled by player 2 being the second most important game piece. In StarCraft II, the game piece Twilight Council is a prerequisite for Dark Shrine, meaning that whenever Dark Shrine is

an important game piece, so is the Twilight Council indirectly. As such, clusters 1, 2, and 3 in some way all have a substantial dependency on the same game piece: Twilight Council. Note, however, that cluster 3 seems to be differentiated by the fact that the Twilight Council importance is not as large relative to the other importances as it is in clusters 1 and 2. The list of the ten most important game pieces for cluster 3 also contains completely different game pieces in comparison to cluster 1 and 2, which have very similar lists of most important game pieces.

In a similar fashion, importances of game pieces for clusters 4, 5, and 6 are shown in 4.12, 4.13, and 4.14, respectively.

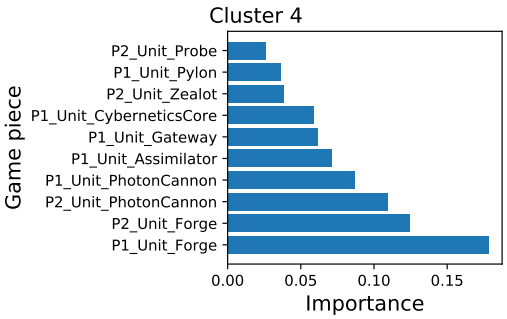


Figure 4.12: Plot of most important game pieces in cluster 4 at $t = 270$ s.

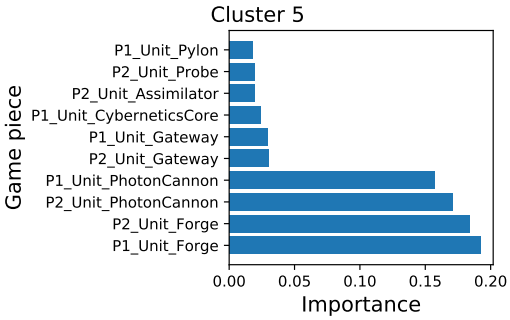


Figure 4.13: Plot of most important game pieces in cluster 5 at $t = 270$ s.

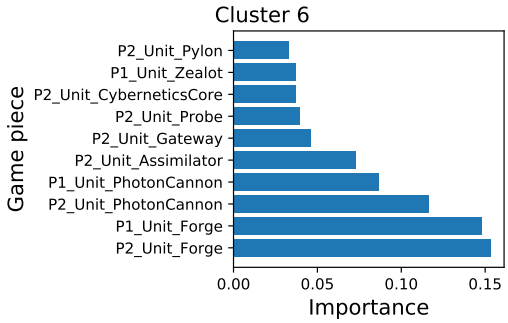


Figure 4.14: Plot of most important game pieces in cluster 6 at $t = 270$ s.

Observe that Forge and Photon Cannon for both players bear the highest importances for clusters 4, 5, and 6. It is also interesting to note that, in comparison to clusters 1, 2, and 3, the importances of the ten most important game pieces for cluster 4 and 6 are more equally distributed. This means that there are no game pieces which dominates clusters 4 and 6 as much as Twilight Council and Dark Shrine dominates clusters 1, 2 and 3. Both players are also represented fairly equally in the most important game pieces Forge and Photon Cannon for clusters 4, 5 and 6, while these are absent in clusters 1 and 2.

To summarize, based on feature importance analysis, clusters 1 and 2 are both primarily determined by combinations of Twilight Council and Dark Shrine, while cluster 4, 5, and 6 are primarily differentiated by combinations of Forge and Photon Cannon. Cluster 3 is

distinguished by Twilight Council, Dark Shrine, Forge, and Photon Cannon, and therefore seems to be a mix between the rest of the clusters. Since this cluster also contains nearly 60% of the total data, it may be interpreted as containing all games which are not primarily distinguished by a combination of Twilight Council and Dark Shrine or Forge and Photon Cannon. Therefore, it is natural to divide the clusters into the following groups: group 1, consisting of clusters 1 and 2 and group 2, consisting of clusters 4, 5, and 6. Cluster 3 is treated as an outlier, since it appears to be defined as the complement to the other clusters.

4.3.2 Analysis of the development of important game pieces within clusters

Figures 4.15, 4.16, 4.17, and 4.18 show the mean line plots of the game pieces determined to be most important for group 1 in the previous section: Twilight Councils controlled by player 1 and 2, and Dark Shrines controlled by player 1 and 2. The mean lines represent the average amount of a game piece found in the games within each cluster over time.

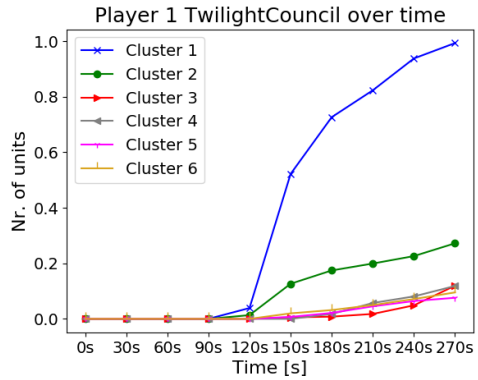


Figure 4.15: Plot of the average number of Twilight Councils controlled by player 1 for each cluster.

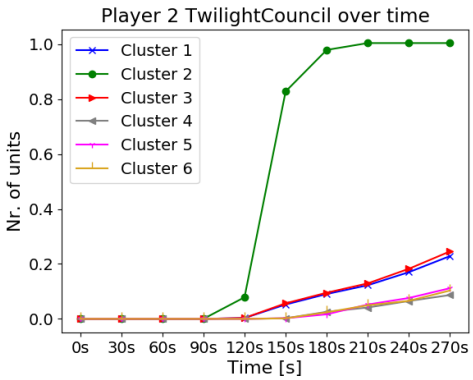


Figure 4.16: Plot of the average number of Twilight Councils controlled by player 2 for each cluster.

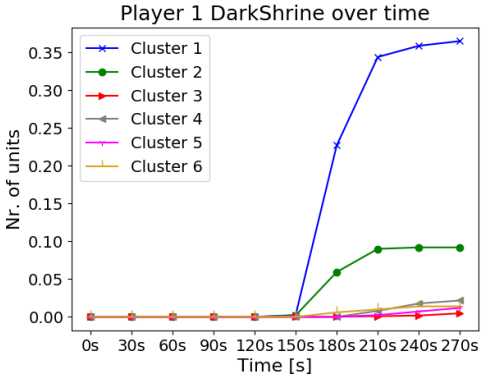


Figure 4.17: Plot of the average number of Dark Shrines controlled by player 1 for each cluster.

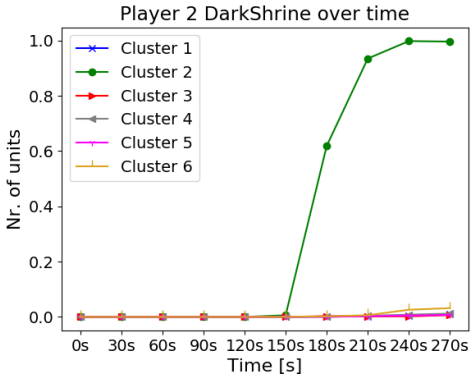


Figure 4.18: Plot of the average number of Dark Shrines controlled by player 2 for each cluster.

Observe that cluster 1 and 2 seem to mirror each other, as the mean plot lines for player 1 and 2 are similar for both the Twilight Council and Dark Shrine game pieces. This indicates that cluster 1 contains games in which player one acquires these game pieces, and cluster 2 contains games in which player two acquires these pieces.

Cluster 3 seems to be differentiated from the others in the mean line plots as well, apart from being similar to cluster 1 in Twilight Council and Dark Shrine for player 2. Also note that cluster 3 is not mirrored by another cluster in the way one would expect following the observations done for clusters 1 and 2.

Overall, one can clearly see that early game acquisitions of Twilight Councils and Dark Shrines seem to differentiate clusters in group 1 from clusters in group 2, especially when viewing the mean line plots for each variable, while cluster 3 seems to follow no distinguishing pattern in these game pieces.

Figure 4.19, 4.20, 4.21, and 4.22 show the mean line plots of the game pieces determined to be most important for group 2 in the previous section: Photon Cannon controlled by player 1 and 2, and Forge controlled by player 1 and 2.

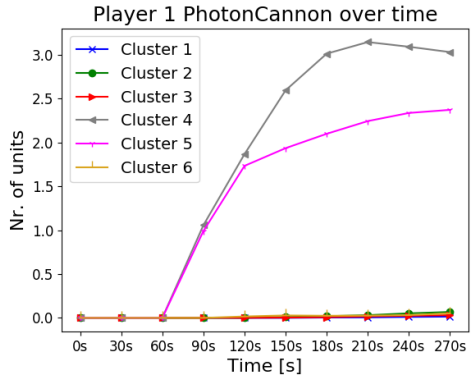


Figure 4.19: Plot of the average number of Photon Cannons controlled by player 1 for each cluster.

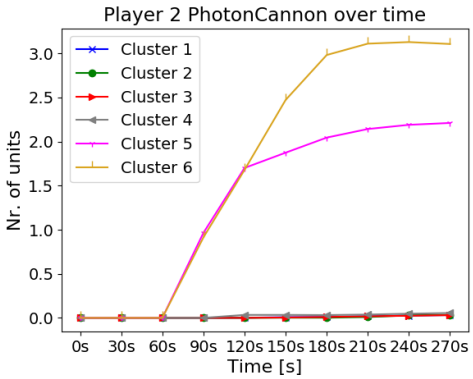


Figure 4.20: Plot of the average number of Photon Cannons controlled by player 2 for each cluster.

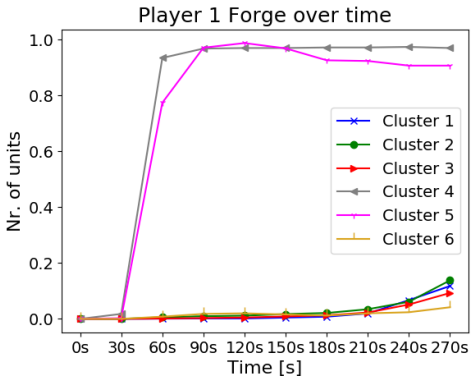


Figure 4.21: Plot of the average number of Forges controlled by player 1 for each cluster.

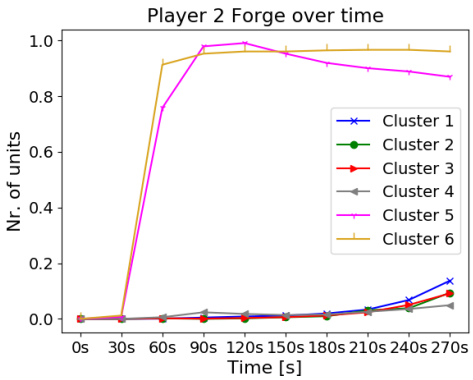


Figure 4.22: Plot of the average number of Forges controlled by player 2 for each cluster.

The results presented in Figures 4.19 - 4.22 indicate that early game acquisition of Photon Cannon and/or Forge differentiate clusters 4, 5, and 6 from the other group of clusters. Once again, cluster 3 seems to follow no distinguishing patterns for these game pieces.

The same figures show that the clusters seem to be mirrored in these game pieces, reflecting the fact that in some games both of the players acquire one of these pieces, and in other games only one of the players do so. Since player number assignment is random in StarCraft II, this produces the mirroring effect. To see this effect, one can observe the mean line plots for Forge. Since the line representing cluster 5 is identical for both players, this cluster consists of games where both players acquire Forge early. Meanwhile, the lines for cluster 4 and 6 trade places between the players, i.e. the mean line for Forge units controlled by player 1 in cluster 4 is similar to the mean line for Forge units controlled by player 2 in cluster 6. This indicates that cluster 4 and 6 represent early game acquisition of Forge by player 1 and player 2, respectively. An identical analysis can be made for Photon Cannon.

The results of summing the feature importances of all features for each time point are presented in Figure 4.23. As time progresses, the two groups of clusters show clear differences in terms of how important the numbers of different game pieces are at certain points in time.

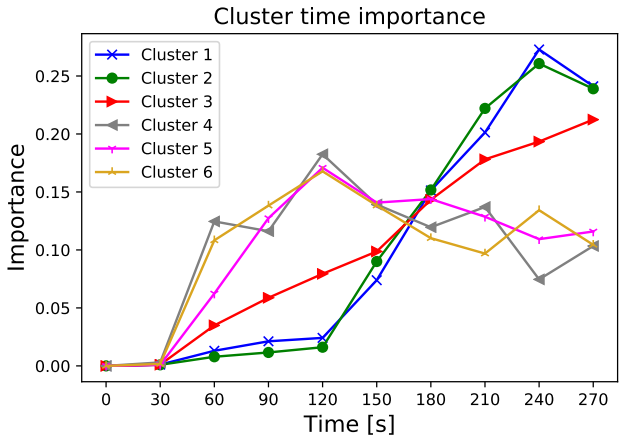


Figure 4.23: Time importance for all six clusters.
In this graph, the importances of the data sampled at time t is visualized for all clusters. Note that clusters 1, 2 and 4 follow a similar pattern. The same goes for clusters 5 and 6, while 3 behaves in a different manner.

It is clear that for clusters in group 1 the configuration of game pieces later in the game are more distinguishing. The opposite is true for group 2, where the number of game pieces earlier in the game are deemed more important. This also falls in line with which game pieces are deemed important by the random forest classifiers. The first group of clusters are mainly dependent on game pieces which can only be acquired later on in a StarCraft II game, whereas the game pieces that the second group of clusters depend on are available earlier in the game. It is also noteworthy that cluster 3 behaves somewhat independently here, acting almost as an intermediary between the two groups.

To summarize, the results of the feature importance analysis in terms of player strategies are presented in Table 4.2.

	Player 1	Player 2
Cluster 1	Twilight	NOT Twilight
Cluster 2	-	Twilight
Cluster 3	NOT Twilight/Forge	NOT Twilight/Forge
Cluster 4	Forge	NOT Forge
Cluster 5	Forge	Forge
Cluster 6	NOT Forge	Forge

Table 4.2: Table of player strategies contained in the clusters.
Each strategy is denoted by the game piece which is most significant for that strategy. For example, Forge denotes a strategy which is distinguished by early acquisition of the game pieces Forge and/or Photon Cannon. Twilight is short for Twilight Council and includes Dark Shrine related strategies.

5

Discussion

This chapter discusses the results of this thesis in relation to the problem statements introduced earlier. Recommendation for areas of potential future work are made throughout this section.

5.1 Data analysis and clustering

One basic assumption was made at the beginning of this study: the game state at any point in the game is dependent on the previous development of the game. Therefore, it seems meaningless to find clusters of game states at a single time point, and this prompted a more complex approach to the clustering. The approach which was decided upon, clustering of the time developments of the game states, was deemed to be the logical conclusion following this assumption. However, the approach was implemented rather naively, in the sense that game state time series were simply iteratively appended in vectors for each time point, resulting in the game data at later time points having parameters numbering over 1000.

In addition to this, since the vectors provided to the cluster algorithm were scaled, the clustering remained objective with respect to the different game pieces so that no game piece had disproportional influence on the clustering by the virtue of commonly being controlled by the players in larger numbers. On the other hand, certain game pieces perhaps ought to have greater impact on the clustering by virtue of their gameplay role. Therefore, manually selecting weights of the game pieces before clustering is a question for future work.

The particular clustering method, agglomerative hierarchical clustering, was chosen rather arbitrarily, though it was chosen as a result of extensive experiments and as a result it has certain merits over other methods. Since there exist various kinds of appropriate clustering methods, it is a daunting or even impossible task to identify the optimal method of clustering for this particular problem. Once again, there is therefore potential for future work to improve upon this.

The method of evaluating clusterings could also be studied more carefully. For example, different limits to the minimum acceptable cluster size could be tested, as the 5%-limit may be too restrictive or generous. There might also be other types of indices than Dunn's index more suited to this specific purpose.

To summarize, there are many areas of improvement and potential future work to be done for

this particular method. There may be other methods of analysis on multivariate time series, with other considerations taken to scaling and processing of the data, which may render even more well separated clusters and therefore better results. In the end, however, the chosen method proved to be successful in the sense that it managed to distinguish differences in the game state time series which are interpretable in terms of how some game pieces develop over time in the games.

5.1.1 Feature importance analysis

The results of the cluster analysis together with the random forest feature importance analysis show that the method presented in this thesis is able to distinguish meaningful game state time series in StarCraft II. The method generated clusters containing several qualitative differences in terms of the strategies of both players. Note that the summation method used to produce game piece importance from random forest feature importances is rather naive. However, when viewing the mean line plots for the game pieces deemed important, there are obvious differences between the clusters, which somewhat confirms the summation method as having some meaning to it.

The greater overarching problem with the approach of this study, also noted by Hayes and Beling [17], it is impossible to precisely interpret the clusters without missing potential subtlety. The clusters seem to be composed of a few easily interpreted strategical differences, but there could be certain features only important in certain complex combinations.

5.1.2 Choice of features

While the intention of the clustering process was to find clusters based on as much information from the game as possible, thereby letting the algorithm decide what is important to the greatest possible extent, some limitations were made on which features were included. Perhaps most notably, the positions of the game pieces were left out of the clustering due to difficulty in how to represent it as values in a vector. Since the amount of game pieces controlled by a player varies throughout the game, assigning a feature for each position would lead to inconsistently sized game state vectors. Including the positions in the clustering process in some way, however, could lead to more realistic clusters since the location of a player's army or buildings might imply differences in strategy or playing style.

5.1.3 Early and late time importances

As shown in Figure 4.23, the two groups of clusters diverge in terms of the time importance of game pieces. It can be interpreted that membership to either of these two groups is mostly dependent of the overall strategical priorities players have within the game in question. In some games, players will prioritize early combat oriented game pieces, for example acquiring the military building Photon Cannons. In other games, however, both players will be content with little early aggression in order to instead utilize their resources within the own base to be able to afford more advanced pieces later on. In our clusters, these more advanced pieces

are the Twilight Council and Dark Shrine which were deemed most important in the other group.

5.1.4 Player mirroring

During data parsing and game state time series clustering, the game pieces controlled by each player were separated into two different columns, Player 1 and Player 2, and therefore represented as two distinct components in the vectors. In StarCraft II, there is no difference between either player as the designated starting location is decided randomly. Additionally, whether a player becomes Player 1 or Player 2 is also decided randomly. The probability of a certain opening move or game state time series existing in either of the two players data sets should therefore be equal.

As such, if the replay data is of sufficient size, the data points should be mirrored within the game state time series space. This means that a clustering method designed to meet the goals of this study should show no bias toward either of the players. However, the clustering did not show mirroring in all of the studied game state parameters. The reasons for this are unknown and could simply be weakness of the method or perhaps be due to a more complex mirroring which is not easily comprehended by a human. In the latter case, one could argue that the method is a powerful method of distinguishing such complex differences which would be impossible to do manually. Once again, there is potential for future work in this area.

5.1.5 Potential subcluster structures

In the clustering chosen for further analysis, performed at 270 seconds, cluster 3 contained roughly 60% of the games that lasted until that point, while the rest of the clusters barely exceeded the 5%-limit. Seeing as the smaller clusters proved to represent fairly distinct playing styles, it is probable that dissecting cluster 3 would result in identifying further styles of play within StarCraft II. Upon closer inspection of cluster 3 in Figure 4.3, it is evident that dividing the cluster into 5 clusters would yield new clusters that are quite distinct, even in relation to the other clusters. This could be achieved with a dynamic dendrogram tolerance cutoff, although this would also necessitate a new method of evaluating clusterings. This new approach would be fitting for future work.

5.2 Classification

In this section, the performance of the classifiers are discussed and compared. It is also discussed how the parameters for the classifiers could have been tweaked and optimized in a more detailed manner. In addition to this, it is discussed whether a more simple and elegant solution could have been found for the problem.

5.2.1 Random forest or neural network

The results yielded by the classification procedure show no obvious differences between the neural network and random forest classifiers in terms of classification accuracy. When it comes to the question of which classifier is more suitable for the task of classification, random forest comes out on top mainly for three reasons. Firstly, training the random forest classifier is not as computationally heavy as training the neural network. Secondly, there are not as many parameters to be determined for the model, which makes the process of design easier. Lastly, the random forest model provides clear insight into what features are the best predictors for a given classification problem, which has been of utmost importance for the procedure of determining what features best separated the clusters.

5.2.2 Classification accuracies on vision data

The strong vision contains more precise information on the enemy piece count than the weak vision. Because of the increased precision, the classification accuracy should yield better results when using the strong vision compared to the weak vision. Surprisingly, the resulting accuracies are roughly the same, as can be seen in Table 4.1. A review of the game piece importance may shed some light as to why the results are rather close, see Figures 4.9 - 4.14.

The game pieces of highest importance, except for the Photon Cannon, all have in common that they are types of game pieces that a player only requires one instance of. The speculative reason as to why a more precise unit count does not have larger impact on classification accuracy is that the precise count of any game piece is not of importance, but rather the fact that there exists one instance of that piece is enough. This implies that the actual difference between strong and weak vision, which is the precise counting, does not carry the weight one would assume concerning classification purposes, or at least not with this particular clustering.

Additionally, as the correct vision a human has access to lies within the interval between the two algorithms, see discussion in Section 5.3.2, one can with certainty determine that the accuracy of the classifiers when run on correct human vision data would lie within this tiny interval.

5.2.3 Choice of hyper-parameters

Hyper-parameters greatly impact the performance of classifiers. The initial classification aim was to reach accuracy above chance, but the implementation of the classifiers showed high accuracy on onset. Therefore, tuning of hyper-parameters in order to optimize classifiers to their capacity was given low priority. However, the method included arbitrarily testing a few different configurations for the neural network in order to establish a loose understanding of the effects of the hyper-parameter choices in regard to accuracy.

Across all configurations that were tested, the resulting accuracies were similar enough to be considered equally adequate, and a final configuration was selected based on simplicity. If the

purpose of this thesis was to develop an optimal classifier, however, a slightly higher accuracy might have been possible using a proper optimization algorithm, as opposed to arbitrarily tweaking parameters. Developing an optimal classifier for the problem is a suggestion for future work.

5.2.4 Simpler methods of classification

A shortcoming of random forest and neural network classifiers is that the models potentially are unnecessarily complex. A consequence of this is that it becomes hard to understand how the models work. For example, the closest one can come to understanding the random forest classifier is to extract the model's feature importances, which is not very close to understanding precisely how it works. For this reason, it would have been interesting to attempt to solve the classification problem using simpler linear models, with the goal of training a classifier which makes decisions based on more easily interpreted criteria.

By doing this, the problem and its solution could potentially be better understood. Just training a neural network or random forest to solve the problem may be an overly complicated method to solve a relatively simple problem. It does not maximize ones understanding of the problem, and it has no realistic chance at finding the most simple and elegant solution. There is no guarantee that the simpler models would be able to solve the problem, but it would have been more reasonable to start with this more basic approach and then gradually increase the models' complexities to solve the task. In this thesis, the solution was to start out with potentially overly complex models and thereby disregarding precise understanding of the problem at hand, and how the models work.

Since the data analyzed in this thesis is not proven to be non-linear, it would indeed be appropriate to start out by assuming that it is linear and see what implications this has on classification performance. Only if linear models do not suffice, there would be a good reason to use non-linear models, such as random forests or neural networks. This would be an appropriate domain for future work with problems similar to this one.

5.3 Data parsing

In this section, the choices of parser for the full game states and vision data are motivated. Additionally, the potential source of error in poor replay quality is discussed.

5.3.1 Choice of game state parser

The process of parsing game states of replays was initially performed using the Python library sc2reader [47]. This tool allows for time efficient parsing as it can read game events from replay files without having to simulate the game logic and engine. However, sc2reader has two major limitations. Firstly, these game events are sufficient to generate a game state of pieces each player owns at a given time but inadequate for vision parsing, as no events are

triggered or stored upon enemy game pieces entering a player's line of sight. Secondly, most game events are non-existent in the replays provided by Blizzard, perhaps as a side-effect of anonymization or of file compression. Using sc2reader for parsing would therefore result in the requirement of another sufficiently sized source of replay files. Such a source was failed to be found.

After some trial and error, the Python library sc2reaper [1] became the chosen parsing software. This instrument requires the game engine, using the pysc2 [37] API, to simulate the game from replay files. Simulating the game enables the collection of vision information. It also results in lengthy parsing times, however, as running the game engine results in each replay taking minutes to parse, compared to the milliseconds necessary using sc2reader.

5.3.2 Vision parsing

Being able to accurately represent the vision a player has of their opponent's game pieces was deemed not feasible with the available tools. Therefore, two different vision parsing algorithms are used where one parses strictly less information, and one parses strictly more information, than what a human player has in any given situation.

This first algorithm is weaker than the ground truth human vision as it cannot tell the difference from game pieces of the same type seen at different times. If the system spots one particular game piece at a given time, it cannot tell if that game piece is different to one it has seen before.

In a scenario where the player first spots a certain game piece in one corner of the map, and then soon afterwards spots another game piece of the same type on the other, a human would be able to recognize that these two game pieces are different individuals. As such, a human would be able to realize that the opponent controls at least two of this game piece, whilst the weak vision parsing algorithm would only be able to record having seen one.

Meanwhile, the stronger parsing algorithm calculates a ceiling to what a human would be able to infer of the current game state from the player perspective as it utilizes the identification numbers of game pieces in order to discern individuals. These identification numbers are hidden to players and as such, the stronger algorithm naturally has access to superhuman level of information.

Consider the scenario where a game piece controlled by the opponent moves into the vision of a player, leaves the field of vision, and then finally returns again. A human would not be able to tell with absolute certainty if the game pieces they spotted are the same individual piece or two of the same type. However, the stronger parsing algorithm is capable of distinguishing visually identical game pieces by their identification numbers, and would therefore be able to tell the pieces apart.

By utilizing these two parsing algorithms, and their respective data sets, one can definitively determine that the true partial game state perceived by a human from their vision lies within the intervals of these data sets.

5.3.3 Gameplay quality of replays

This thesis bases its data on the replays provided from Blizzard [36] and the replays used are exclusively of ranked games from players ranking in the top 5% of the player-base. However, even if the results of these games affect the participants' ranking, there is always the risk of improper and uncommitted effort from any of the players. As StarCraft II accounts are free to acquire [48], players may own multiple accounts of which not all of them are used for serious competitive play.

With the extensive size of the data set used in this thesis, one can reasonably assume that it contains a non-zero amount of replays representing improper gameplay. Some replays may contain players simply not trying to win, or playing in some other extraordinary fashion. Such abnormal replays would still pass the criteria applied in the defined filters, described in Section 3.1.1. However, outliers that are significant enough to affect the results of the clustering would have been clearly visible in the visualization of the clusters.

5.4 Ethical considerations

The goal with this thesis is, simply put, to further advance AI. Advancing AI may have very serious implications for societies, and is therefore an important point for ethical discussion. Future AI may be able to perform tasks that used to be performed by humans, for a fraction of the cost. This could lead to the end of humans performing such jobs, potentially leaving a substantial proportion of the world population without employment, resulting in an increase of socioeconomic rifts.

Another serious implication which might not be as probable but could have even more devastating consequences is the risk of creating a singularity. This means that there might be a risk that the AI we create becomes so intelligent that we lose control over it.

Leaving the general risks of contributing to the overall research into advanced AI, the most important ethical consideration with this thesis is the fact that it uses gameplay data from thousands of players. Technically, this could be considered as private information. This issue can be mitigated by the means of data anonymization, but there is always a strong ethical aspect in handling private information.

In addition to this, there is the possibility that the results presented in this thesis are used for cheating and/or other nefarious purposes in the setting of professional StarCraft II E-sport. This may have impact on the livelihoods of professional players, since there is a growing population of people earning a living solely on E-sports. However, this is not very likely to become a real problem, as there already exist strict frameworks and regulations for handling such problems in this context [49]. Additionally, a powerful AI trained in the context of StarCraft II may prove to be an assistance to human players using the AI for practice.

5.5 Possible uses of results

Difference between resulting clusters in terms of units and timings can be viewed as expressions of player strategy. Viewed in this way, clusters can hypothetically be used to analyze popularity and the contents of strategies in the games that generated the data. Such a meta game analysis could potentially be employed by game developers to patch the game such that all units are used or to provide casters with interesting statistics. Future research could yield additionally interesting results.

Another view of the results of this thesis is to further enhance the classification to be able to use it as a means of identifying enemy strategy. A player could use such a system while playing as it guesses the complete game state, including the opponent's game pieces. In such a case, the player could get an advantage by using the deduction powers of the classification system to guess currently hidden game elements only known by the opponent. The actual usefulness of this deduction might be limited, however, since the classifiers potentially use the same information to guess the opponent's strategy as a human would.

5.6 Future work

As a suggestion for continued work on this topic, further investigations could be conducted as to whether the clustering of training data leads to clusters with a significantly lower variance than the entire set, proportionally to the reduction in variance that should arise directly from splitting the data set. This reduction in variance is the overall motivation for the clustering performed in this project, but as was stated in Section 1.3, validating this effect is not a part of the thesis.

Additionally, further investigating other methods for both clustering and classification could lead to improved results. Properly optimizing the hyper-parameters of the neural network or testing simpler classification methods could lead to better accuracy and more robust models. Attempting to split the existing clusters into smaller clusters using more advanced clustering algorithms could increase both the number and quality of the clusters.

6

Conclusions

During the conception of this thesis, the following questions were specified as a means of achieving the overall goals of the study:

- Is it possible to find meaningful clusters of game state time series using replay data from StarCraft II?
  - If so, what is an appropriate clustering method?
- Which game pieces are relevant for effectively distinguishing between these clusters?
- Are neural networks and/or random forests feasible methods of classifying clusters of game state time series using incomplete information?
  - If so, how accurate are the methods?

The study concludes that it is possible to find clusters within replay data representing game state time series from StarCraft II, using agglomerative hierarchical cluster analysis with ward linkage and the cosine distance metric. Using feature importance analysis via random forest, the clusters were found to represent meaningful differences in how the two players prioritize construction of certain game pieces.

While the acquisition timings of certain game pieces seem to have greater importance as to which cluster a specific game belongs to than other pieces, they are not the sole contributors. There seems to be a more complex interplay between multiple game piece acquisition timings that contribute to this determination.

Investigations into the problem of classifying which cluster a specific game corresponds to show that artificial neural networks and random forests both produce similar results with high accuracy. The specific layouts seem to be of less importance beyond a certain threshold.

Bibliography

[1] M. G. Duque, "sc2reaper," 2019. [Online]. Available: https://github.com/miguelgondu/sc2reaper Accessed on 11 March 2019.

[2] IBM, "Icons of Progress - Deep Blue," 2011. [Online]. Available: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/ Accessed on 4 February 2019.

[3] D. Hassabis and D. Silver, "Alphago zero: Learning from scratch," 2017. [Online]. Available: https://deepmind.com/blog/alphago-zero-learning-scratch/ Accessed on 4 February 2019.

[4] S. Gadam, "Artificial intelligence and autonomous vehicles," 2018. [Online]. Available: https://medium.com/datadriveninvestor/artificial-intelligence-and-autonomous-vehicles-ae877feb6cd2 Accessed on 4 February 2019.

[5] B. Marr, "Machine Learning In Practice: How Does Amazon's Alexa Really Work?" 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/10/05/how-does-amazons-alexa-really-work/#130829521937 Accessed on 4 February 2019.

[6] G. Seif, "Deep Learning for Image Recognition: why it's challenging, where we've been, and what's next," 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef Accessed on 12 February 2019.

[7] OpenAI, "OpenAI Five," 2018. [Online]. Available: https://blog.openai.com/openai-five/ Accessed on 4 February 2019.

[8] Blizzard Entertainment, "Game overview," 2019. [Online]. Available: https://starcraft2.com/en-us/game Accessed on 15 May 2019.

[9] Ranked For Teh Win, "1v1 Population," 2019. [Online]. Available: https://www.rankedftw.com/stats/population/1v1/#v=2&r=-2&sy=g&sx=sl Accessed on 29 March 2019.

[10] Esport Earnings, "Events for StarCraft II," 2019. [Online]. Available: https://www.esportsearnings.com/games/151-starcraft-ii/events Accessed on 29 March 2019.

[11] O. Vinyals, S. Gaffney, and T. Ewalds, "DeepMind and Blizzard open StarCraft II as an AI research environment," 2017. [Online]. Available: https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/ Accessed on 4 February 2019.

[12] The AlphaStar Team, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," 2019. [Online]. Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/ Accessed on 4 February 2019.

[13] DeepMind, "DeepMind StarCraft II Demonstration," YouTube, 2019. [Online]. Available: https://youtu.be/cUTMhmVh1qs?t=6118 Accessed on 9 February 2019.

[14] A. Pietikäinen, "An Analysis On How Deepmind's Starcraft 2 AI's Superhuman Speed is Probably a Band-Aid Fix For The Limitations of Imitation Learning," 2019. [Online]. Available: https://blog.usejournal.com/an-analysis-on-how-deepminds-starcraft-2-ai-s-superhuman-speed-could-be-a-band-aid-fix-for-the-1702fb8344d6 Accessed on 9 February 2019.

[15] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. P. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A New Challenge for Reinforcement Learning," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[16] Y. Li, Y. Li, J. Zhai, and S. Shiu, "Rts game strategy evaluation using extreme learning machine," *Soft Computing*, vol. 16, no. 9, pp. 1627–1637, Sep 2012. [Online]. Available: https://doi.org/10.1007/s00500-012-0831-7

[17] R. Hayes and P. Beling, "Unsupervised hierarchical clustering of build orders in a real-time strategy game," *The Journal of Computer Games*, vol. 7, no. 1, p. 5–26, 2018. [Online]. Available: https://link.springer.com/article/10.1007/s40869-018-0051-1

[18] T. Korenius, J. Laurikkala, and M. Juhola, "On principal component analysis, cosine and Euclidean measures in information retrieval," *Information Sciences*, vol. 177, no. 22, pp. 4893 – 4905, Nov 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025507002630

[19] F. Murtagh, "A Survey of Recent Advances in Hierarchical Clustering Algorithms," *The Computer Journal*, vol. 26, no. 4, pp. 354–359, Nov 1983. [Online]. Available: https://doi.org/10.1093/comjnl/26.4.354

[20] A. Bouguettaya, Q. Yu, X. Liu, X. Zhou, and A. Song, "Efficient agglomerative hierarchical clustering," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2785 – 2797, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414006150

[21] F. Murtagh and P. Legendre, "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" *Journal of Classification*, vol. 31, no. 3, pp. 274–295, Oct 2014. [Online]. Available: https://doi.org/10.1007/s00357-014-9161-z

[22] J. C. Dunn, "Well-Separated Clusters and Optimal Fuzzy Partitions," *Journal of Cybernetics*, vol. 4, no. 1, pp. 95–104, 1974. [Online]. Available: https://doi.org/10.1080/01969727408546059

[23] J. C. Bezdek and N. R. Pal, "Cluster validation with generalized Dunn's indices," in *Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, Nov 1995, pp. 190–193. [Online]. Available: https://doi.org/10.1109/ANNES.1995.499469

[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York: Springer, 2017.

[25] G. Louppe, "Understanding Random Forests: From Theory to Practice," *arXiv e-prints*, p. arXiv:1407.7502, Jul 2014. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2014arXiv1407.7502L

[26] scikit, "Ensemble methods," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html#id5 Accessed on 27 April 2019.

[27] F. Pedregosa *et al*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct 2011. [Online]. Available: http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[28] H. Lan, "The Softmax Function, Neural Net Outputs as Probabilities, and Ensemble Classifiers," 2017. [Online]. Available: https://towardsdatascience.com/the-softmax-function-neural-net-outputs-as-probabilities-and-ensemble-classifiers-9bd94d75932 Accessed on 24 April 2019.

[29] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Oct 2014. [Online]. Available: https://arxiv.org/pdf/1404.7828.pdf

[30] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent," in *Proceedings of COMPSTAT'2010*, 2010, pp. 177–186. [Online]. Available: https://doi.org/10.1007/978-3-7908-2604-3_16

[31] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, 2012. [Online]. Available: http://arxiv.org/abs/1206.5533

[32] L. Prechelt, "Early stopping - but when?" in *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 55–69. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_3

[33] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb 2012. [Online]. Available: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[34] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics Surveys*, vol. 4, pp. 40–79, Mar 2010. [Online]. Available: https://projecteuclid.org/euclid.ssu/1268143839

[35] C. Elkan, "Evaluating Classifiers," 2012. [Online]. Available: http://cseweb.ucsd.edu/~elkan/250Bwinter2012/classifiereval.pdf Accessed on 15 May 2019.

[36] Blizzard, "sc2client-proto," 2019. [Online]. Available: https://github.com/Blizzard/s2client-proto/tree/master/samples/replay-api Accessed on 3 April 2019.

[37] DeepMind, "pysc2," 2019. [Online]. Available: https://github.com/deepmind/pysc2 Accessed on 11 March 2019.

[38] Ranked For Teh Win, "1v1 League Distribution," 2019. [Online]. Available: https://www.rankedftw.com/stats/leagues/1v1/#v=2&r=0&sx=a Accessed on 11 March 2019.

[39] Liquipedia, "Patches," 2019. [Online]. Available: https://liquipedia.net/starcraft2/Patches Accessed on 3 April 2019.

[40] SciPy, "Scipy," 2019. [Online]. Available: https://www.scipy.org/ Accessed on 22 March 2019.

[41] M. Verleysen and D. François, "The Curse of Dimensionality in Data Mining and Time Series Prediction," in *Computational Intelligence and Bioinspired Systems*. Berlin, Heidelberg: Springer, 2005, pp. 758–770. [Online]. Available: https://doi.org/10.1007/b136983

[42] S. Wang, C. Aggarwal, and H. Liu, "Random-Forest-Inspired Neural Networks," *ACM Transactions on Intelligent Systems and Technology*, vol. 9, no. 6, pp. 69:1–69:25, Oct 2018. [Online]. Available: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/3232230

[43] Keras, "Keras," 2019. [Online]. Available: https://keras.io/ Accessed on 24 April 2019.

[44] TensorFlow, "Tensorflow," 2019. [Online]. Available: https://www.tensorflow.org Accessed on 24 April 2019.

[45] I. Borg and P. J. F. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[46] Scikit learn, "Multi-dimensional scaling," 2019. [Online]. Available: https://scikit-learn.org/stable/modules/manifold.html#multidimensional-scaling Accessed on 15 May 2019.

[47] K. Graylin, "sc2reader," 2015. [Online]. Available: https://github.com/GraylinKim/sc2reader Accessed on 11 March 2019.

[48] Blizzard Entertainment, "StarCraft II Official Game Site." [Online]. Available: https://starcraft2.com/en-us/ Accessed on 29 April 2019.

[49] ——, "2018 StarCraft II World Championship Series Official Competition Rules, Terms and Conditions," 2018. [Online]. Available: https://bnetcmsus-a.akamaihd.net/cms/page_media/m5/M538872VEAGF1516242955164.pdf Accessed on 10 April 2019.

[50] ——, "New Ladder Maps for Season 2, 2018," 2018. [Online]. Available: https://news.blizzard.com/en-us/starcraft2/21776216/new-ladder-maps-for-season-2-2018 Accessed on 15 May 2019.

List of Figures

List of Tables

A

Appendix

A.1 Additional information on StarCraft II

A player begins a game of StarCraft II with worker units and a main base structure. These workers gather resources and can construct buildings. In turn, buildings can have many different uses. Among other uses, buildings can produce units, research upgrades, or provide defensive capabilities.

Games are played on a official set of maps from a rotating map pool [50], which all follow the same underlying structure. The two player halves of the map are mirrored in order to not give one of the two participating players any geographical advantage.

Figure A.1 shows how the entire game UI looks for a player, and the minimap UI element is detailed in Figure A.2. Later, vision is further explained in Figure A.3 and Figure A.4 shows a base as it is being attacked during the later stages of a game.

Figure A.1: An in-game first-person view of a game of StarCraft II.
This figure shows the perspective of the blue player which currently has its Probe *unit selected, shown through the green circle around it and the information shown in the bottom part of the view. In the bottom left is the* minimap*, showing a quick overhead view of the entire game board, and in the top right is the player's resources. The blue player has just spotted a red unit and building.*

Figure A.2: A close-up of the minimap during a game of StarCraft II.
The minimap is seen from the green player's perspective, resulting in the visibility of every green unit and structure on the map. Units and buildings of the opponent's are only visible in proximity to the green player's own, as seen in the bottom left corner of the map. However, as the green player has not scouted the bottom right, the red player's base that lies there is completely shrouded by the dark fog of war. The white trapezoid represents what is currently shown in the view of the main game camera. Note the symmetry of the map.

Figure A.3: Vision levels and restriction in StarCraft II.
These three images depict a scouting blue worker as it spots an enemy worker and an enemy building. The vision of the blue worker is emitted in a circle, but cannot reach over higher elevation. Only when the red worker starts to move up the ramp does it see up onto the high ground. Other vision occluding elements also exist within the game.
Additionally, the high ground in the bottom right of each image showcases the three levels of vision. On the left, the blue player have never had visibility of the high ground, resulting in the area on the other side of the ramp being darkly dimmed. In the middle image, the blue player has active vision of the high ground, resulting in it being lit. Finally, on the right, the high ground is now only slightly dimmed as the blue player has had vision of this area before. Enemy units are only visible when in the highest level of visibility, but buildings remain in their last seen state even when vision of them no longer is granted.

Figure A.4: A green player attacking a red player's base.
The red player's base is placed nearby the blue resource patches, seen on the left, of which there exists often upwards of a dozen on each map. The smaller red units are the worker units which gather resources by travelling between the resource patches and the main structure. Note that the image is taken from the green player's perspective, resulting in the circular field of vision emitting from their units. Both players are playing the Protoss race.

A.2 Proof of equivalence between Euclidean distance and Cosine dissimilarity for normalized vectors

Consider normalized vectors. A vector A is *normalized* if it has unit length, i.e. the square root of the summed squares of its components is equal to one:

$$\sqrt{\sum_i A_i^2} = 1.$$

Normalization of a vector can be achieved by dividing each component of the vector by the norm of the vector. For two normalized vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we have that

$$\begin{aligned}
(d_{\text{euclidean}}(\mathbf{x}, \mathbf{y}))^2 &= \sum_{i=1}^{n} (x_i - y_i)^2 \\
&= \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2 - 2\sum_{i=1}^{n} x_i y_i \\
&= 2(1 - \sum_{i=1}^{n} x_i y_i) \\
&= 2\left(1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}\right) \\
&= 2d_{\cos}(\mathbf{x}, \mathbf{y}),
\end{aligned}$$

where in the third and fourth equality the fact that $\sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\sum_{i=1}^{n} y_i^2} = 1$ is utilized.

A.3 Plot of game length

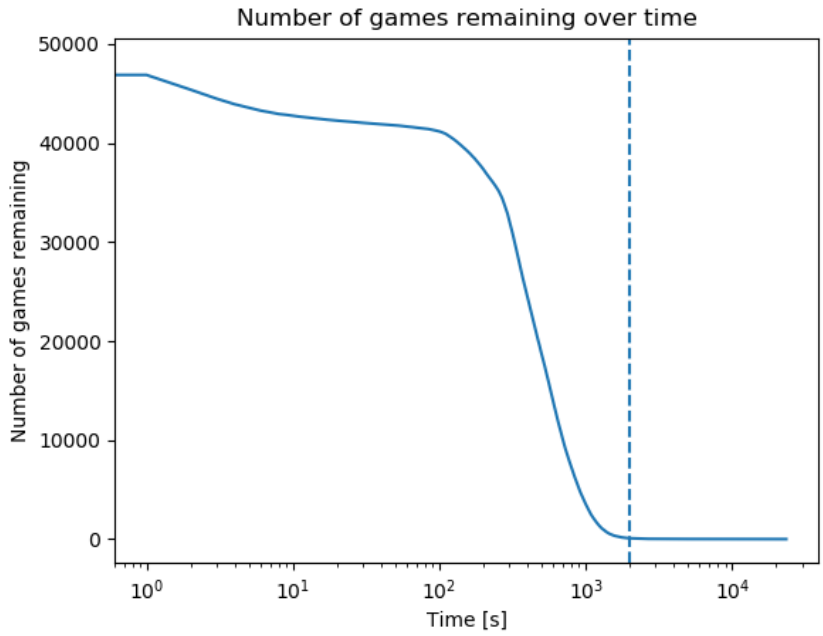


Figure A.5: Plot of number of games remaining over time.
Plot of the number of games remaining over time. Note that the x-axis uses a logarithmic scale. The dashed line indicates the cut-off of 2000 seconds. Games ending after the cut-off were discarded.

A.4 Example of preprocessing

The following is an illustrative example of the preprocessing of the data. Consider a simplified version of StarCraft II in which there are only two game pieces available for play, Probe and Nexus. As such, the game state vectors consist of the number of Probes for each player, x_1 and x_3, and each Nexi for each player, x_2 and x_4. Let's say a game G_i of this simplified version ends at $T_i = 30$ s, and that P_1 command 0 Probes and 2 Nexi at time $t = 0$ s and 1 Probe and 3 Nexi at time $t = 30$ s. Let P_2 command 5 Probes and 6 Nexi at time $t = 0$ s and 10 Probes and 12 Nexi at time $t = 30$ s. As parsed originally, the game is therefore represented as the set of vectors

$$G_i = \{O_i^0, O_i^{30}\},$$

where

$$\begin{aligned} O_i^0 &= (x_1^0, x_2^0, x_3^0, x_4^0) \\ &= (0, 2, 5, 6), \end{aligned}$$

and

$$\begin{aligned} O_i^{30} &= (x_1^{30}, x_2^{30}, x_3^{30}, x_4^{30}) \\ &= (1, 3, 10, 12). \end{aligned}$$

The game is therefore represented as the set of time series up to $t = 30$ s

$$\begin{aligned} \{g_i^t\}_{t=0,30} &= \{O_i^0, (O_i^0, O_i^{30})\} \\ &= \{(0, 2, 5, 6), (0, 2, 5, 6, 1, 3, 10, 12)\}, \end{aligned}$$

and as such, the development over time up to $t = 30$ s of all game pieces commanded by each player is represented at each 30 second time interval. Note that each time series contains all the previous time series vectors, since they are iteratively concatenated.