Use of Machine Learning Techniques in Real-time Strategy Games

Martin Čertický
Technical University
in Košice
Email: martin.certicky@tuke.sk

Martin Sarnovský
Technical University
in Košice
Email: martin.sarnovsky@tuke.sk

Tomáš Varga
Technical University
in Košice
Email: tomas.varga10@student.tuke.sk

***Abstract*—Despite excessive amount of research done in the field of automated RTS gameplay, subtle changes in strategies are often ignored leading to non-optimal results. Researchers often consider obvious strategical decisions in order to cover as many gameplay scenarios as possible. In this paper, we focus on creating a dataset of atypical strategies which are often overlooked and using machine learning methods for detection of these strategies during the gameplay. Such information could be used to predict the strategies before they occur, or to correspond which adaptive behavior is able to answer them. In work presented in this paper, we approached the strategy recognition in StarCraft game using a set of classifiers trained on data obtained from the various replays. As there was not a proper dataset available to solve such task during our work, the dataset was created and annotated to cover four selected strategies for our experiments. Binary classification models were trained to detect each particular strategy and evaluated in a set of replay data using cross-validation technique. Then the overall platform architecture to train the models, export them and use in run-time during the gameplay was designed. Best performed models were then applied to real games to detect the covered strategies in replays or matches by bots or human players.**

***Index Terms*—Data analysis, Machine learning, Classification, StarCraft**

I. Introduction

The field of video-games is ever expanding with many sub-genres available. Although there are a wide variety of existing types, a lot of them share the same basic principles. With the evolution of technology the games methodology have to evolve as well. This is not different with RTS (real-time strategy) games which first came in the early eighties, and are still managing to keep their popularity, sales and player base. RTS games as a genre of video games in which players manage economic and strategic tasks by gathering resources and building bases, increase their military power by researching new technologies and training units, and lead them into battle against their opponent(s), serve as an interesting domain for Artificial Intelligence (AI) research. Because of the nature of genre, the games are dynamic, fast, ever-changing and are one of the most played games throughout the history of video-games.

Research in the area of RTS game AI is usually classified according three levels of abstraction: high-level strategy, middle-level tactics and low-level reactive unit control (referred to as "micromanagement" by RTS players). Reactive control aims at maximizing the effectiveness of individual units of different types in combat by moving them on the battlefield and selecting the attack targets in response to terrain and the activity of opponent's units. Decision-making for reactive unit control can in general take centralized or decentralized (distributed) approach. Centralized approaches try to control the whole group of units at once by searching a game tree. For example, Churchill et al. [1] presented a variant of alpha-beta search and Wang et al. [2] employed a Monte-Carlo planning approach to the problem of micromanagement. Unfortunately, centralized solutions are usually too slow due to large search space and can only be applied to situations with low unit counts [3].

A. Motivation

StarCraft[1] has more than 180+ variations of "typical" strategies, that are hard to identify prior to countering it during gameplay. Professional players spend years practicing their skills and studying many strategies while the casual players rarely choose optimal answer during real gameplay. Despite excessive amount of research done in the field of automated RTS gameplay, subtle changes in strategies are often ignored leading to non-optimal results. Researchers often consider obvious strategical decisions in order to cover as many gameplay scenarios as possible. In this paper, we focus on creating a dataset of atypical strategies which are often overlooked and using machine learning methods for creating adaptive behavior able to answer them.

II. Problem Domain

If the RTS games, especially StarCraft have that many strategies, is it possible to classify or predict them based on real-time data, with simple system to add another strategy later on?

A. Environment Definition

For purposes of this research we have defined StarCraft as a machine learning environment, according to work of Russel and Norvig [4]. Using the the environment part of their P.E.A.S. analysis we were able to define the game accordingly:

[1]https://starcraft.com/

1) Fully Observable vs. Partially Observable Environment: We define StarCraft as an **partially-observable environment**. One of the reasons is built-in "fog of war" mechanic. The fog obscures the vision on map areas for players which do not have any units nor buildings present at the time, even if the areas were previously discovered . Another reason is that one player does not have information about the other without scouting his actions with self-owned units or buildings.

2) Deterministic vs. Stochastic Environment: Due to several random events occurring throughout the games StarCraft is a **stochastic environment**. For example, if an unit attacks another unit placed on higher ground level (since the game environment is not fully 2-dimensional) or if the attack projectile's path is obstructed by environment, the projectile has 52.125% chance to succeed. In case both units have clear vision of each other, the ranged projectiles have 99.609375% chance of success. Cooldown as a time between two attacks in StarCraft is measured in frames per second, which creates another randomness factor in the environment, meaning that 50 frames cooldown actually means a range between 49 to 51 frames.

3) Episodic vs. Sequential Environment: Episodic environment expects agent's actions to be executed in periodical sequences. Another condition is that previous sequences should not affect future ones. None of these conditions are met in StarCraft, therefore we treat the game as a **sequential environment**.

4) Static vs. Dynamic Environment: In StarCraft, more than one player interfere with an environment. Additionally, there are neutral entities (units) present in the game, procuring the game as a **dynamic environment**.

5) Discrete vs. Continuous Environment: Since StarCraft uses real numbers instead of natural numbers for numerous calculations (e.g. health points), we define the environment as **continuous**.

6) Single-agent vs. Multi-agent Environment: As stated above, typically more than one players interfere with (play) the game, therefore we consider StarCraft as a **multi-agent environment**.

B. Challenges

We identified several challenges that needed to be dealt with throughout our research:

- *Expert knowledge of the game:* External consultant was asked to help define future modeled strategies.
- *Missing dataset:* We used replay data from BWAPI bot vs. bot, primarily focusing on one bot with same strategy against the others. How the dataset was constructed is described in chapter V.
- *Feature selection:* The features were chosen using combination of replay observing and consulting with an expert.
- *Proof of concept strategies:* We focused on four selected strategies which served as a test cases for our modeling process.
- *Real-time aspect:* Due to the real-time nature of the game a time interval had to be chosen in order to group the actions.

III. State of the Art

In earlier work of Hsieh and Sun in 2008 they published one of first articles that focus on building AI for StarCraft. They used case-based reasoning approach to construct system for learning and predicting individual player strategies by mining series of actions from replays. In the conclusion of their work, they mentioned how their research was limited due to closed game environments [5]. Later in 2009 the first version of Brood War API (BWAPI)[2] was released, using third-party DLL tool to read and inject the code to memory. Using this API, it was possible to call in-game functions. From that point research on AI with StarCraft as a test bed skyrocket. First competition was held at AIIDE 2010 and then since 2011 the group later continued to automatize much of the process in order to streamline it. After the two years AIIDE started to host annual student competition that run throughout the year with custom made bots in the bot vs. bot matches After the release of BWAPI new research was added each year, and now almost each major machine learning method has been applied in StarCraft bot competitions. The StarCraft has an edge against other RTS games in area of research. This is due to its popularity and amount of replays and datasets created throughout the years. Some of the most researched topics with StarCraft are:

- Planning: Most of the research in the early days of BWAPI focused highly on case-based approach. For example using it as a reasoning technique for selecting build orders [6],[7] or case-based planner using annotated replays while turning them into cases. Later on the research shifted to more abstract tasks such as planning the build order in real-time [8],[9], or even to military approach using chain of command by implementing hierarchical adversarial search framework.
- Constrain Satisfaction and Optimization: Several independent researches were done on walling, in which player place a building to shelter himself against other players. This strategy is specific type of spatial reasoning, used many times by experienced players against rush strategies. First paper published 2013 by M. Čertický used answer set programming (ASP) to solve this problem [10], while second one published in 2014 used adaptive search technique [11]. Other researchers in this area focus on building a framework called GHOST, that could be also used for other games and even in the game engines [12]. GHOST investigates solving any type of problems encoded by a constrain satisfaction/optimization problem.
- Control: As one of the crucial aspect of any game, research done in this area are among the most explored and focused on unit micro management during combat scenarios [1],[3]. As the games with scripted bots can

[2]http://www.starcraftai.com/wiki/

often seem uninteresting for casual players, the focus on using reinforcement learning as an alternative technique has been used for the bots to be more unpredictable and even make their playstyle more human-like [13],[14]. There was also research done on using potential flows for scout units as better scouting has proven to improve player's chances to win [15].

- Datasets: First official BWAPI dataset was released in 2012 [16]. This dataset included such detailed information as where and how to attack, and analysis of units' movements. Later work on datasets also included a extraction tool that we used in our research [17]. The tool can extract every state in the game, basically describing the whole game in one SQL entry. Later during our work, this tool was used to collect information from professional replays (described in chapter V). Another used dataset is named STARDATA [18]. It was released in 2017 together with new extraction tools using Torchcraft[3]. It consists of 65646 StarCraft replays that contains 1535 million frames and 496 million player actions.

In 2009, Weber et al. published on of first researches using machine learning approach in predicting the opponent's strategy in RTS games. It used thousands of professional replays to acquire domain knowledge and perform opponent modeling. Machine learning was used to recognize the units opponent is producing before being fully scouted along with his strategy. Strategic decisions made in-game were encoded into a feature vector of build orders from the game, which was also labeled with specific strategy based on the previous professional gameplay. The evaluation process used several different algorithms such as C4.5, state lattice, NNge, Boosting. With the results machine learning algorithms outperformed the state lattice on every experiments. The algorithms successfully predicted strategy with high confidence [19]. In 2012, Park et al. focused on predicting early stages of opponent's strategy using scouting units with their own navigation strategy. Their training data consisted of collection of matches between their bot (Xel'naga) and various human players. After collecting the replays, they applied feature extraction methods and thirteen machine learning algorithms to pre-processed data. They pointed out how the expert knowledge poses essential role in helping the machine learning methods to successfully train their models [20]. A year later, same team was able to propose a framework that used replays for creating adaptive bot which took advantage of predicting the strategies and build orders of its' opponents. Their experiments showed promising results in a game with full information (e.g. without fog of war) while using decision tree learning. However, it did not succeed with same accuracy in the early stages of the game. After introduction of hybrid technique, when instead of decision tree rotation forest was used the prediction became successful. The experimentation with build orders demonstrated outcome with results of 6-7 minutes, as the best game time to change the build order. At the end the replays with the human players had better results thanks to more precise scouting, which is the biggest weakness of many bots [21].

[3]https://github.com/TorchCraft

IV. Proposed Architecture

We designed a client-server architecture using, as depicted in Fig.1. BWBigBrother is a python server where the training of our machine learning models was done; data processing, prediction, training. This approach was chosen due to different programming languages used in training and processing our models (Python) and BWAPI (Java). Initially, we have experimented with porting our work in PMML[4] and PFA[5] but the results were unsatisfactory. Using client-server approach, we were able to update models in any moment without needing to interfere with player's software while allowing the community to contribute safely with their own work.

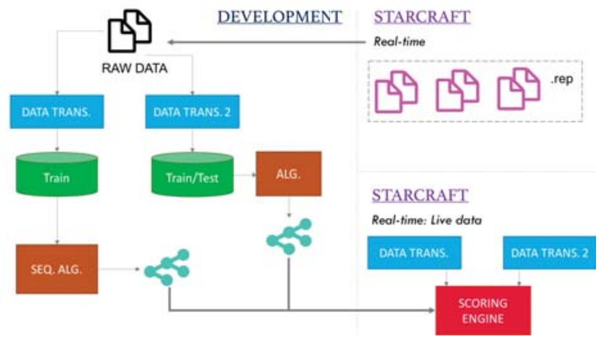

Fig. 1: Description of proposed architecture. Datasets used are described in section V.

We used a web framework Flask in combination with open-source server example SK Learn Flask[6]. This server uses *Pickle* mechanism from Sci-kit Learn library, capable of automated building, training and testing many machine learning models. Additionally, pickle allows users to save and load trained models back to the memory at any time while outputting models in simple *.pkl* file. We chose latter versions of BWAPI to be able to take advantage of community support while allowing new community work to be done with our research.

In the following chapters we address main building blocks along with bottlenecks that arose in the process of development of the system. The BWAPI is used to read data from the memory and send it to our program to onStart() function. In this function, system loads player's information from the settings file and BWTA[7] module for analyzing terrain and map is also launched.

The BWAPI has function *onFrame()* where it receives every update about every unit. The system collects the data only from a player specified in the settings file. Every relevant action by each unit is recorded frame by frame. Only relevant actions are identified to be recorded for modeling purposes. This allows

[4]https://en.wikipedia.org/wiki/Predictive_Model_Markup_Language/
[5]https://en.wikipedia.org/wiki/Portable_Format_for_Analytics/
[6]https://github.com/amirziai/sklearnflask
[7]http://liquipedia.net/starcraft/BWTA

us to store persistent data as each action has to have different attributes.

To record any action a new function *addNewAction()* had to be made, where the system performs a dictionary lookup to ensure consistency between two Java BWAPI interfaces; BWMirror () used here, and JNIBWAPI used in ScExtractor and for our models. This dictionary lookup is converting BWMirror id types to old JNIBWAPI ids.
Another condition in the *onFrame()* sends actions to server every 240 frames. Function *predictActions()* is converting actions to JSON and then sends it with HttpPost method to a server, where upon return it tries to determine if one of the predefined strategies is active or not. This function uses our machine learning models to detect the actual strategy played by opponent. Following sections will describe the models in more detail.

V. Data Processing & Modeling

To correctly predict the player's strategy during the Starcraft match, models for detection of these strategies must be trained, evaluated and then used in the run-time. During the creation of the models, we employed CRISP-DM [22], a standard and widely used methodology for solving of knowledge discovery tasks. Initial phase of the methodology put strong emphasis on the problem formulation and its transformation into the data-mining task. From the problem description (see section 2 Problem Domain) decided to approach the strategy detection as a predictive modeling task. Our approach will be based on training of a set of binary classification models on top of the obtained from the replays. We decided to choose the binary classification approach for several reasons. When expanding the model to the new strategies, such approach presents more flexible solution, as it is not needed to re-train and re-deploy the model (only a new model for particular strategy will be added). Also, the performance of binary models should be better as it is possible to tune the features and model parameters more specific to particular strategy. Following subsections describe the data understanding, pre-processing and modeling phases of the model building process.

A. Data acquisition and annotation

For the problem discussed in this paper, there was not a relevant annotated dataset currently available. Therefore, we needed to construct the dataset from available replays and select the relevant attributes for training of the models. The source of the data were replays of various matches during the SSCAI Tournament. The replays consisted of matches performed by various bots. Over 2961 replays were collected, each of them contained two custom bots made by various people in 1vs1 match. To get the data needed for training of the models, we used ScExtractor tool. The tool extracted all relevant information out of the selected replay files. Then, we developed a script, which plays every match again in real-time, frame by frame, in order to gain complete picture - record every possible state of any unit and any event which occurred during the game. Such database comprised complete game data of hundreds of replays and resulting dataset size was more than 7.6 GB.

B. Data pre-processing

In order to reduce the data size, we selected the tables and attributes relevant to discussed problem. Also, we processed the data using a set of functions to enhance them with specific attributes related to the player's race or types of units built. After that, the data were aggregated based on 5 second interval (cca 240 frames). Resulting dataset was then annotated using OpenBW replay-viewer module. When the particular strategy was occuring during the match, the corresponding records (target attributes) were flagged to value 1, until bot's strategy changed to something else. Final dataset then consisted of these attributes:

- PlayerReplayID: ID of player replay.
- AC_FRAME: frame number.
- Race: type of player's race.
- NumberOfBuildings: number of buildings constructed.
- NumberOfWorkers: number of workers trained.
- NumberOfAttackUnits: numbers of attack units trained.
- NumberOfAttacks: number of attack actions.
- RatioAttackToNon: ratio attack actions to non-attack actions.
- exp_feature1: Protoss_Forge is built?
- exp_feature2: Protoss_Pylon is build?
- exp_feature3: Protoss_PhotonCannon is build?
- exp_feature4: Is scout (unit first tagged as a scout) in the group?
- exp_feature5: Is scout near the enemy base?
- exp_feature6: Zerg_SpawningPool is build?
- exp_feature7: Zerg_Extractor is build?
- exp_feature8: Zerg_Zeglings are train?
- exp_feature9: How many Protoss_Gateway is build?
- exp_feature10: Protoss_Assimilator is build?
- exp_feature11: Protoss_CybernaticsCore is build?
- exp_feature12: Protoss_Zealots are train?
- exp_feature13: AttackUnits are near the enemy base?
- Results1: Strategy 1: Cannon rush is active?
- Results2: Strategy 2: Zergling rush is active?
- Results3: Strategy 3: 2-gate is active?
- Results4: Strategy 4: 3-gate is active?

C. Training of the models

To solve the problem defined within this paper, we picked three models different classification models to compare; Random Forests, Naive Bayes and Gradient Boosting Tree, in order to get different results. Each model was trained to solve the binary classification task to predict one of the target attributes (Results1-4). We used the implementation of the models from the sci-kit learn python library[8].

Feature selection was also performed for each of the trained models. As we used Random Forests, we extracted features importance from the initial model results. We used the data to

[8]http://scikit-learn.org/

TABLE I: Evaluation of the models for particular strategies.

Strategies	Models			
	RFT		GBM	
	P	R	P	R
Strategy 1	0.690	0.756	0.734	0.742
Strategy 2	0.930	0.832	0.992	0.832
Strategy 3	0.892	0.673	0.980	0.730
Strategy 4	0.928	0.860	0.957	0.886

select the attributes, which were not relevant to train the model for particular strategy. Fig. 2 depicts the features importance for model used to detect the Strategy 1 (Cannon Rush). In his case, we decided to remove the attributes *Race* and *NumberOfWorkers* from the training data. Similar approach was used to reduce the feature space for other strategies.

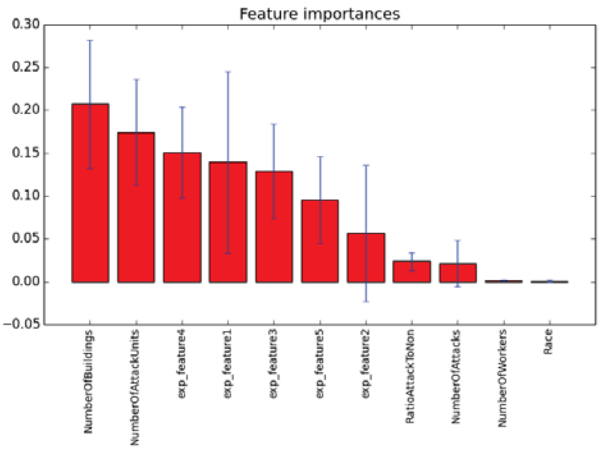

Fig. 2: Feature importance analysis for Strategy 1 (Cannon rush) V.

For training of the models for particular strategies, we used data from 108 replays which corresponded to 17778 records. During the model training, we used *GridSearchCV* method to examine the different combination of model hyper-parameters in order to find the best model with respect to the specified metrics.

VI. Evaluation

For evaluation purposes, we used standard metrics used to specify the model quality such as precision and recall. Those were specified as the main factors during the hyper-parameters tuning and were used to evaluate the final models on the testing set.

For training and evaluation of the models, we used 10-fold cross-validation approach. Tab. I summarizes the performance of the models Naive Bayes model is not included as its results were much worse, when comparing to Random Forests and GBM models. As it is can be seen from the results, GBM models performed well in each task.

By incorporating the metrics we were able to compare the different models. After closer consideration, we decided to use GBM models for several reasons. GBM gave almost constantly the best results in each considered metric on both datasets. he best GBM models were trained using those parameters:

- Strategy 1 model: Maximum features: 10, learning rate: 0.01, max. depth: 5, min. samples in the leaf: 10
- Strategy 2 model: Maximum features: 9, learning rate: 0.01, max. depth: 7, min. samples in the leaf: 50
- Strategy 3 model: Maximum features: 7, learning rate: 0.01, max. depth: 4, min. samples in the leaf: 1
- Strategy 4 model: Maximum features: 9, learning rate: 0.01, max. depth: 9, min. samples in the leaf: 20

We also considered time constraints - time to build the model and time to predict the newly arriving instances (in run-time). From that perspective, both Random Forests and GBM models achieved similar performance results, therefore running time was not considered as important feature when choosing the model.

Trained models then were deployed in the proposed system (see section IV. Proposed architecture). System obtains live data from the actual replay or game (played by both, bot or human player) sends it to the server which computes prediction for each of trained strategies. Results are retrieved to the client and players (or users watching a replay) are being notified about detected strategy. Integration of proposed solution was tested on various replays and game matches involved players and multiple bots.

VII. Conclusion

Presented paper introduced a system able to gather the data from StarCraft games and to detect the player's strategy using machine learning models. As a part of the system evaluation, several predictive models were trained using collected historical game data and used to identify specific strategy chosen by a player or a bot. Our approach is based on training binary classifiers for each covered strategy. Such approach enables to fluently extend the system with a wide range of models to identify other strategies (not covered in our use cases), or to update the existing models with more precise ones. Designed system also enables to deploy such models and use them on live data int the real-time gameplay. The work presented in this paper could also serve as a basis for implementation of more advanced approaches involving machine learning. We also investigated enhancement of the particular strategy identification with finding sequential patterns characteristic for each particular strategy or to combine the detection system with case-based reasoning approaches in order to create recommendation system which would advise the player a best counter-strategy for actual opponent.

Acknowledgment

This work was supported by Slovak Research and Development Agency under the contracts No. APVV-16-0213 and No. APVV-015-0731.

REFERENCES

[1] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for rts game combat scenarios." in *AIIDE*, 2012, pp. 112–117.

[2] Z. Wang, K. Q. Nguyen, R. Thawonmas, and F. Rinaldo, "Monte-carlo planning for unit control in starcraft," in *Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on*. IEEE, 2012, pp. 263–264.

[3] M. Certickỳ and M. Certickỳ, "Evolving reactive micromanagement controller for real-time strategy games," in *Proceedings of Scientific Conference of Young Researchers*, 2015, pp. 225–228.

[4] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[5] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 3106–3111.

[6] B. G. Weber and M. Mateas, "Case-based reasoning for build order in real-time strategy games." in *AIIDE*, 2009.

[7] M. Certickỳ and M. Certickỳ, "Case-based reasoning for army compositions in real-time strategy games," in *Proceedings of Scientific Conference of Young Researchers*, 2013, pp. 70–73.

[8] D. Churchill and M. Buro, "Build order optimization in starcraft." in *AIIDE*, 2011, pp. 14–19.

[9] N. Justesen and S. Risi, "Continual online evolutionary planning for in-game build order adaptation in starcraft," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 187–194.

[10] M. Certicky, "Implementing a wall-in building placement in starcraft with declarative programming," *arXiv preprint arXiv:1306.4460*, 2013.

[11] F. Richoux, A. Uriarte, and S. Ontanón, "Walling in strategy games via constraint optimization." in *AIIDE*, 2014.

[12] F. Richoux, A. Uriarte, and J.-F. Baffier, "Ghost: A combinatorial optimization framework for real-time problems," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 8, no. 4, pp. 377–388, 2016.

[13] A. Shantia, E. Begue, and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in starcraft," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 1794–1801.

[14] N. Usunier, G. Synnaeve, Z. Lin, and S. Chintala, "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks," *arXiv preprint arXiv:1609.02993*, 2016.

[15] K. Q. Nguyen, Z. Wang, and R. Thawonmas, "Potential flows for controlling scout units in starcraft," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–7.

[16] G. Synnaeve, P. Bessiere *et al.*, "A dataset for starcraft ai & an example of armies clustering," in *AIIDE Workshop on AI in Adversarial Real-time games*, vol. 2012, 2012.

[17] G. Robertson and I. D. Watson, "An improved dataset and extraction process for starcraft ai." in *FLAIRS Conference*, 2014.

[18] Z. Lin, J. Gehring, V. Khalidov, and G. Synnaeve, "Stardata: A starcraft ai research dataset," *arXiv preprint arXiv:1708.02139*, 2017.

[19] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 140–147.

[20] H. Park, H.-C. Cho, K. Lee, and K.-J. Kim, "Prediction of early stage opponents strategy for starcraft ai using scouting and machine learning," in *Proceedings of the Workshop at SIGGRAPH Asia*. ACM, 2012, pp. 7–12.

[21] H.-C. Cho, K.-J. Kim, and S.-B. Cho, "Replay-based strategy prediction and build order adaptation for starcraft ai bots," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–7.

[22] R. Wirth, "Crisp-dm: Towards a standard process model for data mining," in *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, 2000, pp. 29–39.