Prediction of Early Stage Opponents Strategy for StarCraft AI using Scouting and Machine Learning

Hyunsoo Park* — Dept. of Computer Engineering, Sejong Univ.
Ho-Chul Cho† — Dept. of Computer Engineering, Sejong Univ.
KwangYeol Lee‡ — Dept. of Electronics Engineering, Sejong Univ.
Kyung-Joong Kim§ — Dept. of Computer Engineering, Sejong Univ.

Abstract

StarCraft is one of the most famous Real-Time Strategy Games and there have been several competitions on AI bots. In order to win StarCraft, players have to predict their opponents strategy and respond properly. Human players used to scout their opponent territory using a unit and gathering information through direct observation to predict their opponents strategy. The accurate prediction of an opponents strategy gives players a big advantage in the early stage of a game. Usually, strategies of StarCraft can be divided into two parts: fast and slow attack strategies. Initial attack timing is an important factor of game strategies. In this paper, we apply a scouting algorithm and various machine learning algorithms to predict an opponents attack timing (strategy). Training data are collected from the games between our Xelnaga bot with the scouting algorithm and various online human players. Experimental results show that the machine learning approach based on realistic scouting data can be beneficial in predicting the opponents early-stage strategy.

CR Categories: I.2.1 [Artificial Intelligence]: Applications and Expert Systems—Games I.2.6 [Artificial Intelligence]: Learning—Knowlege Acquisition;

Keywords: StarCraft, game AI, scouting, strategy prediction, machine learning, artificial intelligence

1 Introduction

StarCraft is a famous Real-Time Strategy (RTS) game developed by Blizzard Entertainment in 1998. Academic conferences for game AI (IEEE CIG and AAAI AIIDE [1]) have open competitions for StarCraft AI [Kim and Cho 2012]. It is still quite challenging to develop AI for the game because it should handle a number of units and buildings while considering resource management and high-level tactics. Also, there is fog-of-war and the opponents territory is not visible. The only way to overcome that uncertainty is by sending units into the enemys area allowing limited visibility around the unit. It is highly dependent on the level of the players skills to do scouting and guessing the opponents strategy from the partial information.

Based on our experience at the CIG 2011 competition, it is common that many AI bots use only pre-fixed build orders without scouting. There is no change of their strategy even when they play again against the same opponent. Although it is desirable to scout, infer the others strategy and change ones own strategy, it is challenging to implement these.

In 2011, we submitted the Xelnaga bot to the CIG 2011 StarCraft AI competition. The program is based on several build orders carefully chosen from experienced human players. Although the bot uses only pre-fixed build orders, it is strong compared to a number of other bots. However, the bot is weak against an opponent with a very fast, unexpected attack or very good defensive behavior. So, we recognized our bot needs to predict an opponents initial strategy, especially the opponents attack timing.

Weber et al. propose to use professional human players replay files as a source of data mining [Weber and Mateas 2009]. It is possible to get text-based information on players actions in the game using specialized programs (replay browser [2]). In the work, they report that machine learning can be useful to predict the strategy of an opponent in the early stage of the game. In the analysis, they pose a strong assumption on the visibility of information. In the first analysis, they assume that there is no fog-of-war and all the information of the opponent is used to predict strategy. In the second experiment, they add random noise and missing data to simulate the fog-of-war. This is still not realistic. Besides this work, there have been several others that use the replay files to learn the strategy prediction models [Hsieh and Sun 2008; Kim and Cho 2011]. However, they assume the same unrealistic settings.

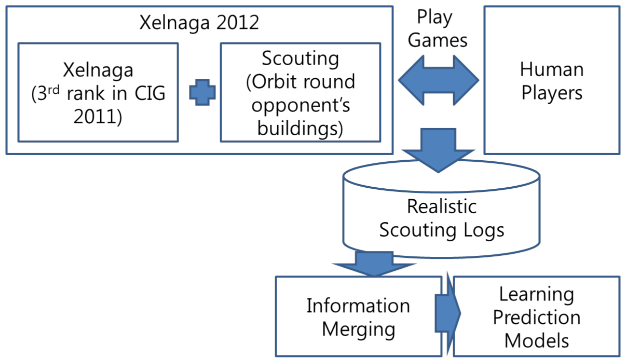

Xelnaga 2012
Xelnaga (3rd rank in CIG 2011)
Scouting (Orbit round opponent's buildings)
Play Games
Human Players
Realistic Scouting Logs
Information Merging
Learning Prediction Models

Figure 1: *An overview of the proposed methods.*

In this work, we plan to collect realistic data to predict an opponents strategy. The first step is adding scouting functionality to our bot. It needs navigation of the scouting unit to maximize survival time and information coverage. Instead of using professional human players replay files, our bot plays a number of games against common human players. Our program automatically records only visible units and buildings of the human player opponent periodically while playing the game. In this way, we can collect realistic

*e-mail:hspark@sju.ac.kr
†e-mail:chc2212@naver.com
‡e-mail:primal13@daum.net
§e-mail:kimkj@sejong.ac.kr ,corresponding author
[1]http://eis.ucsc.edu/StarCraftAICompetition/
[2]http://lmrb.net/

data to train prediction models. Each game is labeled as fast or slow attack automatically (If the attack is faster than our attack, it is defined as fast). Several machine learning algorithms are trained to learn the strategy inference functions. Figure 1 shows the schematic diagram of the proposed methods.

2 Background

2.1 StarCraft AI Competition and Xelnaga

The first StarCraft AI competition was at the AIIDE 2010 (AAAI conference on Artificial Intelligence and Interactive Digital Entertainment). At the event, more than 26 teams registered. In the IEEE Conference on Computational Intelligence and Games (CIG 2011), there was a StarCraft AI competition. In the event, ten teams registered. Xelnaga (Our team) was ranked 3rd in this event. It is a pure rule-based system and its race is Protoss. The main build order was designed by Ho-Chul Cho (4292 wins and 1521 losses on Battle.net). It chooses different strategy depending on the race of opponents. In the final round of the CIG 2011 [3], there were Skynet, UAB, Xelnaga and BotQ [Synnaeve and Bessiere 2011]. They played 30 games against each other. Xelnaga won 11 games. Careful analysis shows that Xelnaga is weak against Skynet and UAB. Xelnaga lost games if there were very early attacks or defensive play. From this observation, we determined to implement scouting and early detection of opponents strategy using machine learning.

2.2 StarCraft scouting

In StarCraft, experienced human players start the game by sending their construction units outside of their territory. The purpose of this scouting is to find the territory of opponents. In the game, the players position is randomly chosen. If the human player plans to execute a fast attack, it significantly changes the build order. It focuses on massive production of attack units as soon as possible and gives up defensive or long-term strategies. If the human player performed offense successfully in a fast attack, he or she easily wins the game. However, there is the possibility that this fast attack can fail when the opponent predicts player's attack. In that case, it is quite difficult to change the final outcome. The purpose of scouting is to see the buildings and units in the opponents area. If the scouting unit successfully enters into the area, it removes the fog-of-war while the scouting unit survives. Although the revealed area is limited to the area around the unit, it is very useful to get the opponents build orders. The appearance of the scouting unit is not good news to the opponent players and they try to kill the scouting unit and delay their construction of critical units to hide their build orders.

Table 1 summarizes the state-of-the-art for scouting for the StarCraft AI bots. It shows that only a small number of participants implement scouting. Most of them use scouting to disturb opponents. In those cases, it is not necessary to collect much information. SPARs scouting behavior is also similar to the humans one. However, there is no article about their approaches.

2.3 Prediction of opponent's strategy using replays

Hsieh et al. collected thousands of StarCraft game replays from GosuGamers.net game community [Hsieh and Sun 2008]. They used case-based reasoning to learn and predict individual players strategies. In their work, cases in CBR contain constructed buildings, units and other resources. They reported that predictive accuracy increased when more replays were inputted into their decision system.

[3]http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2011

Table 1: *Comparison of StarCraft AI bots scouting.*

AI Player	Race	Scouting	Comments
Aiur	Protoss	X	
BTHAI	Zerg	X	
EvoBot	Terran	X	
Nova	Terran	O	Disturbance
Skynet	Protoss	O	Disturbance
BotQ	Protoss	X	
LSAI	Zerg	O	Overload
UalbertaBot	Protoss	O	Disturbance
Beast Jelly	Protoss	X	
Bigbrother	Zerg	X	
Cromulent	Terran	X	
EISBot [Weber et al. 2011]	Protoss	X	
ItalyUndermind	Zerg	O	Disturbance
Quorum	Terran	X	
SPAR	Protoss	O	Observation
Undermind	Terran	X	

Weber et al. collected 5493 StarCraft game replays from GosuGamers.net and TeamLiquid.net, two popular StarCraft websites [Weber and Mateas 2009]. They encoded replays as feature vectors and labeled them using rules based on analysis of expert play. They represented strategy prediction as a multi-class classification problem and applied J48, k-NN, NNge, and LogitBoost. They reported that different algorithms are better at different stages of the game. NNge and k-NN performed well in the initial stage of the game, but degrade in the later stages of the game.

Ontanon et al. used StarCraft replays in order to build a case library for D2, a real-time case-based planning system designed to play RTS games [Weber and Ontañón 2010]. They defined goal ontology for StarCraft and developed a rule set for recognizing when goals are being pursued. The goal ontology was formulated based on analysis of professional StarCraft game play.

Kim et al. tried to analyze build-orders and the relations among them from game replays [Kim et al. 2010]. For example, they figured out how many times build-order A won against build-order B. They used a similarity measure on unit production, building construction, and upgrade order to categorize build-orders. They united build-orders as a tree to choose an appropriate one based on an enemys state and winning ratio.

3 Proposed methods

3.1 Overview of scouting

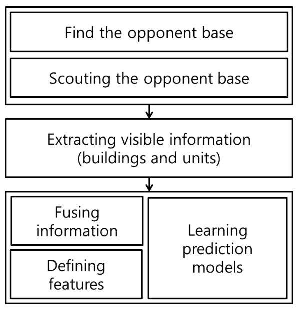

Find the opponent base
Scouting the opponent base
Extracting visible information (buildings and units)
Fusing information
Defining features
Learning prediction models

Figure 2: *An overview of machine learning procedure.*

Our proposed method consisted of three steps. Figure 2 shows an overview of these steps. (1) Find the opponent base using one of the initial construction unit and the unit should survive as long as possible while collecting the opponents structures/units information. If the scouting unit is destroyed by the opponent, the data collection is finished. (2) The next step is to extract important information from the observed raw data. It is necessary to define features for training classifiers. And the view from the scouting unit covers only small parts of the opponents area, it is necessary to fuse the information. (3) The final step is to generate a prediction model from the information.

3.2 Scouting algorithms

3.2.1 Navigation

The purpose of this step is to find the opponents base and collect data about it. The scouting unit is a worker (resource collector) from one of the races. Instead of mining the resources, they explore the dark area to find the opponents base. The first step is to find the location. There are several possible areas for the opponents position on the general game map. A scouting unit checks all candidate areas.

After finding the opponent base area, then the recon unit scouts the area. There are two important goals in this step. (1) The recon unit should survive in the hostile area as long as possible. (2) The unit should reveal as many of the opponents structures/units as possible. However, it is not trivial to satisfy both of them. To collect data precisely, the recon unit should be close to the opponents structures/units. But, this increases the risk that the recon unit will be damaged or destroyed. So, the recon unit has to be close to them, but not too close to be attacked.

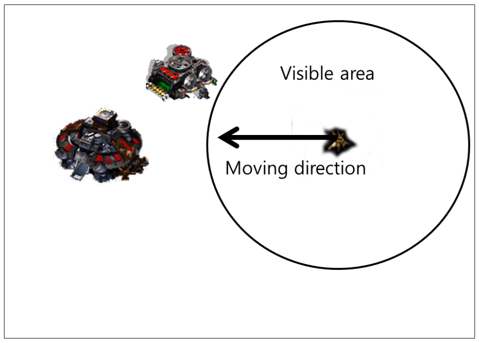
Our recon unit is mimicking the human players scouting. They continuously rotate the area keeping constant distance to the buildings. If the unit is inside the opponents area, the first task is to go to the main building (Nexus, Command Center, or Hatchery). When the opponents building hits the visible area of the scouting unit, it changes the navigation path. We only consider an opponents buildings to decide the direction of movement. When the recon unit sees an opponents building, it changes its direction 90° CCW (Counter Clock Wise) or CW (Clock Wise). So, the recon unit orbits round the opponents structure. CCW or CW is not important in this method.

Ideally, the recon unit orbits perfectly around the opponents building keeping the same distance. However, the recon unit is not rotating in an ideal circle. This is caused by the update rate (update per 13 game ticks). If the update rate is increased, the recon units movements are close to the circle but it consumes much computational resource (critical to a real-time game). Instead of adjusting the updating rate, we change the rotation angle. Testing on 75°, 80°, 85°, and 90° show that 80° is the best one.

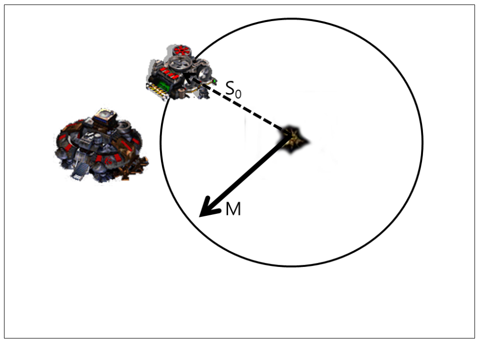
When there is more than one opponent structure in the recon units view, it regards them as one big virtual building. If there are N opponents buildings in the recon units view, each building is identified as $O_1, O_2, O_3, ,$ and O_N. The position of the O_i is defined as (x_i, y_i). If the position of the recon unit is defined as (rx, ry), the vector for each building is defined as $V_i(x_i - rx, y_i - ry)$.

$$V = \sum_{i=1}^{N} V_i \tag{1}$$

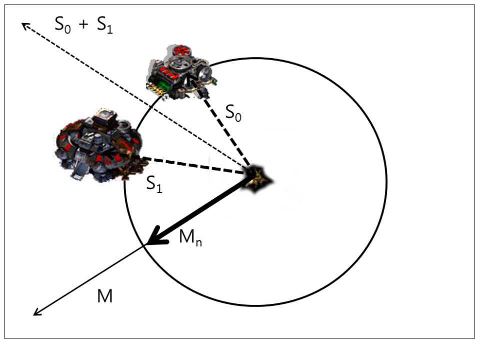
V Rotate CCW with 80°, the final vector is the direction of the recon unit (Figure 3).



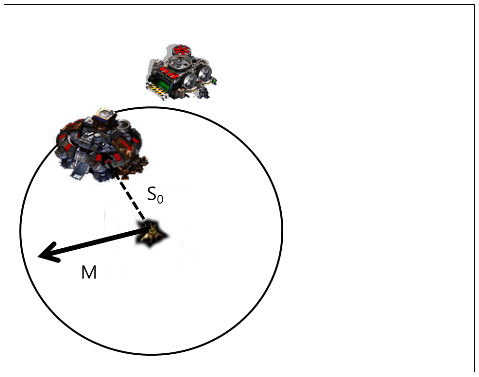
(a) The recon unit is moving to the command center.



(b) The recon unit sees one opponents structure. The direction is changed to M.



(c) If there are two structures in the vision area, it calculates the S0+S1 and change the direction to M.



(d) If there is only one building, it calculates the direction M only based on it.

Figure 3: *An example of a scouting path calculation.*

Our controller continuously records the observed buildings and units per second while the scouting unit is alive. If the unit is killed by an attack by the opponent, the data collection is finished. Table 2 shows an example of recorded raw data from the scouting units. The race of the opponent is Terran. At 122 seconds from the beginning of the game, the recon unit sees one Terran_SCV unit (Its ID is 138). At 123 seconds, there are SCV (ID=138) and Command_Center (ID=139) in the view. At 124 seconds, seven units/buildings are visible.

Table 2: *An example of raw data.*

Time	Building/Unit	ID
122	Terran_SCV	138
123	Terran_SCV	138
123	Terran_Command_Center	139
124	Terran_SCV	138
124	Terran_SCV	144
124	Terran_Barracks	147
124	Terran_SCV	143
124	Terran_SCV	146
124	Terran_SCV	142
124	Terran_Command_Center	139
125	Terran_SCV	144
125	Terran_Barracks	147
...	...	...

3.2.2 Information fusion

The raw data contains all the information observed during the scouting. However, it is not easy to make a decision on the opponents strategy from the raw data. It is necessary to fuse information from the multiple scenes into a summarized one. In this work, we propose to use two different fusing methods.

Table 3: *Examples of preprocessed data.*

(a) An example of count data.

Structure/Units	Count
Terran_Command_Center	1
Terran_Marine	2
Terran_Barracks	1
Terran_Refinery	1
Terran_Supply_Depot	3
Terran_SCV	20
Terran_Factory	1

(b) An example of time data.

Structure/Units	Time
Terran_Command_Center_1	86
Terran_SCV_1	87
Terran_SCV_2	87
Terran_SCV_3	88
Terran_Refinery_1	91
Terran_Supply_Depot_1	94
Terran_Barracks_1	110
Terran_Supply_Depot_2	140
Terran_Factory_1	169
Terran_Marine_1	184
Terran_Marine_2	203
Terran_Supply_Depot_3	208

Figure 3(a) shows the count-based data which counts the number of unique objects observed during the scouting periods until the recon unit is destroyed. For example, the number of Command_Center is one and the number of SCV during the scouting is 20. Because of the unique ID, it is possible to count the number of objects correctly. The data show the existence of specific objects and the mass of them.

Figure 3(b) shows a different view of the raw data by considering the time information. In the data, it records the time that the buildings or units are observed. For example, the Terran_Command_Center_1 means the time that the first Command_Center is observed at 86 seconds. SCV_1, SCV_2, and SCV_3 record the observed time for first SCV, second SCV, and third SCV observed, respectively. From the data, it is not easy to figure out the mass of the units or buildings because it only records the times for the first two or three units or buildings.

3.2.3 Prediction of opponent's strategy

The final stage is to make a decision on the opponents strategy from the fused data. If expert knowledge is available, it can be designed using simple rules. But, it is not trivial to design such rules. First of all, such expert knowledge is not easily defined. Secondly, the information from the scouting unit is not perfect and there is uncertain (unobserved or hidden) information.

The solution to this problem is applying machine learning. Simply, we categorize the opponent strategy in the training data into fast or slow. If the opponent attacks before Xelnaga attacks, it is fast and vice versa. The raw data are converted into summarized form using the preprocessing (information fusion) algorithms. Finally, a set of machine learning algorithms is applied on the summarized data to learn the mapping from the observed information into a class (fast or slow). In this work, the machine learning algorithm is implemented using WEKA [Witten et al. 2011].

4 Experimental results and discussion

4.1 Data collection

The most difficult problem for the StarCraft data mining is to collect data. Because we need data to train our scouting-based bot, the replay files accessible on the internet are not useful. Although we considered generating the data by playing games between our bot and other bots, there are not enough bots available and most of the bots strategies and behavior are too simple. Our choice was to collect the data from the games between our bot and many human players. The largest place to play games is Blizzards official Battle.net. However, Chaos Launcher and BWAPI used for our bot are detected as a cheating program and the connection is refused.

We collected data from a free Battle.net source (private Battle.net server, http://brainclan.com). On the private server, it is possible to connect but human players usually have cheating detection tools and refuse games with our bot. Thus, there were a very small number of cases that a player accepted our bot. In this way, we played 105 games against human players. We used two popular game maps (Python 1.3 and Fighting Spirit 1.3) on this free server. The games were converted into the count-based and time-based training data file (ARFF format for WEKA) for each opponents race.

Table 4: *Summary of collected data (win/lose)*

	vsTerran	vsProtoss	vsZerg	Total
Fast	10 (0/10)	19 (0/19)	3 (1/2)	**32 (1/31)**
Slow	17 (5/12)	30 (10/20)	26 (7/19)	**73 (22/51)**
Total	**27 (5/22)**	**49 (10/39)**	**29 (8/21)**	**105 (23/82)**

Table 4 summarizes the statistics of the games. It shows that our bot won 23 games (approximately 22%). Because our bot is weak on the fast attack, most of games identified as fast were lost. However, for the slow case, the winning ratio was 43%. The players on the free Battle.net are not professional level but theyre not novice because they have played the game for a very long time.

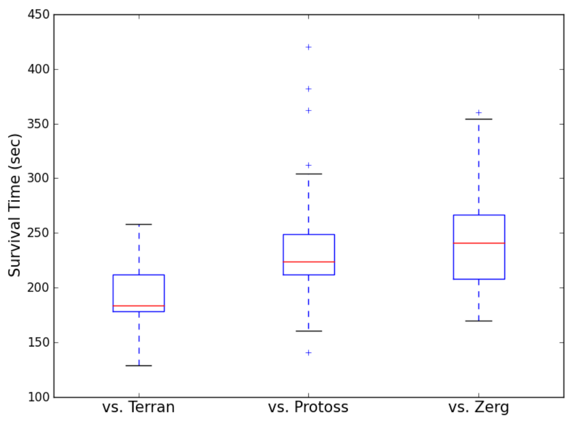


Figure 4: *Survival time of recon unit.*

Every recon unit in all experiments successfully scouted the opponent base. But when the opponent player trains attack units (ex. marine, zealot, or zergling), the opponent player tries to destroy the recon unit. So, all recon units might not be able to avoid destruction. If the recon unit survived for a long time, the recon unit could get lots of valuable information. Figure 4 shows the comparison of survival time of the recon unit in the opponents base according to the opponents race.

It shows that a Terran player destroys the recon unit in about 200 seconds. It is earlier than with Protoss and Zerg. This is because the Terran trains Marines with a long-range weapon. Our recon unit Probe is easily destroyed by the long-range weapon. It is interesting that there was a small number of fast games against Zerg and the recon unit survived for a long time because Zerg cant train for a long-range weapon attack unit in the early stage of the game, so Zerg players had difficulty destroying our recon unit. Zerg players seem to give up the fast strategy because our recon unit survived for a long time and exposed lots of information.

4.2 Machine learning

We have applied thirteen machine learning algorithms to the preprocessing data. Theyre categorized into rules (ZeroR and oneR), decision tree (J48, Cart, Decision Stump and Random Forest), lazy learning (1-NN, and NNge), neural networks (MLP), probabilistic models (Bayesian Network and Nave Bayes), and SVM. Because the number of samples is small, LOOCV (Leave-One-Out-Cross Validation) is used. For each classifier, default parameters in WEKA are used.

Figure 5 shows accuracy of the classification algorithms with three different feature extraction methods. For the Count features, it counts the number of unique buildings/units observed. For the Time features, it records the time information for each event observed. And the Count+Time feature is a combination of all features.

When the opponent is a Terran player, the best classification algorithm is ZeroR which simply classifies samples into the most frequent classes. It means that in the case of Terran, the machine learning is not so useful. Because of this, the recon unit couldnt survive long enough. Terran players Marine (first attack unit) can destroy the bots recon unit easily. Thus the recon unit cant observe the opponents next buildings (another Barracks or Factory), but these buildings are important to specify the opponents strategy. So, it is not possible to predict whether the strategy is fast or not.

When the opponent is Protoss, we have 49 samples. The results show that the best combinations are Decision Stump + Time, Cart + Count, and MLP + Count. The accuracy is 73.4%. It shows that the proper choice of the features is important to get the best performance for each classification algorithm. For example, ADTree shows radical performance improvement if it uses the Count features. Nine of the thirteen classification algorithms show better performance if they use the Count features. The combination of the Count and Time features is not so successful.

When the opponent is Zerg, we have the unbalanced data set. The number of Fast cases is only three. Because we collected the data by playing games against human players, it is not possible to control the choice of his/her strategy. The results show that J48 and ADTree perfectly classify the samples (100%).

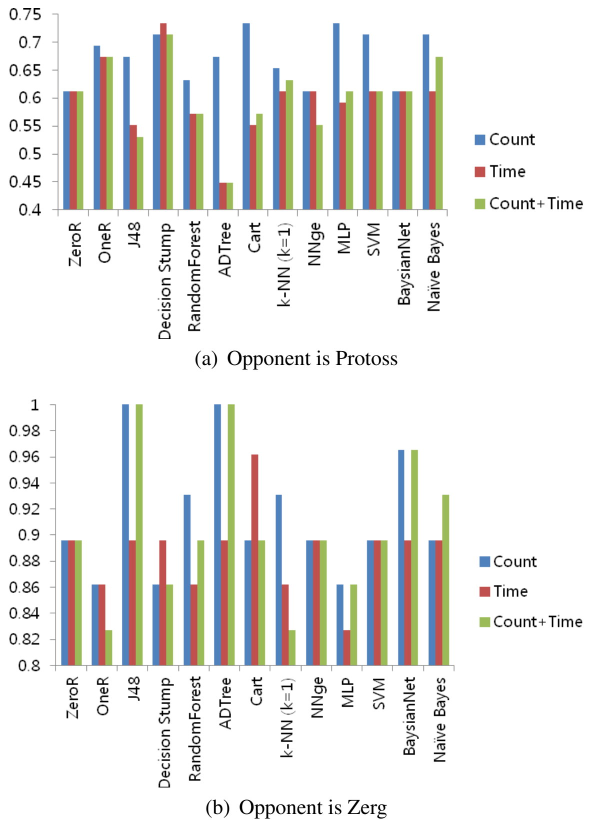


Figure 5: *Classification accuracy of various machine learning algorithms with different feature extraction methods.*

5 Conclusions and future works

Scouting is one of the most important parts of real-time strategy games with fog-of-war. However, it is still in the early stage of development for StarCraft AI competitions. In fact, our team ignored scouting in the CIG 2011 competition but ranked 3rd. However, it is essential to send a scouting unit to recognize the build order of opponents in the early stage of the game to become a strong AI player. In this paper, we introduce a basic navigation strategy and the use of machine learning to design automatically the recognition module for opponents strategy.

After implementing the basic navigation strategy for our bot, we played more than 100 games against human players through a free Battle.net server. Using the realistic data, we introduced two different feature extraction methods and applied several machine learning algorithms. Although expert knowledge is essential to recognize an opponents strategy from the observation, it is possible to build the recognition part automatically with the help of machine learning.

Our navigation strategy is not perfect because it ignores the movement of opponents units. It only considers the position of an opponents buildings. Also, in the navigation strategy, we assume that all the buildings have the same level of danger and importance. This is a strong assumption and needs to be modified. Human players can survive for a longer time than our bot from these results. It is necessary to control the scouting unit considering buildings and the position of attack units. Also, it is desirable to analyze the terrain to plan the path of the scouting unit. However, because it is a real-time strategy game, it must be simplified to response quickly. Although our work focuses on the early stage of scouting, it is still important in the late stage of the game. However, the scouting in the late time is more complex than the one in the early stage.

Acknowledgements

This research was supported by Basic Science Research Program and the Original Technology Research Program for Brain Science through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012-0001749) (2012-0005799)

References

HSIEH, J.-L., AND SUN, C.-T. 2008. Building a player strategy model by analyzing replays of real-time strategy games. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*, IEEE, 3106–3111.

KIM, K.-J., AND CHO, S.-B. 2011. Server-side early detection of starcraft players with non-standard strategic behavior. In *3rd International Conference on Internet*.

KIM, K.-J., AND CHO, S.-B. 2012. 2011 ieee conference on computational intelligence and games [conference reports]. *Computational Intelligence Magazine, IEEE 7*, 1 (feb.), 15 –18.

KIM, J., YOON, K. H., YOON, T., AND LEE, J.-H. 2010. Cooperative learning by replay files in real-time strategy game. In *Proceedings of the 7th international conference on Cooperative design, visualization, and engineering*, Springer-Verlag, Berlin, Heidelberg, CDVE'10, 47–51.

SYNNAEVE, G., AND BESSIERE, P. 2011. A bayesian model for opening prediction in rts games with application to starcraft. In *CIG'11*, 281–288.

WEBER, B. G., AND MATEAS, M. 2009. A data mining approach to strategy prediction. In *CIG'09: Proceedings of the 5th international conference on Computational Intelligence and Games*, IEEE Press, Piscataway, NJ, USA, IEEE Press, 140–147.

WEBER, B. G., AND ONTAÑÓN, S. 2010. Using automated replay annotation for case-based planning in games. In *ICCBR Workshop on CBR for Computer Games (ICCBR-Games)*.

WEBER, B. G., MATEAS, M., AND JHALA, A. 2011. Building human-level ai for real-time strategy games. In *Proceedings of the AAAI Fall Symposium on Advances in Cognitive Systems*, AAAI Press, San Francisco, California, AAAI Press.

WITTEN, I. H., FRANK, E., AND HALL, M. A. 2011. *Data Mining: Practical Machine Learning Tools and Techniques*, 3 ed. Morgan Kaufmann, Amsterdam.