A Data Mining Approach to Strategy Prediction

Ben G. Weber and Michael Mateas

Abstract— **We present a data mining approach to opponent modeling in strategy games. Expert gameplay is learned by applying machine learning techniques to large collections of game logs. This approach enables domain independent algorithms to acquire domain knowledge and perform opponent modeling. Machine learning algorithms are applied to the task of detecting an opponent's strategy before it is executed and predicting when an opponent will perform strategic actions. Our approach involves encoding game logs as a feature vector representation, where each feature describes when a unit or building type is first produced. We compare our representation to a state lattice representation in perfect and imperfect information environments and the results show that our representation has higher predictive capabilities and is more tolerant of noise. We also discuss how to incorporate our data mining approach into a full game playing agent.**

I. Introduction

One of the main challenges for game designers is creating adaptive AI opponents that react to player actions. For real-time strategy (RTS) games, an intelligent AI opponent should be capable of responding to a player's strategy with a counter-strategy. For this process to occur, the AI opponent must first identify the player's strategy. In this paper, we explore a data mining approach to recognizing strategies in a RTS game with partial information.

StarCraft[1] is a science fiction RTS game developed by Blizzard Entertainment^{TM} with three diverse races (Protoss, Terran and Zerg). The game provides functionality to save replays for further review and analysis. Several international competitions are held for StarCraft (World Cyber Games, BlizzCon, IeSF Invitational); South Korea even has a professional league devoted to the game. The popularity of StarCraft, combined with the ability to save replays, has resulted in large collections of game logs that are available for analysis.

The unit types and buildings in RTS games are defined by a tech tree, which is a directed graph without cycles that models the possible paths of research a player can take within a game [1]. A major strategic element in RTS games is build order, which defines the order in which a player expands their tech tree. Build orders target a specific strategy, such as rushing or a timing attack. Rush strategies attempt to overwhelm an opponent with inexpensive units early in the game, while timing attacks engage opponents based on a trigger, such as completing an upgrade. Build orders are a knowledge rich aspect of RTS games. It is common for players to study replays in order to improve their skills and understanding of the game. Professional gamers are known to study the replays of an opponent before an important match, much like a chess grandmaster preparing for a match. Players select a strategy going into a match based on the map, opposing race and predictions of the opponent's strategy.

RTS games enforce imperfect information through a "fog of war", which limits visibility to portions of the map where the player controls units. An important aspect of RTS games is scouting opponents in order to find out which portions of the tech tree have been expanded, enabling the player to develop counter-strategies. It is also necessary to determine when buildings are constructed, because different timings can lead to different types of strategies. Players often utilize strategies such as hiding buildings in order to confuse an opponent or proxying production buildings in order to produce offensive units within an opponent's base. At the highest levels of StarCraft gameplay, players are capable of predicting hidden buildings due to well-known timings of different strategies.

We present a data mining approach for learning strategies in StarCraft. A web crawler was developed to collect a large number of replays from professional and top-ranked amateur players. The replays are converted from a proprietary format into a log of game actions. Each player's actions are encoded as a single feature vector and labeled with a specific strategy using a set of rules based on analysis of expert gameplay. Several machine learning algorithms are then applied to the task of detecting an opponent's strategy and estimating when an opponent will perform actions.

The remainder of this paper is structured as follows. In Section 2 we discuss related work. Section 3 discusses data mining replays and introduces our representation. We then evaluate different algorithms for strategy prediction in Section 4. Finally, we provide conclusions and future work in Section 5.

II. Related Work

There are several techniques for opponent modeling in games. Most work has focused on either tactical or strategic aspects of gameplay. At the tactical level, opponent modeling has been applied to predicting opponent positioning in first-person shooters [2] and Pong [3]. At the strategic level, plan recognition has been applied to predicting an opponent's actions, while various techniques have been applied to automatically learn domain knowledge.

A. Plan Recognition

Plan recognition is the process whereby an agent observes the actions of another agent with the objective of inferring the agent's future actions, intentions or goals [4]. Bayesian

Ben Weber and Michael Mateas are with the Expressive Intelligence Studio at the University of California, Santa Cruz. 1156 High Street, Santa Cruz, CA, USA (email: bweber@soe.ucsc.edu, michaelm@soe.ucsc.edu).

[1]http://www.blizzard.com/us/starcraft/

networks and case-based plan recognition have been applied to plan recognition in games.

Bayesian networks are a probabilistic approach to plan recognition. Explanations are assembled into a plan recognition Bayesian network, which is a representation of a probability distribution over the set of possible explanations [5]. Albrecht et al. apply dynamic belief networks to predict the player's current goal, next action and next location in a multi-user dungeon adventure game [6]. Their representation enables the use of incomplete and noisy data during both training and testing, while supporting a stateful model. The main problem with Bayesian networks is identifying the appropriate network structure. The results of Albrecht et al. suggest that dynamic belief networks offer a promising approach to plan recognition in situations where the causal structure of the network can be clearly identified [7].

Case-based plan recognition is an experience-based approach to plan recognition, where case libraries are constructed by observing game play. Case-based plan recognition has been applied to player modeling in Space Invaders [4]. Each sequence of actions is assigned a support count, which is used to identify common strategies. Action sequences with a high support count are marked as plans, added to the case library and used to predict a player's next action. While this approach is effective for Space Invaders, it is unlikely to scale to real-time strategy games, as such games exhibit a huge increase in the number and complexity of possible action sequences. The possible actions must be encoded as a state-transition table and there are several parameters to tune, including the support count threshold and minimum plan length. Cheng and Thawonmas investigate the application of this approach to assisting human players with low-level actions in a RTS game [8].

B. Learning Domain Knowledge

Two approaches have been applied to learning domain knowledge in RTS games: systems that rely on exploration of the game space and systems that utilize game replays to automatically acquire domain knowledge.

Due to the vast game space of RTS games, exploratory approaches have focused on individual aspects of gameplay. Reinforcement learning [9] and Monte Carlo [10] methods have shown promising results for tactical aspects in RTS games, while genetic algorithms [11] and case-based reasoning [12] have been applied to strategy selection. Techniques for strategy selection have relied on domain-specific knowledge representations, such as a state-lattice [13], to limit the types of strategies that are explored.

Recent work has investigated the use of game logs to automatically learn expert gameplay. Ontañón et al. introduce a case-based planning system that generates a case library from a set of game traces [14]. However, the system utilized a small number of game traces and ran into problems when dealing with a large collection of game traces representing several strategies played on a variety of maps [15].

III. Data Mining Replays

Several websites are dedicated to collecting and sharing StarCraft replays with the gaming community, a large portion of which are from professional and high-ranked amateur matches. Therefore, it is possible to mine these websites in order to build a collection of replays that is a representative set of expert play in StarCraft. Due to the large number of replays available, it is possible to learn a variety of strategies on several maps against different play styles. In the remainder of this section, we describe our data mining approach to automatically learning strategies from collections of replays and discuss some of the challenges faced.

We developed a web crawler to collect StarCraft replays from GosuGamers.net and TeamLiquid.net, two popular StarCraft websites. The Web crawler downloaded collections of replays from professional tournaments including BlizzCon, World Cyber Games, MBC Starleague and the StarCraft Proleague. The crawler also collected replays from top-ranked players on ICCup.com, a popular StarCraft ladder ranking system. We limited our focus to one-versus-one matches, because it is the most common game type for professional StarCraft matches. The resulting number of game traces for the different races are shown in Table I.

TABLE I
Replays collected

Type	# Replays
Protoss vs. Protoss	542
Protoss vs. Terran	1139
Protoss vs. Zerg	1024
Terran vs. Terran	628
Terran vs. Zerg	1150
Zerg vs. Zerg	1010
Total	5493

Our goal is to build a general model of expert StarCraft gameplay. This differs from previous work, which has focused on modeling single opponents [4] or work that has been limited to at most a few hundred game logs [16]. By applying data mining to a large number of game logs, we can develop an opponent model that is not limited to a single opponent, set of maps or style of gameplay.

A. StarCraft Replays

StarCraft replays are stored in a proprietary, binary format. We used the LordMartin Replay Browser[2] to convert the replay files to game logs. A subset of an example log is shown in Table II. The game logs contain only user interface actions performed by each player. Game state is not available in the logs, because replays are viewed by performing a deterministic simulation based on the user interface actions. This limits how much domain knowledge can be extracted from replays, because information, such as the player's current amount of resources, is not available for analysis. However, the production of different unit types and building

[2]http://lmrb.net/

TABLE II
A PARTIAL GAME LOG FROM A STARCRAFT REPLAY

Player	Game Time (minutes)	Action
Player 2	0:00	Train Drone
Player 1	0:00	Train SCV
Player 2	1:18	Train Overlord
Player 1	1:22	Build Supply Depot
Player 1	2:04	Build Barracks
Player 2	2:25	Build Hatchery
Player 1	2:50	Build Barracks
Player 2	2:54	Build Spawning Pool
Player 1	3:18	Train Marine
Player 2	4:10	Train Zergling

types can be extracted, providing sufficient information to analyze a player's build order.

The lack of game state provides a challenge for opponent modeling, because only the player's user interface actions are available for analysis. One approach to overcome this limitation is the use of a state lattice to capture the sequence in which actions are performed. State lattices have been applied to opponent modeling in Wargus [13] and StarCraft [16]. A state lattice is a directed graph without cycles, where each node represents a unique expansion of the tech tree. State lattices are built in a similar manner to constructing decision trees (see [16] for a more detailed explanation). State lattices are useful for predicting the next action given a sequence of actions. However, state lattices are unable to predict when the next action will occur.

B. Encoding Replays as Feature Vectors

The goal of our representation is to capture the timing aspects of a player's strategic decisions in a game, enabling the prediction of distinct strategies and the timing of the opponent's actions. Our feature-vector representation contains temporal features that record when the player expands different branches of the tech tree. Each feature describes when a unit type or building type is first produced.

For each game log, two feature vectors are constructed. Each vector represents a single player's actions for an entire game. Formally, our representation is a feature vector, where each feature f, for a player P, is defined as follows:

$$f(x) = \begin{cases} t & : time\ when\ x\ is\ first\ produced\ by\ P \\ 0 & : x\ was\ not\ (yet)\ produced\ by\ P \end{cases}$$

where x is a unit type, building type or unit upgrade. Each race has a different number of features, based on the tech tree and upgrades. For example, our Protoss representation contains 56 features. A subset of an example feature vector for a Zerg player is shown in Table III. The Queen feature has a value of 0:00, because a queen was not produced by the player. This encoding, while not requiring information about the game state, captures both the tech tree expansion and timing of a player's strategy. This representation varies from related work which encodes single game traces as several cases [14], [17].

TABLE III
A SUBSET OF AN EXAMPLE FEATURE VECTOR

Action (Attribute)	Game Time (Value)
Second Hatchery	2:42
Spawning Pool	3:07
Lair	5:48
Zergling	4:23
Zergling Speed	12:10
Hydralisk Den	12:52
Hydralisk	14:51
Lurker	15:39
Hydralisk Speed	16:48
Hydralisk Range	20:01
Queen	0:00

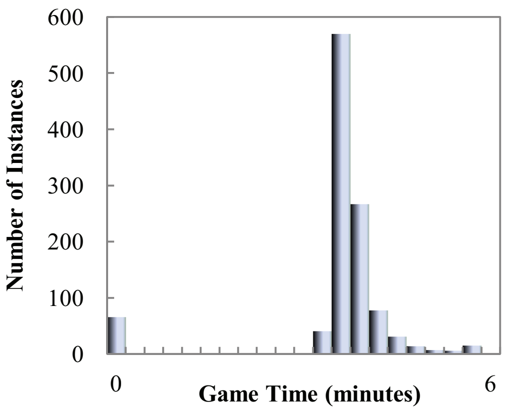

Fig. 1. Factory timing in Terran versus Protoss games

We plotted timing distributions of several features to determine if there are timing patterns, which can be used to predict opponent timing. For certain features, there are specific timings that most players follow, such as factory timing in Terran versus Protoss games (see Figure 1). For other features, there are a wider variety of timings due to branches in strategies, such as spawning pool timing in Zerg versus Terran games (see Figure 2).

C. Labeling Replays

The game logs are labeled with a strategy using rules based on analysis of expert play. Different rule sets are used for labeling the different races. Each rule set is designed to capture the tier 2 strategies of a race. The rule sets label logs with strategies based on the order in which building types are produced. The rule set for labeling Protoss strategies is shown in Figure 3. In the figure, tier 1 strategy refers to strategic decisions made in the early stages of the game. The strategy of the player is not labeled until a tier 2 strategy decision is made, such as building a Stargate, which is labeled as a "Fast Air" strategy. Six strategies were created for each race. If a game does not fit any of the rules, then the strategy is labeled as unknown.

The rule sets were designed to capture a wide variety of strategies in StarCraft. Distributions of the builds versus different races are shown for Protoss in Table IV. In certain

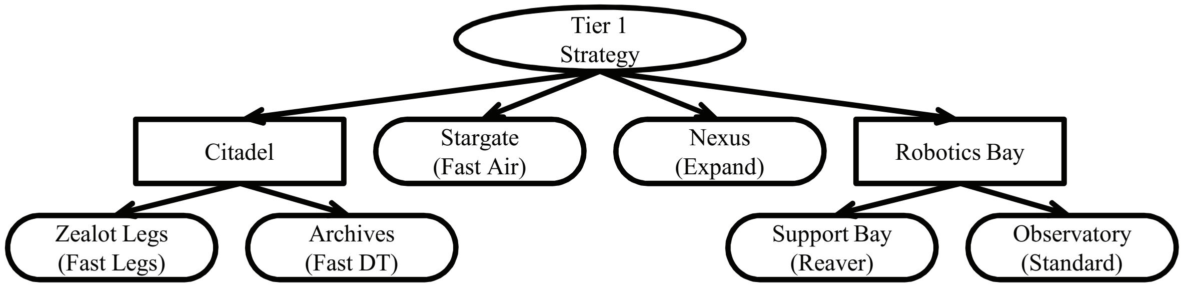

Fig. 3. Rule set for labeling Protoss strategies. The tree is traversed by selecting the first building produced in the child nodes.

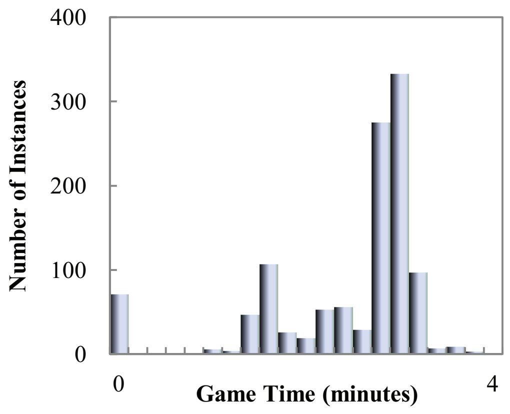

Fig. 2. Spawning Pool timing in Zerg versus Terran games

TABLE IV
STRATEGY DISTRIBUTIONS FOR PROTOSS

Strategy	Versus Protoss	Versus Terran	Versus Zerg
Fast Legs	1%	1%	10%
Fast DT	18%	16%	8%
Fast Air	1%	1%	20%
Expand	22%	31%	46%
Reaver	9%	12%	3%
Standard	27%	32%	1%
Unknown	22%	7%	12%

match ups, such as Protoss versus Terran, a wide variety of strategies are commonly used. However, strategy prediction is easier in some match ups, because a single strategy dominates. For example, the two-hatchery mutalisk strategy is used in over 70% of Zerg versus Zerg games.

IV. Evaluation

We applied several machine learning algorithms to strategy prediction and timing prediction in StarCraft. Classification algorithms were applied to the task of strategy recognition, while regression algorithms were applied to the task of predicting when specific unit types or building types will be produced. Ten-fold cross validation was performed for all of our experiments.

A. Strategy Prediction

We represent strategy prediction as a multi-class classification problem. The following algorithms were applied to classification:

- J48 – C4.5 decision tree [18]
- k-NN – Nearest neighbor [19]
- NNge – Non-nested generalized exemplars [20]
- LogitBoost – Additive logistic regression [21]

We used the implementations of these algorithms provided in the Weka [22] toolkit. The LogitBoost algorithm was configured to use 100 iterations and a shrinkage rate of 0.5. All of the other algorithms used the default settings.

In addition to the Weka algorithms, we tested two additional classifiers. The rule set classifier predicts strategies using the exact rules used to label the strategies. Since the replays are labeled based on tier 2 strategies, this classifier is not accurate until the opponent's strategy has been executed. We also implemented a state lattice classifier for comparison.

The algorithms were evaluated at different time steps throughout the game. We simulated different time steps by setting all features with a value greater than the current game time to 0. This transformation is applied to training data as well as the test data. The precision of the algorithms versus game time for strategy prediction are shown in Figure 4 and Figure 5. A comprehensive listing of the performance of the algorithms at five and ten minutes game time is shown in Table V.

The results show that different algorithms are better at different stages of the game. The instance-based algorithms (NNge and k-NN) perform well in the initial stages of the game, but degrade in the later stages of the game, while boosting performs poorly initially and improves in the later stages. All of the machine learning algorithms outperformed the state lattice classifier.

Interestingly, the machine learning algorithms had higher precision than the exact rule set during the first 8 minutes of the game. While the precision of the rule set classifier eventually reaches 100%, there is a significant difference between this classifier and the machine learning algorithms in the early stages of the game. These results indicate that the algorithms have "foresight" of the opponent's strategy. Here, we define "foresight" to be the area between the classification algorithm and the rule set. Given this metric, the machine learning algorithms clearly outperform the state lattice classifier.

TABLE V

PRECISION OF STRATEGY PREDICTION FOR GAMES AT 5 MINUTES AND 10 MINUTES.

	5 minutes				10 minutes			
	NNge	J48	Boosting	Lattice	NNge	J48	Boosting	Lattice
Protoss vs. Protoss	**0.49**	0.43	0.47	0.39	0.80	0.81	**0.86**	0.56
Protoss vs. Terran	**0.68**	0.63	0.61	0.45	0.89	0.91	**0.94**	0.62
Protoss vs. Zerg	0.63	0.63	**0.66**	0.62	0.85	0.87	**0.87**	0.44
Terran vs. Protoss	**0.76**	0.66	0.63	0.45	0.81	0.80	**0.94**	0.51
Terran vs. Terran	**0.82**	0.75	0.77	0.57	0.85	0.81	**0.92**	0.56
Terran vs. Zerg	**0.91**	0.88	0.90	0.86	**0.94**	0.90	0.86	0.60
Zerg vs. Protoss	0.53	0.56	**0.60**	0.48	0.84	0.85	**0.87**	0.49
Zerg vs. Terran	**0.53**	0.50	0.49	0.41	0.87	**0.91**	0.89	0.65
Zerg vs. Zerg	0.83	0.82	0.83	**0.84**	0.94	0.95	**0.95**	0.82
Overall	**0.69**	0.65	0.66	0.56	0.86	0.87	**0.91**	0.58

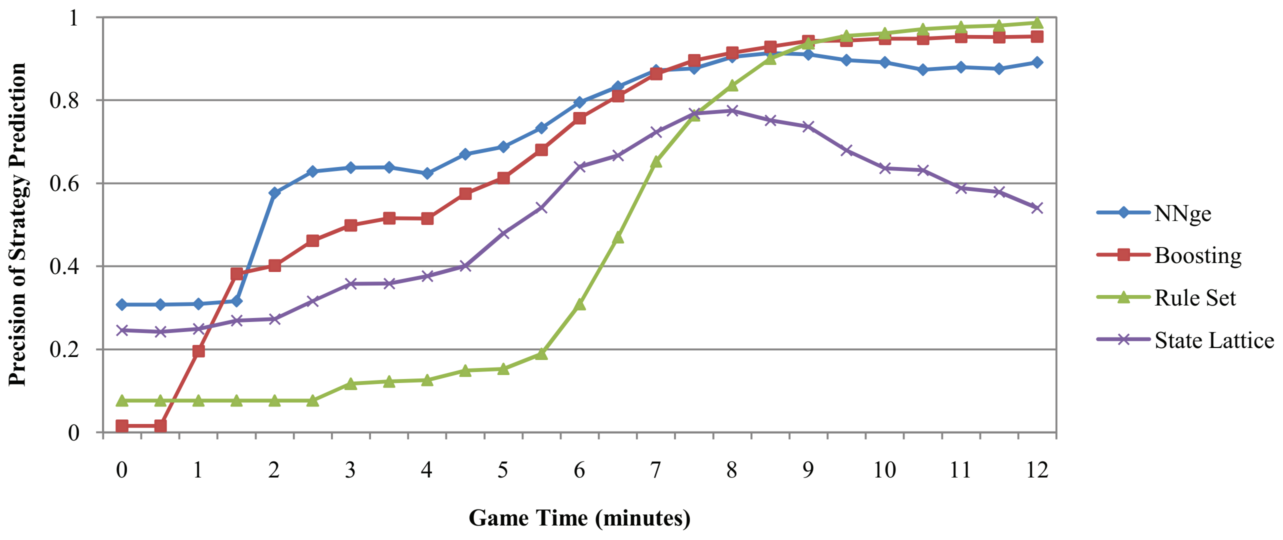

Fig. 4. Precision of strategy prediction for Protoss versus Terran

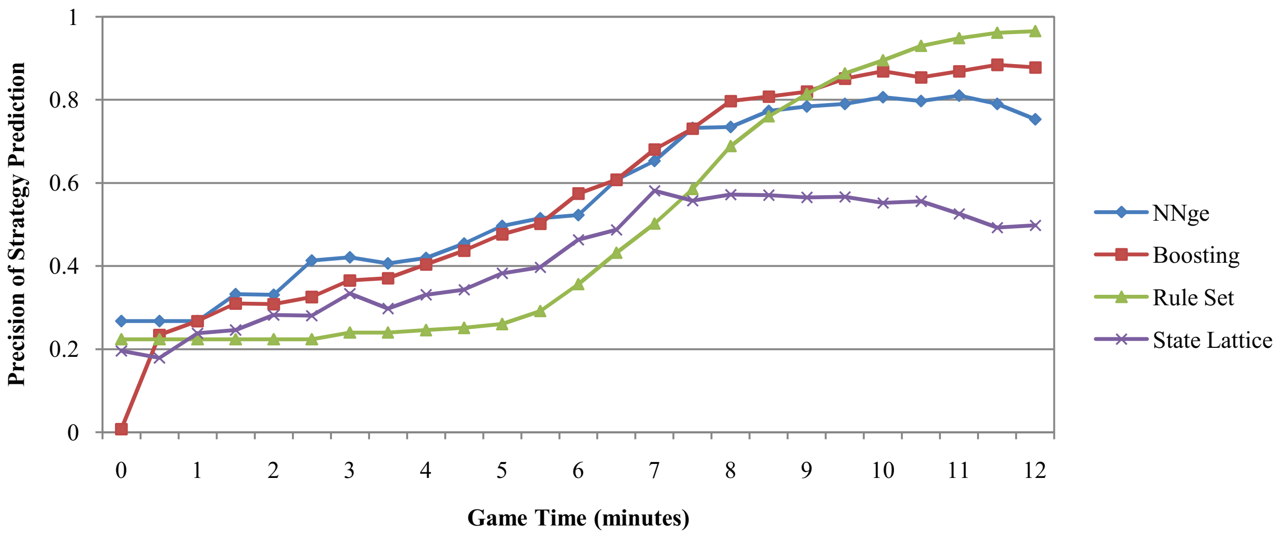

Fig. 5. Precision of strategy prediction for Protoss versus Protoss

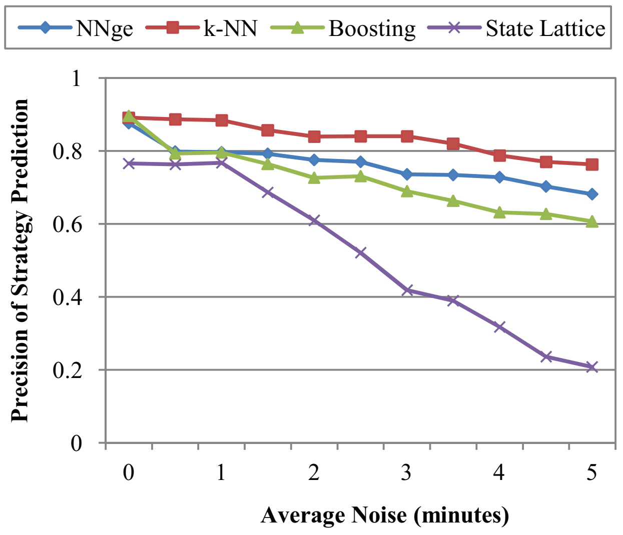

Fig. 6. Precision on noisy data. Noise is applied uniformly to individual attributes, simulating delayed opponent scouting.

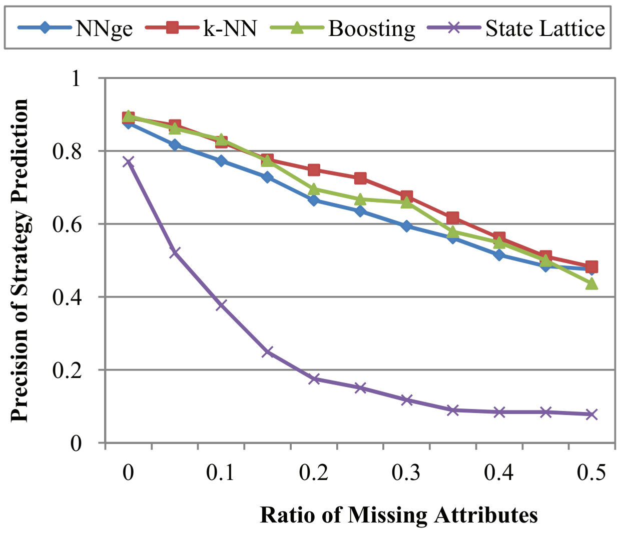

Fig. 7. Precision in an imperfect information environment. The missing attribute ratio specifies the probability that an attribute will be set to 0, simulating inability to scout an opponent.

B. Imperfect Information

Experiments were conducted to analyze the effects of noise on the classification algorithms. These experiments simulate the "fog of war" in StarCraft, which limits visibility to portions of the map in which the player controls units. Delayed scouting can be simulated by adding noise to features, while inability to scout an opponent's base can be simulated by missing features.

The first experiment added a uniform distribution of noise to individual features, which simulates delayed scouting in StarCraft. The results for this noise transformation are shown in Figure 6. All of the algorithms degrade in precision as more noise is applied to the test data set. However, the precision of the k-NN algorithm does not degrade as quickly as the other algorithms.

The second noise experiment tests the effects of imperfect information on the classification algorithms. Attributes were randomly set to 0, based on a missing attribute probability. This simulates a player that is unable to scout an opponent's base in StarCraft. The results for the missing attribute transformation are shown in Figure 7. The precision of the machine learning algorithms decreased linearly with respect to the ratio of missing attributes, while the precision of the state lattice classifier degrades to that of a random classifier after 20% of attributes are missing. The results of these two experiments indicate that the machine learning algorithms are more tolerant of noisy and missing features than the state lattice classifier.

C. Timing Prediction

We represent timing prediction as a regression problem. The following algorithms were applied to regression:

- ZeroR - Estimates the mean
- LinearRegression - Akaike linear regression [23]
- M5' – Inducing model trees [24]
- AdditiveRegression - Stochastic gradient boosting [25]

The additive regression algorithm was configured to use 100 iterations and a shrinkage rate of 0.5. All of the other algorithms used the default settings.

Regression tests were performed for individual attributes. The training and test data sets were transformed prior to training the algorithms. Given a feature f, all other features with a value greater than f's value are set to 0. This transformation sets the feature vector to the game state before the unit type, building type or unit upgrade, f, was produced.

A subset of the regression results for Zerg versus Terran games is shown in Table VI. The M5' algorithm performed the best on almost all of the features. The algorithms perform poorly on the Zergling Speed feature, which has a standard deviation of 8 minutes, because it is typically upgraded in response to an aggressive opponent rather than planned as part of a standard build. The M5' algorithm was able to predict the timing of the lair and hive structures with a mean error of less than 40 seconds. A lair enables Zerg players to build tier 2 units and buildings, while a hive enables Zerg players to build tier 3 units and buildings. Therefore, M5' can predict when the player is upgrading to the next tier of the tech tree with an average error of less than 40 seconds.

Additional regression results for Protoss versus Zerg are shown in Figure 8. Overall M5' predicted action timing with the smallest error. However, linear regression performed well on highly correlated features. For example, linear regression is able to accurately predict observer timing, because there is a direct correlation between constructing an observatory and producing an observer (see Figure 9).

D. Speed

We evaluated the speed of the strategy prediction algorithms. The results for the classification algorithms are shown in Table VII. Decision tree methods were the fastest, while instance-based algorithms were the slowest. The boosting

TABLE VI

SUBSET OF REGRESSION RESULTS FOR ZERG VERSUS TERRAN. THE VALUES (MINUTES) SHOW THE AVERAGE DIFFERENCE BETWEEN PREDICTED AND ACTUAL ACTION TIMINGS.

Action	ZeroR	Linear Regression	M5'	Additive Regression
Spawning Pool	0:28	0:17	**0:04**	0:06
Zergling	0:42	0:48	**0:25**	0:32
Zergling Speed	4:02	3:54	3:28	**2:49**
Second Hatchery	0:44	0:35	**0:19**	0:21
Hydralisk Den	2:15	1:24	**0:45**	0:52
Lair	1:02	0:55	**0:33**	0:37
Hive	4:40	0:45	**0:39**	0:57
Consume	4:55	0:32	**0:27**	0:55

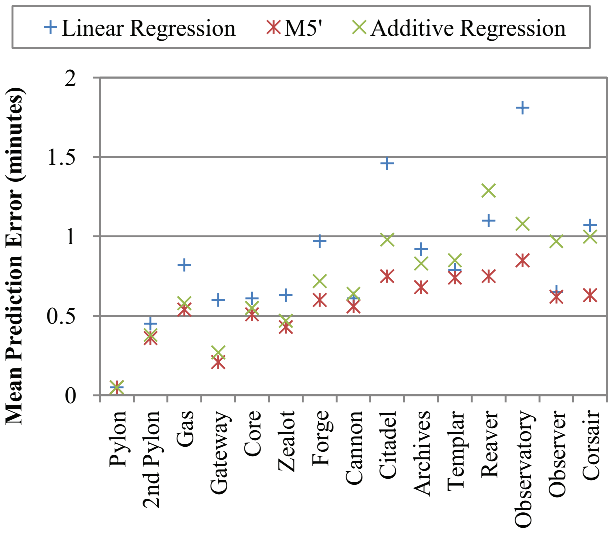

Fig. 8. Subset of regression results for Protoss versus Zerg. The points show the average difference between predicted and actual action timings.

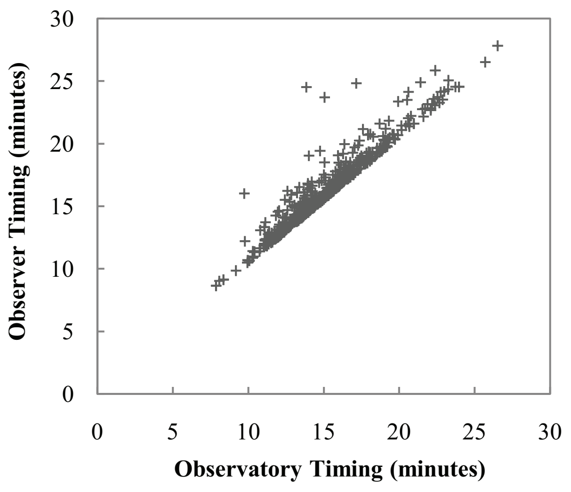

Fig. 9. Observatory and observer timing in Protoss versus Zerg games

TABLE VII

TIME TAKEN TO PERFORM ONE THOUSAND CLASSIFICATIONS

Algorithm	Time (ms)
NNge	1246
k-NN	4218
J48	6
LogitBoost	65

TABLE VIII

TIME TAKEN TO PERFORM ONE MILLION PREDICTIONS

Algorithm	Time (ms)
ZeroR	14
Linear Regression	5679
M5'	7937
Additive Regression	6352

algorithm is dependent on the number of iterations, rather than the number of examples, and was much faster than NNge and k-NN. While some of the classifiers are faster than others, all of the classifiers can be executed once per second, or every game cycle, with minimal impact on the overall system.

Results for the speed of the regression algorithms are shown in Table VIII. The speed of the linear regression algorithm is dependent on the number of features, while the speed of additive regression is based on the number of iterations. Again, all algorithms can be executed once per second with minimal impact on the system.

V. CONCLUSIONS

In this paper we have demonstrated a data mining approach to strategy prediction in real-time strategy games. We collected thousands of replay files from professional matches and applied machine learning algorithms to acquire domain knowledge and perform opponent modeling. By applying data mining to a large number of game traces, we can develop an opponent model that is not limited to a single opponent, set of maps or style of gameplay.

Machine learning was applied to the task of recognizing an opponent's strategy as well as predicting when the opponent will produce a specific unit type or building type. We presented a novel representation for encoding strategic decisions made in a game as a feature vector. Each feature vector was labeled with a specific strategy based on analysis of expert play.

Several algorithms from the Weka toolkit were applied to strategy prediction and timing prediction. The algorithms were compared against a state lattice classifier and a classifier that used the exact rule set. The machine learning algorithms using our representation outperformed the state lattice on all experiments and also had higher precision than the rule set classifier in the initial stages of a game. The machine learning algorithms show "foresight" in that they are able to predict a strategy with high confidence before it is executed. For example, the NNge algorithm was able to predict an opponent's tier 2 strategy with 70% precision five minutes into a game. For timing prediction, our results showed that M5' predicted unit and building timing with the smallest average error on most of the features. However, the results showed that regression worked well for only a subset of the features.

We also explored strategy prediction in imperfect information environments. The first experiment simulated delayed scouting by adding noise to attributes, with the instance-based algorithms being the most tolerant to noise. The second experiment simulated lack of scouting by adding missing features. The precision of the machine learning algorithms decreased linearly as the amount of information decreased. The state lattice classifier performed poorly on both experiments.

VI. Future Work

There are two main directions for future work. First, future work could focus on improving the precision of strategy and timing prediction. Domain knowledge could be incorporated into classification and regression algorithms in the form of situation assessment [15] or domain specific metrics [17]. However, the main focus should be on regression, because regression is currently accurate for only a subset of the features. Regression could be improved by incorporating the game mechanics into the predictions. For example, a regression algorithm could factor in the construction times of buildings.

Another area for future work is incorporating our results from data mining into a full game playing agent. A candidate framework for this research is the integrated agent architecture of McCoy and Mateas [26]. The agent uses a reactive planner to play full games of Wargus. Currently, all of the behaviors are hard-coded by a developer. New predicates could be introduced into the planner which specify that the opponent is pursuing a specific strategy. These predicates could be used as preconditions to select among behaviors. In the current agent, the rules for deciding to progress to tier 2 are hard coded. Our results for predicting when the opponent is going to change tiers could be used as preconditions for this behavior.

References

[1] S. Bakkes, P. Spronck, and J. van den Herik, "Rapid Adaptation of Video Game AI," in *Proceedings of the IEEE Symposium on Computational Intelligence in Games*. Perth, Australia: IEEE, 2008, pp. 79–86.

[2] S. Hladky and V. Bulitko, "An Evaluation of Models for Predicting Opponent Positions in First-Person Shooter Video Games," in *Proceedings of the IEEE Symposium on Computational Intelligence in Games*. Perth, Australia: IEEE, 2008, pp. 39–46.

[3] A. Fink, J. Denzinger, and J. Aycock, "Extracting NPC behavior from computer games using computer vision and machine learning techniques," in *IEEE Symposium on Computational Intelligence and Games*. Honolulu, Hawaii: IEEE, 2007, pp. 24–31.

[4] M. Fagan and P. Cunningham, "Case-based plan recognition in computer games," *Lecture notes in computer science*, vol. 2689, pp. 161–170, 2003.

[5] E. Charniak and R. P. Goldman, "A bayesian model of plan recognition," *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, 1993.

[6] D. Albrecht, I. Zukerman, and A. Nicholson, "Bayesian Models for Keyhole Plan Recognition in an Adventure Game," *User Modeling and User-Adapted Interaction*, vol. 8, no. 1, pp. 5–47, 1998.

[7] S. Carberry, "Techniques for plan recognition," *User Modeling and User-Adapted Interaction*, vol. 11, no. 1-2, pp. 31–48, 2001.

[8] D. Cheng and R. Thawonmas, "Case-based plan recognition for real-time strategy games," in *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*. Reading, UK: University of Wolverhampton, 2004, pp. 36–40.

[9] M. Molineaux, D. W. Aha, and P. Moore, "Learning Continuous Action Models in a Real-Time Strategy Environment," in *Proceedings of the Florida Artificial Intelligence Research Conference*. Coconut Grove, Florida: AAAI Press, 2008, pp. 257–262.

[10] R.-K. Balla and A. Fern, "UCT for Tactical Assault Planning in Real-Time Strategy Games," in *Proceedings of the International Joint Conference on Artificial Intelligence*. Pasadena, California: Morgan Kaufmann, 2009, pp. 40–45.

[11] M. J. V. Ponsen, H. Muñoz-Avila, P. Spronck, and D. W. Aha, "Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1535–1540.

[12] D. Aha, M. Molineaux, and M. Ponsen, "Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game," *Lecture notes in computer science*, vol. 3620, pp. 5–20, 2005.

[13] M. Ponsen, "Improving adaptive game AI with evolutionary learning," Master's thesis, Delft University of Technology, Delft, the Netherlands, 2004.

[14] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Learning from Demonstration and Case-Based Planning for Real-Time Strategy Games," *Soft Computing Applications in Industry*, pp. 293–310, 2008.

[15] K. Mishra, S. Ontañón, and A. Ram, "Situation assessment for plan retrieval in real-time strategy games," in *Proceedings of the European Conference on Case-Based Reasoning*. Trier, Germany: Springer, 2008, pp. 355–369.

[16] J. Hsieh and C. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *Proceedings of the International Joint Conference on Neural Networks*. Hong Kong, China: IEEE, 2008, pp. 3106–3111.

[17] B. G. Weber and M. Mateas, "Conceptual Neighborhoods for Retrieval in Case-Based Reasoning," in *Proceedings of the International Conference on Case-Based Reasoning*. Seattle, Washington: Springer, 2009, pp. 343–357.

[18] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, California: Morgan Kaufmann, 1993.

[19] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[20] B. Martin, "Instance-based learning: nearest neighbour with generalisation," Master's thesis, University of Waikato, Hamilton, New Zealand, 1995.

[21] J. Friedman, T. Hastie, and R. Tibshirani, "Additive Logistic Regression: A Statistical View of Boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–374, 2000.

[22] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. San Francisco, California: Morgan Kaufmann, 2005.

[23] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[24] Y. Wang and I. Witten, "Inducing model trees for continuous classes," in *Poster Papers of the European Conference on Machine Learning*, Prague, Czech Republic, 1997, pp. 128–137.

[25] J. Friedman, "Stochastic gradient boosting," *Computational Statistics and Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[26] J. McCoy and M. Mateas, "An Integrated Agent for Playing Real-Time Strategy Games," in *Proceedings of the AAAI Conference on Artificial Intelligence*. Chicago, Illinois: AAAI Press, 2008, pp. 1313–1318.