Predicting the Outcome of Small Battles in StarCraft *

Antonio A. Sánchez-Ruiz

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid (Spain)
antsanch@fdi.ucm.es

Abstract. Real-Time Strategy (RTS) games are popular testbeds for AI researchers. In this paper we compare different machine learning algorithms to predict the outcome of small battles of marines in StarCraft, a popular RTS game. The predictions are made from the perspective of an external observer of the game and they are based only on the actions that the different units perform in the battlefield. Our empirical results show that case-based approaches based on k-Nearest Neighbor classification outperform other standard classification algorithms like Linear and Quadratic Discriminant Analysis or Support Vector Machines.

Keywords: Prediction, StarCraft, Linear and Quadratic Discriminant Analysis, Support Vector Machines, k-Nearest Neighbors

1 Introduction

Real-Time Strategy (RTS) games are popular testbeds for AI researchers [4] because they provide complex and controlled environments in which to carry out different experiments. In this paper we assume the role of an external observer of the game that tries to predict the outcome when the armies of two different players engage in combat. As a spectator of the game, we can only base the predictions on the actions of the different units in the battlefield. From this perspective, we can consider each army as a group of agents working in a coordinated manner to defeat the other army. We know that the units in the game are not really agents because they are not autonomous (in fact they are controlled by a human player or by the internal AI of the game), but from the perspective of an external observer we only see several units performing actions in a simulation, and we do not know whether those actions are consequence of individual decisions or some superior intelligence. Therefore, our approach to prediction in RTS games could be applied as well to multi-agent simulations.

The ability to predict the outcome of battles is interesting because it can be used to dynamically modify the strategy of the player. For example, the player could decide to change the composition of the army, to bring more troops into

* Supported by Spanish Ministry of Economy and Competitiveness under grant TIN2014-55006-R

Fig. 1: Screenshot of our custom map: Battle of Marines

the battlefield or deploy them differently, or even to flee if the prospects are not good. In general, an agent able to make predictions (running an internal simulation based on what he knows) might be able to adapt his behavior more successfully than other agent without this ability.

In this work, we compare classical classification algorithms like Linear and Quadratic Discriminant Analysis, Support Vector Machines, and two instance-based classifiers based on the retrieval of the k-Nearest Neighbors (kNN). kNN classifiers can be seen as simple Case-based Reasoning (CBR) systems that only implement the retrieval phase of the CBR cycle. In this paper we study the accuracy of the prediction during the course of the battle, the number of games that each algorithm needs to learn, and the stability of the prediction over time.

The rest of the paper is organized as follows. Sections 2 and 3 describe the scenario used in the experiments, the process to extract the data for the analysis and the features chosen to represent the game state. Sections 4 and 5 explain the different classification algorithms and the results obtained. The paper concludes with the related work, and some conclusions and directions for future research.

2 Battles of Marines in StarCraft

StarCraft[1] is a popular RTS game in which players have to harvest resources, develop technology and build armies combining different types of units to defeat

[1] http://us.blizzard.com/en-us/games/sc/

game	frame	units1	life1	area1	units2	life2	area2	distance	winner
1	0	6	40	2772	6	40	2520	1309.903	1
1	3	6	40	2772	6	40	2520	1309.903	1
1	6	6	40	2736	6	40	2925	1302.857	1
1	9	6	40	2964	6	40	2923	1282.317	1
1	12	6	40	3876	6	40	2905	1266.277	1
1	15	6	40	4332	6	40	3045	1246.241	1

Table 1: Examples of game states extracted from a Starcraft game trace.

the other players. The combination of different types of units and abilities, and the dynamic nature of the game force players to develop strategies at different levels. At the *macro* level, players have to decide the amount of resources invested in map exploration, harvesting, technology development, troops, offensive and defensive forces, among others. At the *micro* level players have to combine different types of units, locate them in the map and use their abilities. In this paper we focus on small battles, that is, at the micro level.

StarCraft also provides a map editor to create custom games. Using this tool, we have created a simple combat scenario (Figure 1) in which each player controls a small army of 6 *terran marines* (marines are basic ground combat units with ranged attack). The game always begins with the same initial configuration, each army located on opposite sides of the map, and the game ends when all the units of one player are destroyed. In this type of scenario it is very important to strategically locate the units on the map to take advantage of the range attack and concentrate the fire on a few units to destroy them as soon as possible.

3 Data Collection and Feature Selection

In order to obtain the data to train the different classifiers, we played 200 games collecting traces that describe the evolution of the games. We configured the map so the internal game AI controls both players so (1) we can automatically play as many games as required, and (2) we know that all the games are well balanced (since the StarCraft AI is playing against itself). Finally, there is a third player that only observes the game (it does not intervene) and extracts the game traces to a file so they can be analyzed later[2].

The data set contains *traces* of 200 games in which player 1 won 119 times and player 2 the remaining 81. They are very fast games with an average duration of 19.12 seconds. In each trace we store the *game state* 6 times per second, so each game is described with approximately 114 games states or *samples*.

Each game state is stored using a vector of features (Table 1) that represents the combat power of each army and the strategic deployment of the troops in the map. The combat power is represented using the number of units alive in each

[2] We use the BWAPI framework to extract information during the game (https://github.com/bwapi/bwapi).

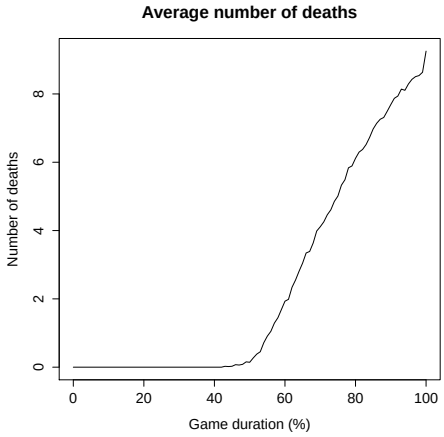

Fig. 2: Average number of deaths during the game.

army and their average life. To represent the strategic distribution of troops in the map we compute the area of the minimum axis aligned rectangle containing the units of each player and the distance between the centers of both rectangles. The rectangle area is a measure of the dispersion of the units, and the distance between the centers indicates how close the two armies are. Each game state is labeled later with the winner of that particular game. The table also shows the game and the current frame for clarity (1 second is 18 game frames although we only sample 6 of them), but we do not use those values in the prediction.

The features to describe the strategic distribution of the troops in the map are especially important during the first seconds of the game. Figure 2 shows the average number of dead units during the 200 games. As we can see, during the first half of the game the armies are approaching each other and the fight does not start until the second half. Thus, during the first seconds of the game the predictions will depend only on the relative location of the units.

4 Classification algorithms

We will compare the following classification algorithms in the experiments:

- Linear Discriminant Analysis (LDA) [8] is classical classification algorithm that uses a linear combination of features to separate the classes. It assumes that the observations within each class are drawn from a Gaussian distribution with a class specific mean vector and a covariance matrix common to all the classes.
- Quadratic Discriminant Analysis (QDA) [9] is quite similar to LDA but it does not assume that the covariance matrix of each of the classes is identical, resulting in a more flexible classifier.

Classifier	Accuracy	Parameters
Base	0.5839	
LDA	0.7297	
QDA	0.7334	
SVM	0.7627	kernel = radial, C = 1, sigma = 0.3089201
KNN	0.8430	k = 5
KKNN	0.9570	kernel = optimal, kmax = 9, distance = 2

Table 2: Classification algorithms, configuration parameters and global accuracy.

- Support Vector Machines (SVM) [7] have grown in popularity since they were developed in the 1990s and they are often considered one of the best *out-of-the-box* classifiers. SVM can efficiently perform non-linear classification using different kernels that implicitly map their inputs into high-dimensional feature spaces. In our experiments we tested 3 different kernels (lineal, polynomial and radial basis) obtaining the best results with the radial basis.
- k-Nearest Neighbour (kNN) [2] is a type of instance-based learning, or lazy learning, where the function to learn is only approximated locally and all computation is deferred until classification. The kNN algorithm is among the simplest of all machine learning algorithms and yet it has shown good results in several different problems. The classification of a sample is performed by looking for the k nearest (in Euclidean distance) training samples and deciding by majority vote.
- Weighted K-Nearest Neighbor (kkNN) [10] is a generalization of kNN that retrieves the nearest training samples according to Minkowski distance and then classifies the new sample based on the maximum of summed kernel densities. Different kernels can be used to weight the neighbors according to their distances (for example, the *rectangular* kernel corresponds to standard un-weighted kNN). We obtained the best results using the *optimal* kernel [14] that uses the asymptotically optimal non-negative weights under some assumptions about the underlying distributions of each class.

The three first algorithms use the training data (labeled game states in our experiments) to infer a generalized model, and then they use that model to classify the test data (new unseen game states). The last two algorithms, however, use the training data as cases and the classification is made based on the most similar stored cases. All the experiments have been run using the R statistical software system [11] and the algorithms implemented in the packages *caret*, *MASS*, *e1071*, *class* and *kknn*.

5 Analyzing the results

Table 2 shows the configuration parameters used in each classifier and its accuracy computed as the ratio of samples (game states) correctly classified. The optimal parameters for each classifier were selected using repeated 10-fold cross

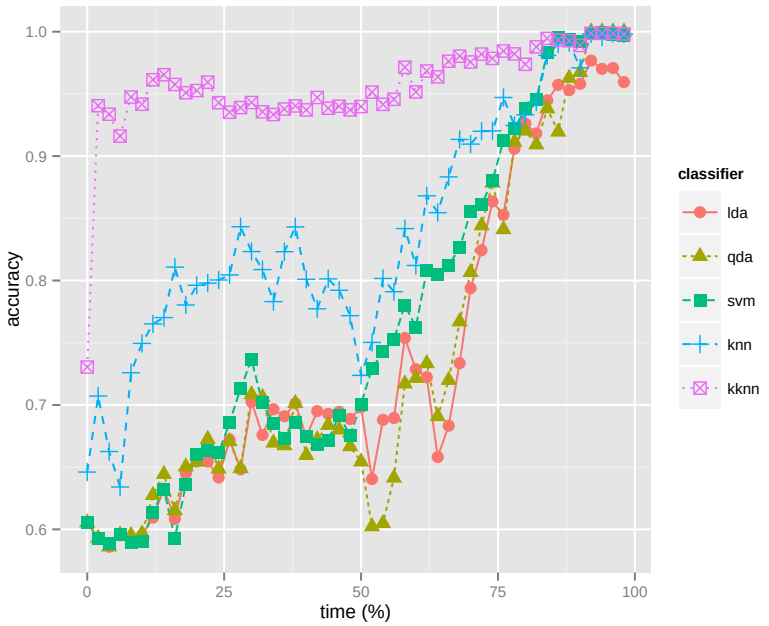

Fig. 3: Accuracy of the classifiers during the game.

validation over a wide set of different configurations. The accuracy value has been calculated as the average of 32 executions using 80% of the samples as the training set and the remaining 20% as the test set.

The *base* classifier predicts the winner based only on the proportion of samples belonging to each class (58.39% of the samples correspond to games won by player 1) and it is useful only as a baseline to compare the other classifiers. LDA, QDA and SVM obtain accuracy values ranging from 72% to 76%. The two instance-based algorithms, on the other hand, obtain higher precision values. It is especially impressive the result of kkNN that is able to predict the winner 95.70% of the times. These results seem to indicate that, in this particular scenario, it is quite difficult to obtain a generalized model, and local based methods perform much better.

The global accuracy value may not be informative enough because it does not discriminate the time of the game when the prediction is made. It is reasonable to expect the accuracy of the predictions to increase as the game evolves as it is shown in Figure 3. The x-axis represents the percentage of elapsed time (so we can uniformly represent games with different duration) and the y-axis the average accuracy of each classifier for game states from that time interval.

Selecting a winner during the second half of the game is relatively easy since we can see how the battle is progressing, but during the first half of the game the prediction problem is much more difficult and interesting since we only see

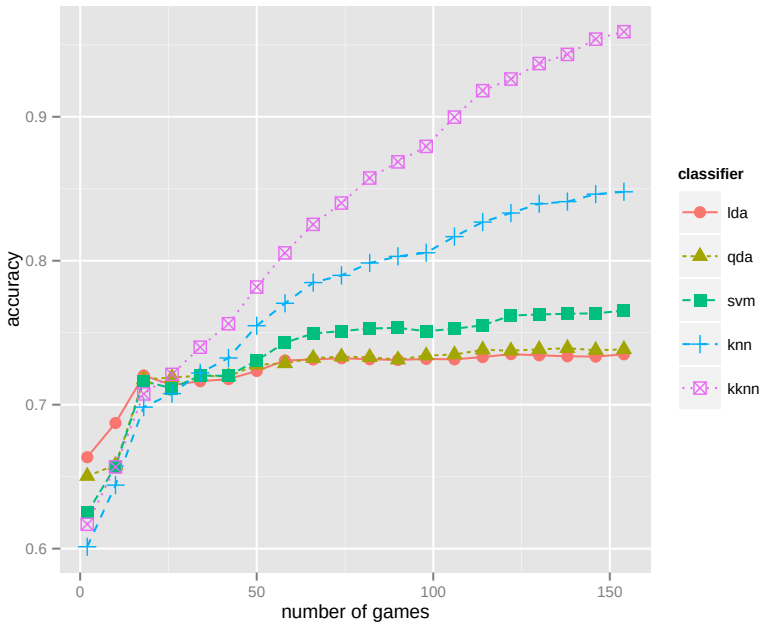

Fig. 4: Accuracy of the classifiers vs. the number of training games.

the formation of the armies (*pre-battle* vs. *during battle* prediction). LDA, QDA and SVM do not reach 90% of accuracy until the last quarter of the game. kNN is able to reach the same accuracy at 66% of the game. The results of kkNN are spectacular again, classifying correctly 90% of the game states from the first seconds. kkNN is the only algorithm able to effectively find useful patters in the training data before the armies begin to fight. Our intuition is that the training data is biased somehow, probably because the StarCraft AI is playing against itself and it does not use so many different attack strategies. In any case, kkNN seems to be the only algorithm to effectively predict the outcome of the battle from the first moves of each army.

Another important aspect when choosing a classifier is the volume of training data they need to perform well. Figure 4 shows the accuracy of each classifier as we increase the number of games used during the training phase. In the first 20 games all the algorithms perform similarly but then only kNN and kkNN keep improving fast. It makes sense for instance-based algorithms to require a large number of samples to achieve their highest degree of accuracy in complex domains, while algorithms that infer general models stabilize earlier but their prediction is more biased.

Finally, Figure 5 shows the stability of the predictions. We divided the game in 20 intervals of 5% of time. The y-axis represents the number of games for which the classifier made a prediction in that time interval that remained stable

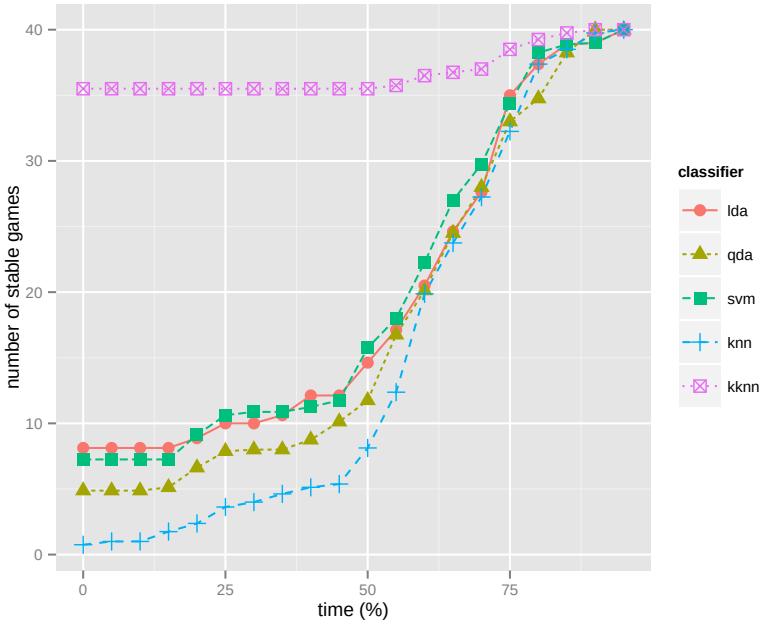

Fig. 5: Number of games for which each classifier becomes stable at a given time.

for the rest of the game. For example, the y value for $x = 0$ represents the number of games in which the prediction became stable during the first time interval (0-4.99% of the game). Most of the classifiers need to wait until the last quarter of the game to be stable in 80% of the games, except kkNN that is very stable from the beginning. There are a few games, however, in which all the classifiers are wrong until the end of the game because the army that was winning made bad decisions during the last seconds.

In conclusion, instance-based classifiers seems to perform better in our scenario, and kkNN in particular is the only algorithm that is able to effectively find useful patters in the training data before the armies begin to fight. It is also the most stable and it only performs worst than the other algorithms where there is a very small number of training games available.

6 Related work

RTS games have captured the attention of AI researchers as testbeds because they represent complex adversarial systems that can be divided into many interesting subproblems [4]. Proof of this are the different international competitions in AIIDE and CIG conferences. We recommend [12] for a complete overview of the existing work on this domain, the specific AI challenges and the solutions that have been explored so far.

There are several papers regarding the combat aspect of RTS games. [6] describes a fast Alpha-Beta search method that can defeat commonly used AI scripts in small combat scenarios. It also presents evidence that commonly used combat scripts are highly exploitable. A later paper [5] proposes new strategies to deal with large StarCraft combat scenarios.

Several different approaches have been used to model opponents in RTS games in order to predict the strategy of the opponents and then be able to respond accordingly: decision trees, kNN, logistic regression [17], case-based reasoning [1, 3], bayesian models [16] and evolutionary learning [13] among others.

A paper very related to our work is [15], where authors present a Bayesian model that can be used to predict the outcome of isolated battles, as well as to predict what units are needed to defeat a given army. Our approach is different in the sense that we try to predict the outcome as the game progresses and our battles begin with 2 balanced armies (same number and type of units). We use tactical information regarding the location of the troops and we use StarCraft to run the experiments and not a battle simulator.

7 Conclusions and Future work

In this paper we compare different machine learning algorithms in order to predict the outcome when two small marine armies engage in combat in the StarCraft game. The predictions are made from the perspective of an external game observer so they are based only on the actions of the individual units. The proposed approach is not limited to RTS games and can be used in other domains like multi-agent simulations, since it does not depend on whether the actions are decided by each unit autonomously or by a global manager. Our results indicate that, in this scenario, instance-based classifiers such as kNN and kkNN behave much better than other classifiers that try to infer a general domain model in terms of accuracy, size of the training set and stability.

There are several possible ways to extend our work. We have only considered a small battle scenario with a limited number of units. As the number and diversity of units increases, the number of possible combat strategies also grows creating more challenging problems. Our map is also quite simple and flat, while most of the StarCraft maps have obstacles, narrow corridors, wide open areas and different heights providing locations with different tactical value. The high accuracy values obtained by kkNN from the first seconds of the battle make us suspicious about the diversity of the strategies in the recorded games. We plan to run new experiments using human players to verify our results. Finally, our predictions are based on static pictures of the current game state. It is reasonable to think that we could improve the accuracy if we consider the evolution of the game and not just the current state to predict the outcome of the battle.

References

1. Aha, D.W., Molineaux, M., Ponsen, M.: Learning to win: Case-based plan selection in a real-time strategy game. In: in Proceedings of the Sixth International

Conference on Case-Based Reasoning. pp. 5–20. Springer (2005)
2. Altman, N.S.: An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. American Statistician 46, 175–185 (1992)
3. Auslander, B., Lee-Urban, S., Hogg, C., Muñoz-Avila, H.: Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. In: Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings. pp. 59–73 (2008)
4. Buro, M., Furtak, T.M.: RTS games and real-time AI research. In: In Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS. pp. 51–58 (2004)
5. Churchill, D., Buro, M.: Portfolio greedy search and simulation for large-scale combat in starcraft. In: 2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013. pp. 1–8 (2013)
6. Churchill, D., Saffidine, A., Buro, M.: Fast Heuristic Search for RTS Game Combat Scenarios. In: Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, October 8-12, 2012 (2012)
7. Cortes, C., Vapnik, V.: Support-Vector Networks. Mach. Learn. 20(3), 273–297 (Sep 1995)
8. Fisher, R.A.: The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics 7(7), 179–188 (1936)
9. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, Springer New York Inc., New York, NY, USA (2001)
10. Hechenbichler, K., Schliep, K.: Weighted k-Nearest-Neighbor Techniques and Ordinal Classification (2004)
11. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning: with Applications in R. Springer Texts in Statistics, Springer (2013)
12. Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft. IEEE Trans. Comput. Intellig. and AI in Games 5(4), 293–311 (2013)
13. Ponsen, M.J.V., Muñoz-Avila, H., Spronck, P., Aha, D.W.: Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning. In: Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA. pp. 1535–1540 (2005)
14. Samworth, R.J.: Optimal weighted nearest neighbour classifiers. Ann. Statist. 40(5), 2733–2763 (10 2012)
15. Stanescu, M., Hernandez, S.P., Erickson, G., Greiner, R., Buro, M.: Predicting Army Combat Outcomes in StarCraft. In: Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, Boston, Massachusetts, USA, October 14-18, 2013 (2013)
16. Synnaeve, G., Bessière, P.: A Bayesian Model for Plan Recognition in RTS Games Applied to StarCraft. In: Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011, October 10-14, 2011, Stanford, California, USA (2011)
17. Weber, B.G., Mateas, M.: A Data Mining Approach to Strategy Prediction. In: Proceedings of the 5th International Conference on Computational Intelligence and Games. pp. 140–147. CIG'09, IEEE Press, Piscataway, NJ, USA (2009)