Predicting the Winner in Two Player StarCraft Games *

Antonio A. Sánchez-Ruiz

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

Abstract. In this paper we compare different machine learning algorithms to predict the outcome of 2 player games in StarCraft, a well-known Real-Time Strategy (RTS) game. In particular we discuss the game state representation, the accuracy of the prediction as the game progresses, the size of the training set and the stability of the predictions.

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

1 Introduction

Real-Time Strategy (RTS) games are very popular testbeds for AI researchers because they provide complex and controlled environments on which to test different AI techniques. Such games require the players to make decisions on many levels. At the *macro* level, the players have to decide how to invest their resources and how to use their units: they could promote resource gathering, map exploration and the creation of new bases in the map; or they could focus on building defensive structures to protect the bases and training offensive units to attack the opponents; or they could invest in technology development in order to create more powerful units in the future. At the *micro* level, players must decide how to divide the troops in small groups, where to place them in the map, what skills to use and when, among others. And all these decision have to be reevaluated every few minutes because RTS games are very dynamic environments due to the decisions made by the other players.

Most of the literature related to AI and StarCraft focuses on the creation of bots that use different strategies to solve these problems. There are even international competitions in which several bots play against each other testing different AI techniques [5, 4, 3]. In this paper we use a different approach, our bot does not play but acts as an external observer of the game. Our goal is to be able to predict the winner of the game with certain level of trust based on the events occurring during the game. In order to do it, we have collected data

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

from 100 different 2 player games, and we have used them to train and compare different learning algorithms: Linear and Quadratic Discriminant Analysis, Support Vector Machines, k-Nearest Neighbors.

The rest of this paper is organized as follows. Next section describes StarCraft, the RTS game that we use in our experiments. Section 3 explains the process to extract the data for the analysis and the features chosen to represent the game state. Section 4 describes the different data mining classifiers that we use to predict the winner. Next, Section 5 analyzes the predictions produced by the different classifiers and the accuracy that we are able to reach. The paper concludes with a discussion of the related work, conclusions and some directions for future work.

2 StarCraft

StarCraft[1] is a popular Real-Time Strategy game in which players have to harvest resources, develop technology, build armies combining different types of units and defeat the opponents. Players can choose among 3 different races, each one with their own types of units, strengths and weaknesses. The combination of different types of units and the dynamic nature of the game force players to adapt their strategies constantly, creating a really addictive and complex environment. Because of this, StarCraft has become a popular testbed for AI researchers that can create their own bot using the BWAPI[2] framework.

In this paper we will focus on just one of the three available races: the *Terrans* that represent the human race in this particular universe. At the beginning of the game (see Figure 1), each player controls only one building, the command center, and a few collecting units. As the game progresses, each player has to collect resources, build new buildings to develop technology and train stronger troops in order to build an army and defeat the opponents. Figure 2 shows the same game after one hour of play, and now both players control several different units. In fact, the mini-map in the bottom left corner of the screen reveals the location of both armies (blue and red dots), and the game seems balanced because each player controls about half of the map[3].

3 Data Collection and Feature Selection

In order to collect data to train the different classifiers we need to play several games. Although StarCraft forces the existence of at least one *human*[4] player in the game, we have found a way to make the internal AI that comes implemented

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

[2] http://bwapi.github.io/

[3] In this example we have removed the fog-of-war that usually hides the parts of the map that are not visible for the current player.

[4] Note that *human* players are actually the ones controlled by bots using BWAPI while *computer* players are controlled by the game AI.

Fig. 1: StarCarft: first seconds of the game.

Fig. 2: StarCraft: state of the game after 1 hour playing.

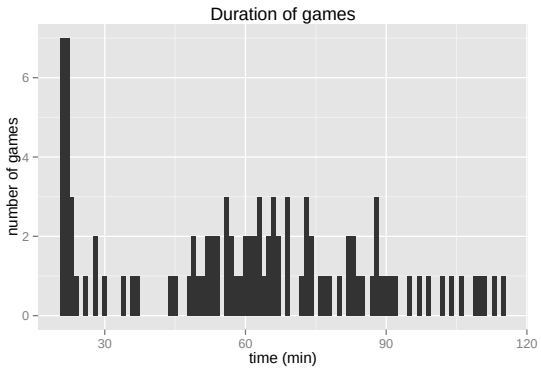

Fig. 3: Duration of the games in minutes

in StarCraft to play against itself. This way we are able to play as many games as we need automatically, and we are sure the game is well balanced since both players are controlled by the same AI.

It is possible to modify the predefined maps included in StarCraft to make the internal game AI to play against itself using a map editor tool provided with the game. In our experiments we have modified the 2 players map *Baby Steps*, so that StarCraft controls the first 2 players and there is an extra third human player. There are different AI scripts available depending on the desired level of aggressiveness, we have used *Expansion Terran Campaign Insane*. The human player has no units, will be controlled by our BWAPI bot and has full vision of the map. Finally, we disable the normal triggers that control the end of the game so we can restart the game from our bot when one the first 2 players wins. This last step is important because the normal triggers would end the game as soon as it starts because the third player has no units. Therefore, our bot cannot interfere in the development of the game but can extract any information we require.

We have created a dataset containing *traces* of 100 games in which each player won 50% of the times. Figure 3 shows the duration in minutes of the games. There are a few fast games in which one of the players was able to build a small army and defeat the other player quickly, but most games last between 45 and 100 minutes. The average duration of the games is 60.83 minutes.

Figure 4 shows the evolution of resources and units of one player computed as the average values of 100 games. The x-axis represents time as a percentage of the game duration so we can uniformly represent games with different duration, and the y-axis the number of resources (left image), buildings and troops (right image). Regarding resources, we see that during the first quarter of the game the player focus on gathering resources that will be expended during the second quarter, probably building an army and developing technology. During the second half of the game resources do not change so much, probably because

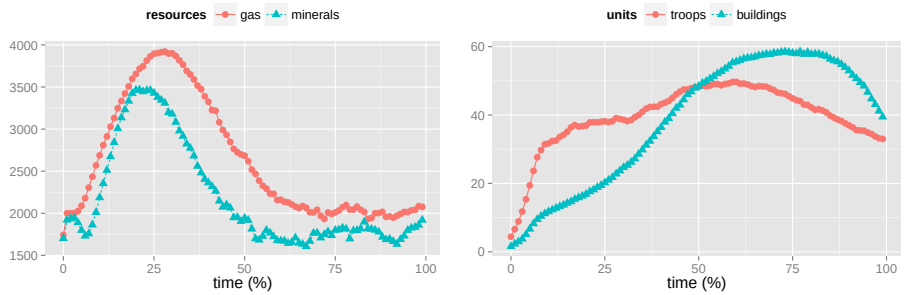

Fig. 4: Available resources, buildings and troops as the game progresses.

game	frame	gas1	minerals1	scv1	marine1	[...]	gas2	minerals2	scv2	marine2	[...]	winner
1	9360	2936	2491	18	23	...	2984	2259	20	26	...	1
1	9450	2952	2531	18	20	...	3000	2315	20	20	...	1
1	9540	2968	2571	18	14	...	3024	2371	20	14	...	1
1	9630	2892	2435	18	12	...	2940	2219	20	7	...	1

Table 1: Features selected to represent each game state (traces). We store the game and current time, the strength of each player (resources, troops and buildings) and the winner.

there are not so many resources left in the map and the player has to invest them more carefully. Regarding troops and buildings, the initial strategy is to build an army as fast as possible, while the construction of buildings seems more lineal. During the second half of the game there are more buildings than troops in the map, but we need to take into account that some of those buildings are defensive structures like anti-air turrets or bunkers that also play a role in combat. The final fall in the number of troops and buildings correspond to the last attacks, in which half of the times the player is defeated.

During the games we collect *traces* representing the state of the game at a given time. Each trace is represented using a vector of features labeled with the winner of the game (see Table 1). We try to capture the strength of each player using the available resources and the number of units of each particular type controlled at the current time. The table also shows the game and the current frame (1 second are 18 game frames) for clarity, but we do not use these values to predict the winner. We extract one trace every 5 seconds collecting an average of 730 traces per game.

There are 2 different types of resources (minerals and gas), 15 different types of troops and 11 different types of buildings only in the *Terran* race. So we need a vector of 28 features to represent each player in the current state. We could have decided to represent the strength of each player using an aggregation function instead of using this high dimensional representation, but since this is

a strategy game we hope to be able to automatically learn which combination of units is more effective.

4 Classification algorithms

We will use the following classification algorithms in the experiments:

- Linear Discriminant Analysis (LDA) [10] 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) [11] 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.
- Support Vector Machines (SVM) [9] 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 polynomial.
- 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) [12] 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 [17] that uses the asymptotically optimal non-negative weights under some assumptions about the underlying distributions of each class.

All the experiments in this paper have been run using the R statistical software system[13] and the algorithms implemented in the packages *caret*, *MASS*, *e1071*, *class* and *kknn*.

5 Experimental results

Table 2 shows the configuration parameters used in each classifier. The values in the table for each classifier were selected using repeated 10-fold cross validation

Classifier	Accuracy	Parameters
Base	0.5228	
LDA	0.6957	
QDA	0.7164	
SVM	0.6950	kernel = polynomial, degree = 3, scale = 0.1, C = 1
KNN	0.6906	k = 5
KKNN	0.6908	kernel = optimal, kmax = 9, distance = 2

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

over a wide set of different configurations. The overall accuracy value represents the ratio of traces correctly classified, and it has been computed as the average accuracy value of 16 executions using 80% of the traces as the training set and the remaining 20% as the test set.

One open problem in classification is to be able to characterize the domain to decide in advance which learning algorithm will perform better. We usually do not know which algorithm to choose until we have run the experiments. In our experiments all of them seem to perform very similar. The *base* classifier predicts the winner according to the number of traces in the dataset won by each player (i.e. ignores the current state to make the prediction) and it is included in the table only as a baseline to compare the other classifiers. The best results are for QDA that reaches a level of accuracy of 71%. 71% might not seem to be very high but we have to take into account that the games are very balanced because the same AI controls both players and the distribution of resources in the map is symmetrical for both players. Besides, in this experiment we are using all the traces in the dataset, so we are trying to predict the winner even during the first minutes of each game.

Figure 5 shows some more interesting results, the average accuracy of the different classifiers as the game progresses. RTS games are very dynamic environments and just one bad strategic decision can tip the balance towards one of the players. How long do we have to wait to make a prediction with some level of trust? For example, using LDA or QDA we only have to wait until a little over half of the game to make a prediction with a level of accuracy over 80%. It is also interesting that during the first half of the game the classifiers based on lazy algorithms like KNN and KKNN perform better, and other algorithms like LDA and QDA obtain better results during the second half. All the classifiers experience a great improvement in terms of accuracy when we get close to the middle of the game. We think that at this point of the game both players have already invested most of their resources according to their strategy (promoting some type of units over others, locating the defensive buildings in the bases...) so it is easier to predict the outcome of the game. When the games reaches the 90% of their duration, all classifiers obtain a level of accuracy close to 100% but that is not surprising because at this point of the game one the players has already lost an important part of his army.

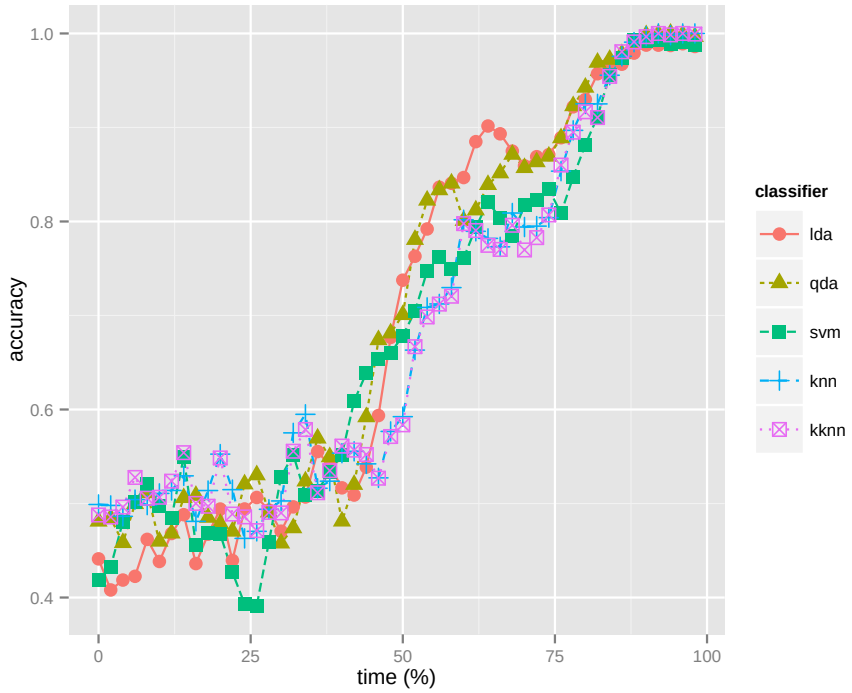

Fig. 5: Accuracy of classifiers as the games progress.

Another important aspect when choosing a classifier is the number of samples you need during the training phase in order to reach a good level of accuracy. Figure 6 shows the level of accuracy of each classifier as we increase the number of games used for training. Lazy approaches like KNN and KKNN seem to work better when we use less than 25 games for training, and LDA is able to model the domain better when we use more than 30 games.

Finally, we will analyze the stability of the predictions produced by each classification algorithm. It is important to obtain some prediction that do not change constantly as the game progresses. Figure 7 shows the number of games at a given time for witch the prediction did not change for the rest of the game (in this experiment we make 20 predictions during each game at intervals of 5% of the duration). So, for example, when we reach the half of the game LDA will not change its prediction anymore for 10 out of the 20 games we are testing.

In conclusion, is this domain and using our game state representation, LDA seems to be the best classifier. It obtains a level of accuracy over 80% when only 55% the game has been played, it learns faster than the other algorithms from 30 games in the training set, and it is the most stable classifier for most part of the game.

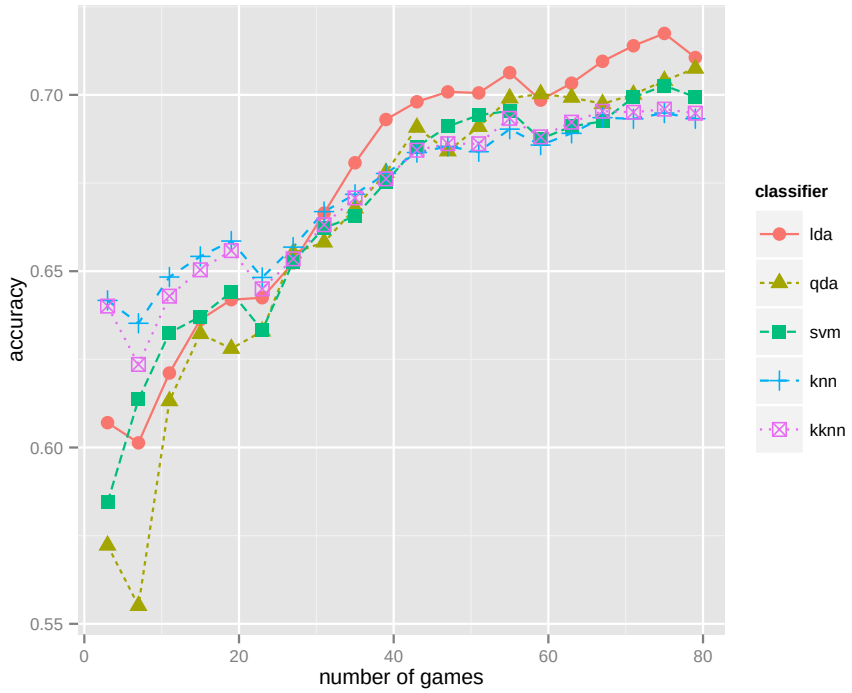

Fig. 6: Accuracy of classifiers depending on the number of games used to train them.

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[6]. Proofs of this are the different international competitions have taken place during the last years in AIIDE and CIG conferences[5, 4, 3]. We recommend [15] and [14] 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. [8] describes a fast Alpha-Beta search method that can defeat commonly used AI scripts in RTS game small combat scenarios. It also presents evidence that commonly used combat scripts are highly exploitable. A later paper [7] 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 [20], case-based reasoning [1], bayesian models [19] and evolutionary learning [16] among others.

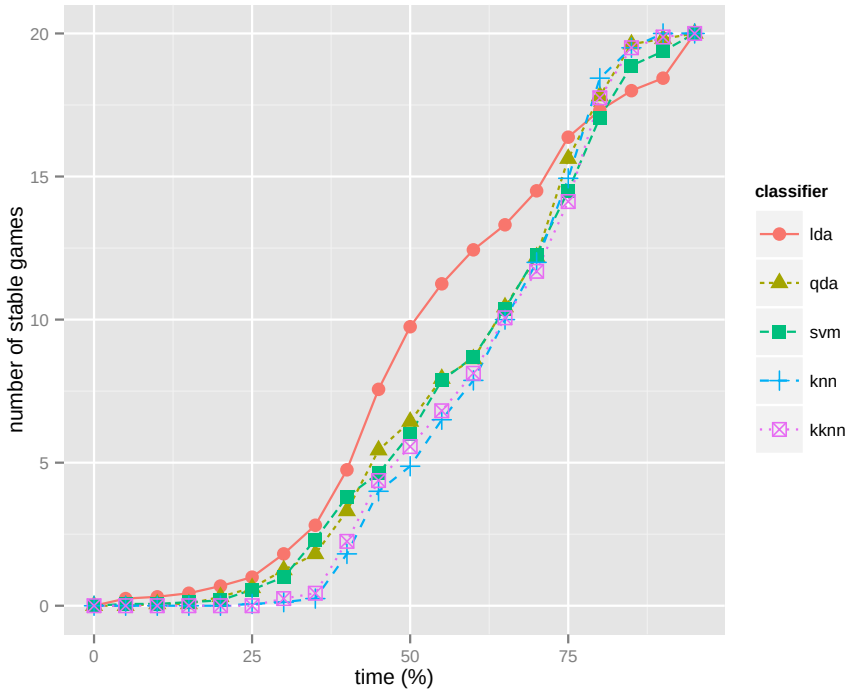

Fig. 7: Number of games for which each classifier becomes stable at a given time.

In [18] authors present a Bayesian model that can be used to predict the outcomes of isolated battles, as well as predict what units are needed to defeat a given army. Their goal is to learn which combination of units (among 4 unit types) is more effective against others minimizing the dependency on player skill. Our approach is different in the sense that we try to predict the outcome in whole games and not just the outcome of battles.

7 Conclusions and Future work

In this paper we have compared different machine learning algorithms in order to predict the outcome of 2 player Terran StarCraft games. In particular we have compared Linear and Quadratic Discriminant Analysis, Support Vector Machines and 2 versions of k-Nearest Neighbors. We have discussed the accuracy of the prediction as the game progresses, the number of games required to train them and the stability of their predictions over time. Although all the classification algorithms perform similarly, we have obtained the best results using Linear Discriminant Analysis.

There are several possible ways to extend our work. First, all our experiments take place in the same map and using the same StarCraft internal AI to control

both players. In order to avoid bias and generalize our results we will have to run more experiments using different maps and different bots. Note that it is not clear whether the accuracy results will improve or deteriorate. On the one hand, including new maps and bots will increase the diversity in the samples making the problem potentially more complex but, on the other hand, in this paper we have been dealing with an added difficulty that is not present in normal games: our games were extremely balanced because the same AI was controlling both players. Each bot is biased towards some way of playing, like humans, and we are not sure about the effect that may have in our predictions.

Another approach to extend our work is to deal with games with more than 2 players. These scenarios are much more challenging, not only because the prediction of the winner can take values from a wider range of possibilities but because in these games players can work in group as allies (*forces* in StraCraft terminology). On the other hand, we have addressed only one of the three available races in our experiments and, of course, in the game some units from one race are more effective against other units of other races.

Finally, in this paper we have chosen to use a high dimensional representation of the game state that does not take into account the distribution of the units and buildings in the map, only the number of units. We do not consider either the evolution of the game to make a prediction, we forecast the outcome of the game based on a picture of the current game state. It is reasonable to think that we could improve the accuracy if we consider the progression of the game, i.e., how the game got to the current state. We think there is a lot of work to do selecting features to train the classifiers.

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. Buro, M., Churchill, D.: AIIDE 2012 StarCraft Competition. In: Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, October 8-12, 2012 (2012)
4. Buro, M., Churchill, D.: AIIDE 2013 StarCraft Competition. In: Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, Boston, Massachusetts, USA, October 14-18, 2013 (2013)
5. Buro, M., Churchill, D.: AIIDE 2014 StarCraft Competition. In: Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014, October 3-7, 2014, North Carolina State University, Raleigh, NC, USA (2014)
6. 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)
7. 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)

8. 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)
9. Cortes, C., Vapnik, V.: Support-Vector Networks. Mach. Learn. 20(3), 273–297 (Sep 1995)
10. Fisher, R.A.: The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics 7(7), 179–188 (1936)
11. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, Springer New York Inc., New York, NY, USA (2001)
12. Hechenbichler, K., Schliep, K.: Weighted k-Nearest-Neighbor Techniques and Ordinal Classification (2004), http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:19-epub-1769-9
13. James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning: with Applications in R. Springer Texts in Statistics, Springer (2013)
14. Lara-Cabrera, R., Cotta, C., Leiva, A.J.F.: A review of computational intelligence in RTS games. In: IEEE Symposium on Foundations of Computational Intelligence, FOCI 2013, Singapore, Singapore, April 16-19, 2013. pp. 114–121 (2013)
15. 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)
16. 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)
17. Samworth, R.J.: Optimal weighted nearest neighbour classifiers. Ann. Statist. 40(5), 2733–2763 (10 2012)
18. 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)
19. 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)
20. 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)