Early Prediction of the Winner in *StarCraft* Matches

Antonio Álvarez-Caballero[1], J. J. Merelo[1], Pablo García Sánchez[2] and A. Fernández-Ares[1]
[1]*Department of Computer Architecture and Computer Technology, University of Granada, Spain*
[2]*Department of Computer Science, University of Cádiz, Spain*

Keywords: Prediction, Classification, Strategy, Planification.

Abstract: A fast and precise prediction of the outcome of a game is essential for the design of bots that play the game; it can be used either offline as a fast way to design bot strategies or online for conserving resources and conceding defeat or speed up victory, as well as evaluating the consequences of actions. The objective of this paper is predicting the winner of a *StarCraft* match as soon as possible. This study is done with supervised learning, because a lot of suitable data is available. The main problem of this approach is the big amount of generated data, so it has to be selected and organised properly and be treated with proper tools. A set of six learning algorithms is used, from simpler ones to more complex algorithms. *Spark* and *MLlib* are used due to their capabilities to deal with big amounts of data. With the learned models, time of matches are restricted, trying to get a time bound for predicting results. With this approach we get that it is not necessary to play a whole match to predict its winner with high accuracy: with 10 minutes we can predict the outcome with 90% of accuracy.

1 INTRODUCTION

Real Time Strategy or *RTS* games are a very suitable kind of videogames to use supervised learning with, because usually they have a very large set of features which could be used to analyze the game deeply. In particular, *StarCraft* is a RTS game from the 90s. In this game both players begin with a simple building and some workers, and the objective is to create an army that can defeat the opponent's one. This is usualy achieved by gathering resources from the map and building some important structures to get the best units for your army. However, there are infinite strategies to follow, and all of them are correct ones. Whatever strategy it is used, this process generates a lot of data, which can be used to get hidden information about the match.

Usually videogame developers do not allow access matches' data to users. *StarCraft* is not an exception, but their community has created an *API* to access data and manipulate the game itself: *BWAPI*. With this tool users can create artificial agents which play the game making competitions. This *API* is also commonly used for gathering data. These framework has been used in previous year to obtain a big number of game datasets. The conclusions that can be extracted from this data could be very important for researchers and players, because they could offer extra information such as the opponent strategy of matches in real time.

In this paper we prove that with a well designed set of *StarCraft* features, the winner of a match can be predicted accurately, even in an early stage of the match.

In this work a complete Knowledge Discovery in Databases (*KDD*) process is done. The data were collected from (Robertson and Watson, 2014), a set of six relational databases which contains a very big amount of data from more than 4500 *StarCraft* replays. A preprocessing with *SQL* was made to organise the data and extract our set of important features. Finally, the modelling was made using *Spark* and *MLlib* due to their capability to deal with a big amount of data, allowing us to extract useful information as the winner in an early stage of the matches and a ranking of useful features.

The main conclusion obtained is that the winner can be predicted without playing the whole match. With 10 minutes approximately, it is enough to get predictions with an accuracy ratio of 90%. Keeping in mind that the average duration of a match is 48 minutes approximately, the time reduction is considerable. It could be useful combined with metaheuristics to optimize agents for this videogame faster, by

ÃĄlvarez-Caballero A., Merelo J., GarcÃ■a SÃąnchez P. and FernÃąndez-Ares A.
Early Prediction of the Winner in StarCraft Matches.
DOI: 10.5220/0006587304010406
In *Proceedings of the 9th International Joint Conference on Computational Intelligence (IJCCI 2017)*, pages 401-406
ISBN: 978-989-758-274-5

using a surrogate-based model. Furthermore, an agent with the possibility of predicting accurately the winner could adapt its strategy to change the outcome of the match.

This could not be achieved without a good set of features. Training a classifier is easy, but it does not help if data have no quality. This set of data and features could be used in other works based on *StarCraft* data to try to improve their results.

2 STATE OF THE ART

In the *StarCraft* research a lot of approaches have been presented. The most used approach is developing probabilistic graphical models to predict the winner of a match. Some examples are in (Synnaeve and Bessière, 2011) and (Stanescu et al., 2013), where important events in the match are used to predict the outcome: when a very important building appears, an important event succession for a race, the birth of the best unit of a race, etc.

Another approach based on supervised learning is presented in (Sánchez-Ruiz, 2015), but the environment is homogeneus and controlled. It is possible that it doesn't show the diversity in *StarCraft* matches. A better dataset is presented in (Robertson and Watson, 2014), which is very heterogeneus, complete and granulated.

Further works look for plans and strategies based on predictions of the outcome of matches, as we can see in (Oen, 2012) and in (Alburg et al., 2014).

Another approach is developing strategies using *Genetic Programming*, creating plans automatically which can win. These kind of algorithms are very time consuming, so whatever saved time would be appreciated. This approach gives good results, as we can see in (Fernández-Ares et al., 2016) and (García-Sánchez et al., 2015).

3 METHODOLOGY

In this paper we do a complete *KDD* process using *SQL* and some Apache tools: *Spark* with its *Scala* interface and *MLlib*. We did this election because Apache echosystem is suitable for dealing with very large datasets, offering a framework which produces similar projects in centralized and distributed environments.

Its *Scala* interface was chosen because it is the most complete one for *Spark* and *MLlib*. The only thing it is missed is an implementation of *KNN*, so `saurfang:spark-knn` from spark-packages is taken. Furthermore, *Scala* is a modern, functional and object-oriented language which is used widely in some companies as LinkedIn, Twitter or Siemens. One of these advantages is that *Scala* compiles to the Java Virtual Machine or *JVM*. As a consequence, multiplatform code is developed. This code is available at GitHub, https://git.io/vdmyj.

3.1 Feature Selection

The data we use is taken from (Robertson and Watson, 2014), who with their work offer six relational databases of one versus one matches, with all the possible combinations of races that the game offers.

In Figure 1 we can see the entity-relationship diagram of the databases that contain the matches. Understanding all features was easy because (Robertson and Watson, 2014) work is totally open, so we could explore the code associated. Furthermore, a lot of features have the same name that attributes from the *BWAPI*, although a set of features was calculated by the researchers like the distance to the base in a moment of the match.

To get a rows and columns dataset, we propose this structure. Each row of the dataset will be a precise instant of the match, determined by a *Frame*. Each instant has the information of resources of each player. This approach is different to other ones presented in Section 1. It seems easy but the organisation of the data was not trivial.

We present here the list of selected features, also exposed in Figure 1. Some of them are used only to organise the data, the identifiers of replay, player and region. Features which are used to model are **bold**.

- replay: Contains data about each match.
  - ReplayID: Match identifier.
- playerreplay: Contains data about a player in a match.
  - PlayerReplayID: Player identifier.
  - ReplayID: Match identifier.
  - Winner: Winner of the match. This is the target we want to predict.
- resourcechange: Contains data associated to changes in player's resources.
  - PlayerReplayID: Player identifier.
  - **Frame**: Frame when the resource changes.
  - **Minerals**: Amount of minerals of a player.
  - **Gas**: Amount of gas of a player.
  - **Supply**: Carrying capacity of a player.
  - **TotalMinerals**: Total amount of minerals of a player, without costs.

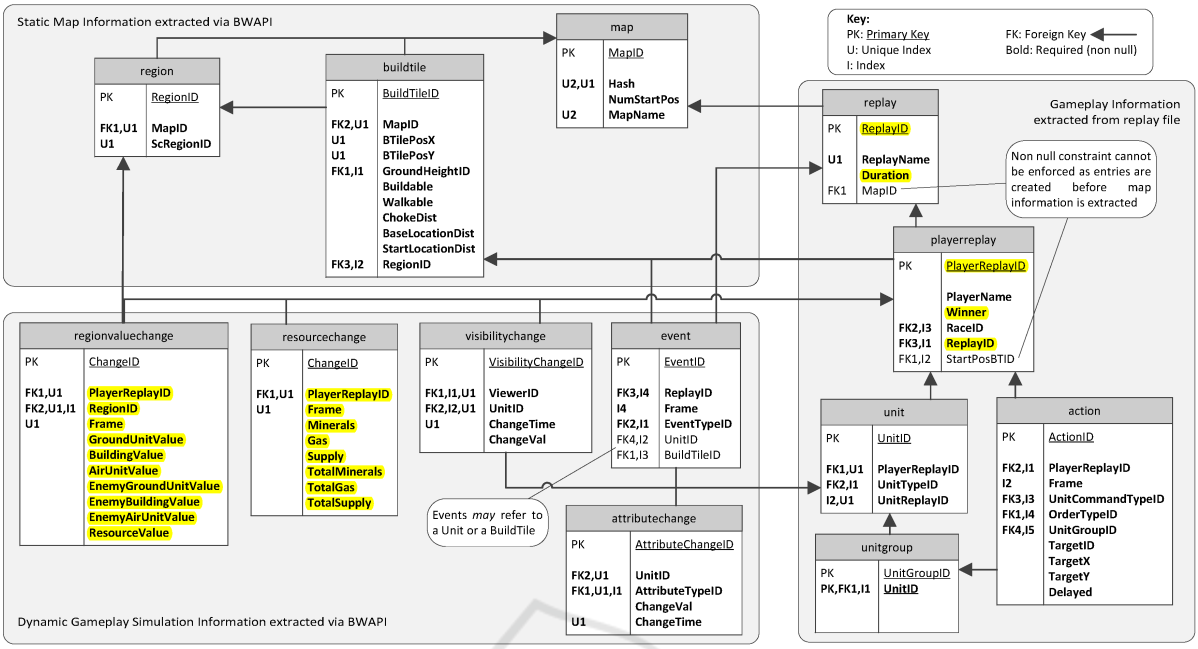

Figure 1: Selected features in databases. Original diagram by (Robertson and Watson, 2014).

  - **TotalGas**: Total amount of gas of a player, without costs.
  - **TotalSupply**: Total amount of supply of a player, without costs.
- regionvaluechange: Contains data associated to changes of a player in a map region. Each *value* is the sum of the price of an unit, expressed as minerals and gas.
  - PlayerReplayID: Player identifier.
  - RegionID: Region of the map identifier.
  - **Frame**: Frame when the value changes.
  - **GroundUnitValue**: Value of ground units in this region.
  - **BuildingValue**: Value of buildings in this region.
  - **AirUnitValue**: Value of air units in this region.
  - **EnemyGroundUnitValue**: Value of enemy ground units in this region. This value is estimated, the player knows the units they see of the enemy.
  - **EnemyBuildingValue**: Value of enemy buildings in this region. This value is estimated, the player knows the units they see of the enemy.
  - **EnemyAirUnitValue**: Value of enemy air units in this region. This value is estimated, the player knows the units they see of the enemy.
  - **ResourceValue**: Value of resources in this region. This value is estimated, the player knows the units they see of the map. If the player does not know a region, they estimate this value as the maximum available in the region.

The features related to a player are presented twice, one for each player. Furthermore, a **race** feature is used, as the races implied in the match. As a consequence, we have 28 features used in the modelling stage of the work.

3.2 Data Preprocessing

As we said in Section 3.1, each observation of the data has 28 features related to the resources and units of each player. We have to note some little tips about the data organisation.

- Values of *Frame* are the instants when any player has a minerals, gas or supply change. We did this election because in these games the basic resources are always changing due to necessity of them to buy units or buildings. This implies some missing values in the resources of the player who doesn't make a change in that frame, so we had to recover the last value of that resource to keep consistency.
- Values of *value* of units and buildings changes do not occur in the same instant that the resources' changes. For this reason, as we did with *Frame*, we had to recover the last *value* of the region *values* of each player, too.

- *Value* depends of the region of the map. It's important in our approach to get a full *value* measure, a value that represents all the player units or buildings, so we had to sum the *value* of every region of the map.

3.3 Modelling

We have chosen six algorithms for the modelling. Some of them are very simple, because we want to test if a simple algorithm can model the data.

- *Naive Bayes* (NB)
- *Logistic Regression* (LR)
- *KNN* (KNN)
- *Multilayer perceptron* (MLP)
- *Random Forest* (RF)
- *Gradient Boosting Tree* (GBT)

The final results are obtained using this approach 5 times:

1. Split data using 70% to train and 30% to validation.
2. Inside the first partition, train a model using 10-fold Cross Validation.
3. With the final model, validate it in the second partition.

Using this approach, we get 5 accuracy measures for each model, so we can test the obtained samples using appropiate statistical tests. We use *Friedman test* to get statistical differences among all the classifiers, and a pairwise *t-test* to get differences between classifiers.

Accuracy is used because some algorithms as *Gradient Boosting Tree* or *Multilayer perceptron* does not implement a predict function which returns the probability of each class: they only return the class. With this important restriction *Area Under Curve* cannot be used. Furthermore, the data is not imbalanced so accuracy is a suitable measure.

4 RESULTS

In Figure 2 and Table 1 is exposed the median of the accuracy of each model with their parameters. The accuracy is evaluated in the validation sets. Standard deviation of the accuracies are shown as error bars.

Median with standard deviation is shown because data is not centered around the mean in all the selected instants.

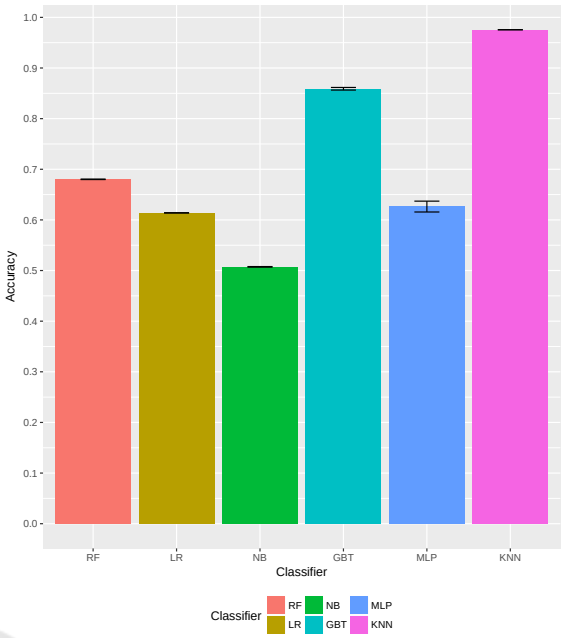

Figure 2: Measures obtained with all models.

In general, all classifiers but *Naive Bayes* predict pretty well, but there are two clear winners: *Gradient Boosting Tree* and *KNN*. This one offers a great accuracy with a very simple approach, but it is not suitable for doing a lot of predictions in real time, because of its lazy approach.

In Figure 3 is exposed the accuracy differences between classifiers over time. *KNN* can classify

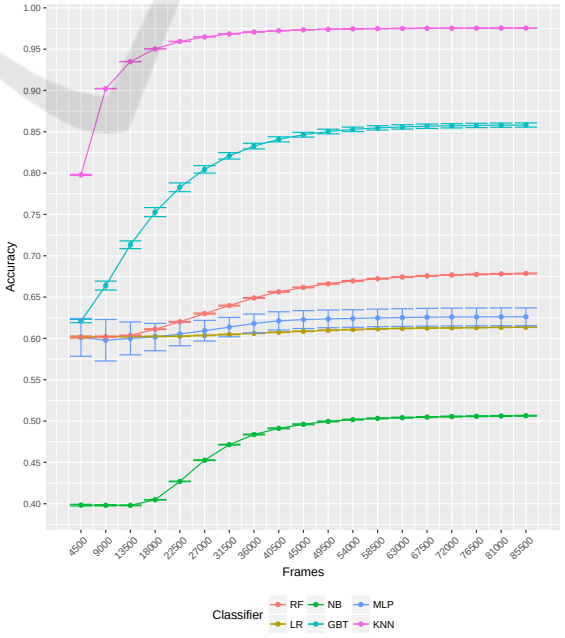

Figure 3: Comparing classifiers over time.

Table 1: Trained algorithms with their final parameters.

	Classifier	Accuracy	SD	Params
1	GBT	0.859000	0.002563	numTrees = 150, maxDepth = 10
2	KNN	0.975589	0.000091	K = 3
3	LR	0.613878	0.000294	maxIter = 150, regParam = 0.3
4	MLP	0.626332	0.010706	Hidden layers = (10,10)
5	NB	0.507295	0.000550	Smoothing = 1
6	RF	0.680166	0.000474	numTrees = 150, maxDepth = 10

Table 2: Accuracy of classifiers in each validation set.

	RF	LR	NB	GBT	MLP	KNN
1	0.680004	0.613650	0.507551	0.859000	0.605377	0.975575
2	0.680166	0.614189	0.508414	0.854445	0.626332	0.975589
3	0.679876	0.613430	0.507042	0.860723	0.628845	0.975416
4	0.681080	0.613981	0.507155	0.856990	0.611386	0.975620
5	0.680194	0.613878	0.507295	0.860127	0.627228	0.975652

Table 3: p-values from a pairwise t-test.

	RF	LR	NB	GBT	MLP
LR	0.000000				
NB	0.000000	0.000000			
GBT	0.000000	0.000000	0.000000		
MLP	0.000486	0.277633	0.000058	0.000004	
KNN	0.000000	0.000000	0.000000	0.000000	0.000001

with a precision of $0.902 \pm 3.292 \times 10^{-4}$ only with 10 minutes of match. The mean of the duration is 4.397×10^4 frames, which are equal to 48.854 minutes. It implies that with only a 20.469% of the mean duration of the match, we can predict accurately the winner of a match. It is not necessary to play the whole match to get the winner with high confidence.

To complete the study, we present a Friedman test to see significative differences over the classifiers. All frames of the matches are used. We can see in Table 2 the accuracy of each classifier.

With a p-value of 1.889×10^{-4}, we can confirm that statistical signification exists. We can see in Table 3 a pairwise test, with *Bonferroni* adjust method of the p-value.

As we can see, we can confirm that with the usual signification level, 0.05, there are statistical differences among all classifiers but *Logistic Regression* and *Multilayer Perceptron*. They are not the best classifiers so this is not important. The important fact is that there are statistical differences between *KNN* and *Gradient Boosting Tree*.

5 CONCLUSIONS

With this study we can extract some conclusions. The first one is that the set of features is well formed: the winner can be predicted from the selected features with high confidence, as we exposed in Section 4.

The second and main conclusion is that predictions with this data are very accurate in an early stage, in particular using a *KNN* classifier. It can predict with 90% of accuracy using 10 minutes of match only. This is very important because it proves that it is not necessary to play a whole match to predict the winner accurately.

As future work there are some research opportunities using this study. With a *KNN* classifier, a competitive *bot* could be developed with an important skill: the winner's prediction. It could be used to improve the adaptability of the agent, giving advantage to its opponents.

Another work could be the improvement of *bot* optimization using the early winner knowledge. We could use this knowledge to improve the evaluation step of the algorithms, giving the opportunity to use more exhaustive setups on the algorithms.

ACKNOWLEDGEMENTS

This work has been supported in part by: de Ministerio español de Economía y Competitividad under project TIN2014-56494-C4-3-P (UGR-EPHEMECH) and by CONACYT PEI Project No. 220590.

REFERENCES

Alburg, H., Brynfors, F., Minges, F., Mattsson, B. P., and Svensson, J. (2014). Making and acting on predictions in starcraft: Brood war. Master's thesis, University of Gothenburg.

Fernández-Ares, A., García-Sánchez, P., Mora, A. M., Castillo, P. A., and Merelo, J. J. (2016). There can be only one: Evolving RTS bots via joust selection. In Squillero, G. and Burelli, P., editors, *Applications of Evolutionary Computation - 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings, Part I*, volume 9597 of *Lecture Notes in Computer Science*, pages 541–557. Springer.

García-Sánchez, P., Tonda, A. P., Mora, A. M., Squillero, G., and Guervós, J. J. M. (2015). Towards automatic starcraft strategy generation using genetic programming. In *2015 IEEE Conference on Computational Intelligence and Games, CIG 2015, Tainan, Taiwan, August 31 - September 2, 2015*, pages 284–291. IEEE.

Oen, R. E. (2012). Aspire adaptive strategy prediction in a rts environment. Master's thesis, University of Bergen.

Robertson, G. and Watson, I. D. (2014). An improved dataset and extraction process for starcraft AI. In Eberle, W. and Boonthum-Denecke, C., editors, *Proceedings of the Twenty-Seventh International Florida Artificial Intelligence Research Society Conference, FLAIRS 2014, Pensacola Beach, Florida, May 21-23, 2014*. AAAI Press.

Sánchez-Ruiz, A. A. (2015). Predicting the winner in two player starcraft games. In Camacho, D., Gómez-Martín, M. A., and González-Calero, P. A., editors, *Proceedings 2st Congreso de la Sociedad Española para las Ciencias del Videojuego, Barcelona, Spain, June 24, 2015.*, volume 1394 of *CEUR Workshop Proceedings*, pages 24–35. CEUR-WS.org.

Stanescu, M., Hernandez, S. P., Erickson, G., Greiner, R., and Buro, M. (2013). Predicting army combat outcomes in starcraft. In Sukthankar, G. and Horswill, I., editors, *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-13, Boston, Massachusetts, USA, October 14-18, 2013*. AAAI.

Synnaeve, G. and Bessière, P. (2011). A bayesian model for opening prediction in RTS games with application to starcraft. In Cho, S., Lucas, S. M., and Hingston, P., editors, *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31 - September 3, 2011*, pages 281–288. IEEE.