Supervised learning as a tool for metagame analysis

Rob Argue
University of Maryland
Department of Computer Science
rargue@cs.umd.edu

1 Abstract

I explore the possibility of using supervised learning as a tool for analyzing metagames. In particular I look at a data set for Natural Selection 2 build 229. The first goal of the article is to predict the outcome of individual matches based on a set of features of the round which capture the strategies being used. Using an averaged perceptron, I predict the outcome of a round with 85.42% accuracy. The second goal is to attempt to use machine learning to analyze the balance of different strategies in the metagame. Balance in this context is taken to mean a similarity in the prevalence of parallel strategies. I use a random forest of decision trees to extract the most important feature that determines the outcome of the game over a subset of all features. This exposes player total resources and player skill as the most important features, which was to be expected, and shows an imbalance in one of the strategies in the game which was largely considered to be imbalanced by the community in build 229. These methods are effective in both predicting outcomes and identifying potential balance issues, and provide a tool for the automated analysis of metagames.

2 Introduction

2.1 Background

An important part of game design is understanding games from a strategic level. For the purposes of this article, I define *strategy* to mean the particular set of actions a player or team uses during one round (instance of the game). The set of all strategies is thus the set of all possible actions available. Note that not all actions may necessarily be available to a player at the same time, i.e., one action may preclude another or have a prerequisite action. I refer to a set of strategies for which one strategy precludes all others as *parallel strategies*, and refer to a series of actions which are ordered by prerequisites as *linear strategies*. In this context, an *asymmetric game* is defined as one in which all players/teams do not have the same set of available actions for any given round. *Metagaming* means any strategic decisions which are made outside of the game, and the *metagame* of a game is defined as the collection of strategies used by all players at a particular point in time. Thus the study of metagames is the study of the utilization of strategies by players across a game as a whole and how that changes over time.

2.2 Problem

Game balance is an important topic in the success of a game, and it is especially crucial for multiplayer games. Without balance being maintained to within a reasonable degree, players will lose interest in a game and cease to play it. Balance here is defined as the equal effectiveness of all parallel strategies within a metagame. It is important to note that there are two main consequences of an imbalance in a game: if one strategy is more effective, it can lead to an overabundance of that strategy in the metagame, or in the case of an asymmetric game, imbalance can lead to one player having an innate advantage in any given round.

It is easy to identify a game as unbalanced. such games have a lopsided win ratio between strategies, or a disproportionately high percentage of the metagame using a single strategy or small subset of strategies. The difficult part of balancing comes in trying to understand why a particular strategy is more successful or prevelant, and how that imbalance can be addressed. A further complication arises in games in which the overall strategy can be decomposed into a set of substrategies since a single substrategy may be causing the imbalance.

I propose using supervised learning as a tool to analye metagames with the intent of a) being able to predict the outcome of games based on the the strategies employed by both teams, and b) being able to extract information on strategies that may be unbalanced in a given metagame. In particular, I use perceptrons and a random forest as supervised classifiers.

2.3 Perceptrons

The perceptron (Rosenblatt, 1958) is a linear classifier which take a weighted sum of multiple inputs, and passes them through a threshold function to produce an output signal. It has been shown to be an effective means of classifying data that is linearly separable. To improve how well the perceptron generalizes, it can be extended into a voted perceptron (Freund and Shapire, 1999). A voted perceptron tracks the number of correct classifications each set of weights gets correct, and uses that information to assign each set of weights a number of votes for prediction. This can be sped up by instead using an averaged perceptron, which keeps an average weight vector rather than all previous weight vectors and their votes.

Additionally, the perceptron can be extended to handle non-linear decision surfaces by including products of the inputs. The kernel trick (Aizerman et. al., 1964) allows polynomial expansions of the inputs to be used with only a constant factor penalty to the runtime.

2.4 Random forest

Random forests (Breiman, 2001) are an ensemble method for classification which uses the mode response over a large number of decision trees to classify an input. Each tree is created over a random subset of the features. Random forests differ from other ensemble tree methods such as bagging (Breiman, 1996) in that they use a subset of features to construct each tree rather than a subset of the data. Random forests have been shown to have accuracy comparable to other popular supervised learning algorithms, while remaining more resilient to the presence of noise.

2.5 Data

I have chosen to use Natural Selection 2 (NS2) as a case study for this article. This game allows a combination of individual skill and team strategy, is asymmetric, has a set of well-known strategies that break into substrategies (in this case, single decisions), and has a known imbalance. In particular, I consider build 229 of NS2, where one particular strategy is cited by players as being the reason one team has a large advantage.

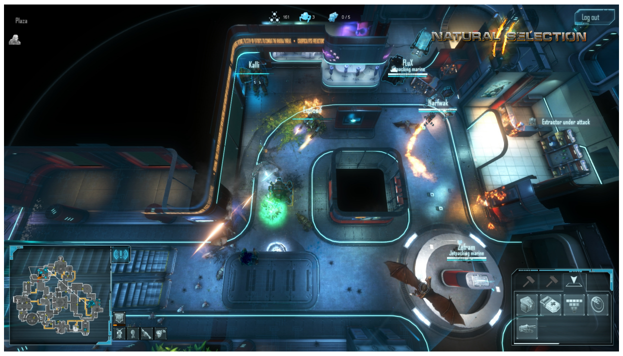

Fig. 1: NS2 commander interface [1]

2.5.1 Natural Selection 2

Natural Selection 2 (NS2) is an asymmetric first-person shooter (FPS) / real-time strategy (RTS) hybrid game. Two teams, aliens and marines, are tasked with eliminating the opposing team's base. Each team's size can vary, but rounds are most commonly played with 3 to 12 players per team. Each team has a commander who plays the game as a top-down RTS, while the rest of the team controls individual units and plays the game as an FPS. Teams can place resource towers (RTs) on resource nodes to collect resources, which are in turn used to purchase upgrades. Resources go to two pools: team resources (T-res) and personal resources (P-res), which are used by the commander and players respectively. T-res can be spent to place structures and purchase upgrades for the entire team, either automatically applying the upgrade to the players, or unlocking things for players to buy. P-res can be used to purchase upgrades for an individual player, some of which can also be purchased for a player by the commander using T-res. Some upgrades are limited to a team controlling a number of tech points, which can be captured by purchasing and building a command station for the marines or a hive for the aliens.

Winning a round comes down to a combination of individual player skill and overall strategy. I use kill-to-death ratio (KDR) as an approximation of player skill, which is generally agreed to be the case on a large scale. Strategy breaks down into two major areas: upgrades and map control. Upgrades are simply the upgrades that are purchased by the commander and the time at which they are purchased (from which upgrade order can be extracted). Map control

[1] Retrieved April 28, 2014 from http://unknownworlds.com/ns2/media/

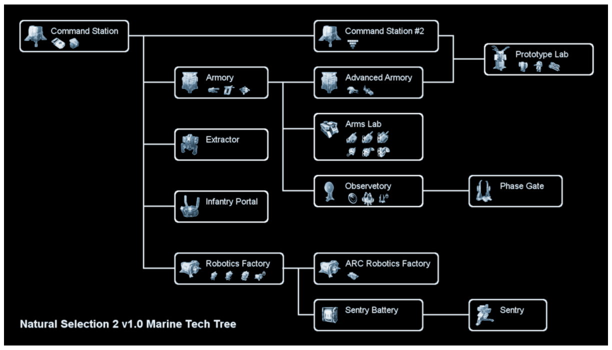

Fig. 2: Marine tech tree [2]

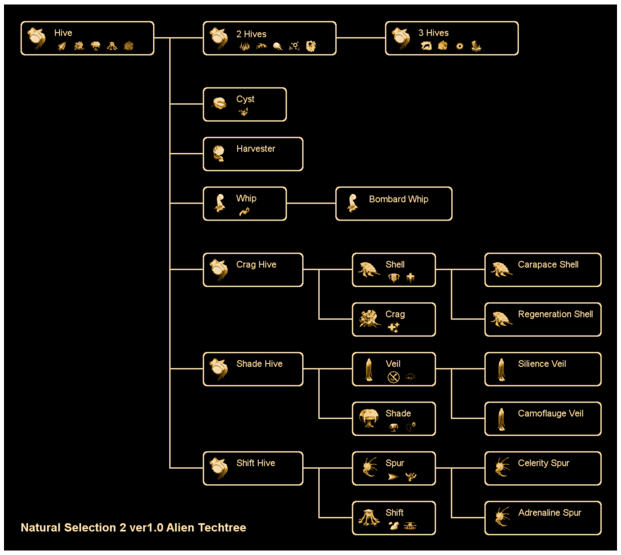

Fig. 3: Alien tech tree [2]

is harder to define, but includes which rooms are controlled by which team and which locations are being pushed and/or defended. Rooms can have strategic value from their position on the map, and from the resource nodes and tech points they contain.

In NS2 the teams have parallel strategies available, i.e., their choices of actions (insofar as this article is concerned) are mutually exclusive. Each team has a varying amount of parallel and linear strategies. The marine team has a mostly linear strategy, with a few parallel branches (Fig. 2), while the alien team has a mostly parallel strategy (Fig. 1) with three major branches (shift, shade, and crag).

2.5.2 Data Set

The data I use for this article is a SQL database dump collected by NS2 Stats (NS2stats, 2012). I limit the data to NS2 build 229, because that was the most interesting build available in terms of balance due to the alien's crag branch being overly effective. I further limit the data to rounds between 1 minute and 2 hours in order to eliminate non-games and provide a maximum value for time. The time limitation is important so that there would not be an artificially large bias on events that did not occur, as their time of occurrence is set to the maximum time value. From this I use a training set consisting of about 2500 rounds and a test set consisting of a out 700 rounds. I use C# and the LinQ library to extract features from the database and store them in a CSV file for use in machine learning algorithms. The particular set of features I use for this article are round length, total KDR for each team, marine-to-alien T-res gathered ratio, marine-to-alien T-res used ratio, and the initial purchase time of structures and upgrades. I omit all alien structures except the hive, as they are either directly tied to an upgrade and thus redundant, or fall into the category of structures used for map control which is beyond the scope of this article. Likewise sentries and sentry batteries are omitted from the marine team. All buildings and upgrades not purchased are set to the maximum time value.

This particular set of features focuses on the strategy in terms of the RTS side of the game and for the most part filters out combat strategy. The reasoning for this is that combat strategy information is not readily available and would have to be reconstructed from the dataset, which would be a monumental task by itself. All information regarding map control is omitted due to the difficulty of use. The full information on the building and destroying of RTs is only included vaguely implicitly in the resource features and could be expanded upon. The number of tech points is also only included implicitly by the upgrades limited to a certain number of tech points that are purchased, however the first purchase of a second tech point is included as a feature. Note specifically that this completely omits any direct information about a third hive for the aliens. Player skill is wrapped into a single team feature, which, while very indicative of the skill of a team, does not provide any granularity which may be important. Further information could be extracted regarding the spread of skill across a team or about player skill by using information from other rounds. I omit this additional information to improve the robustness of the algorithms in dealing with new players as well as player skill improving over time.

[2]Retrieved April 28, 2014 from http://forums.unknownworlds.com/discussion/122833/techtree

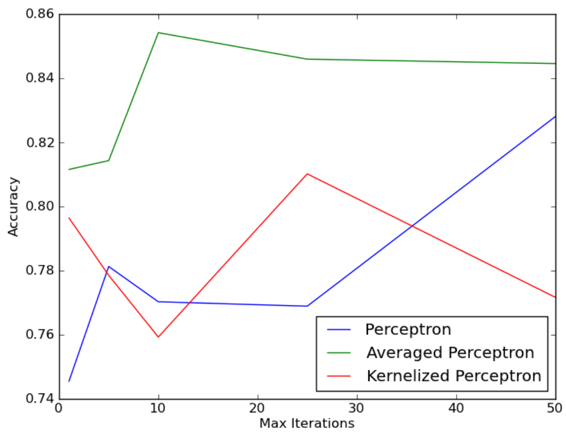

Fig. 4: Max iterations vs. perceptron accuracies on test data

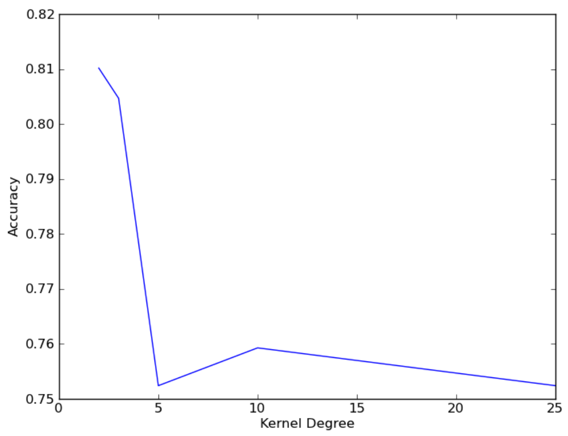

Fig. 5: Kernel degree vs. kernelized perceptron accuracy on test data

3 Experiments

3.1 Predicting round outcomes

The first step to show that machine learning is a viable analysis tool for strategic content of metagames is to demonstrate that a machine learning algorithm can construct an accurate model of outcomes of games based on the strategies employed. In this particular case, it should be noted that player skill also is a large factor in the outcome, but that is due to the nature of the game rather than a particular issue with the methods used.

While the decision surface for this space is not linear, I make the assumption that it would be roughly linear. This intuition comes from considering that any one feature has a direction in which the "goodness" increases monotonically. For any combination of features, the goodness of each feature does not detract from the goodness of the other features, thus the decision surface should be linear. The decision surface will be "fuzzy" to an extent as well. For similarly effective opposing strategies, either team could win with some probability. Considering this, I start with a perceptron as a predictor. In particular, I use 50 inputs comprising the 49 features from the data set and a bias. For the output function I use a step function. All weights are initialized to 0.

In order to improve generalization, an averaged perceptron is implemented. To capture the non-linear nature of the decision surface, I use a kernelized perceptron using polynomial kernels of varying degrees. For each type of perceptron, I test a range of max iterations, with the best result for the kernel (a degree-3 polynomial) representing the kernelized perceptron. The resulting accuracies on the testing data are summarized in Fig. 4 and Fig. 5. The highest accuracy of 85.42% is achieved by the averaged perceptron with 10 iterations.

Considering that this dataset had a limited number of features as compared to the total information available, this method preforms well. It is interesting to note that the linear classifier performs better than the non-linear one, which suggests that the assumption made about the linear nature of the decision surface is correct. The advantage to using a linear classifier in this case, where the boundary can often be fuzzy, is that it avoids overfitting the data. It would be interesting to expand this to filter player skill out of outcome prediction, however, the size of this data set is too small to accurately do so for NS2, and therefore it is left as future work.

3.2 Discovering important strategies

The next experiment is to see if machine learning could be used to automatically discover unbalanced strategies which are. In this context, I consider a strategy to be unbalanced if it is disproportionately important to the outcome of the game as compared to other strategies. The intuition behind this is that if all strategies are perfectly balanced then, while one may be more effective in a particular situation, they should be evenly distributed across the metagame.

I use a random forest of decision trees to discover which strategies are most important to game outcome. In order to do so, I transform the continuous features into categorical features with splits discovered by experiment to be effective. Each feature is given a similar number of splits (17 to 18) so the number of possible decisions per strategy is equal. The forest has 5000 decision stumps, each created over a random subset of features of size 5. Then a tally is taken over the stumps of feature names. Adding together some of the similar features (which effectively described the same strategy) gives the following list for the most prevalent features for the decision trees in the forest (note that only the top 10 are shown):

Feature	**Count**
MarineToAlienResRatio	933
KDR	856
Leap	377
BileBomb	345
Blink	302
Regeneration	235
CragHive	231
Carapace	228
Spores	208
ShadeHive	171

Table 1: Feature usage count in randomized forest

The most important feature to the outcome of a game is the resource ratio, which is generally considered to be a strong indicator of how well a team is doing according to the playerbase of the game. The neat most important is KDR, which approximates player skill. I anticipated in advance that both of these would be the determining factors of the game, and indeed the forest recovers this information. Also of note is the three features from just one of the three branches in the alien tech tree (Regeneration, CragHive, Carapace). Considering that is a branch in the strategy, it is expected to hold the same importance as the other two branches (Shift and Shade). By being placed significantly higher than Shift and Shade features, the forest shows an imbalance in the game. This corresponds to the general player opinion of the game that that branch was more effective than it should have been during build 229. I note, however, that the results of the forest do not necessarily indicate that this strategy is overly effective, just that it is not of the same level of importance as the other two branches. In addition to this, some of the early alien lifeform abilities (Leap, BileBomb, Blink, Spores) show up as relatively highly significant in the metagame. These are a more linear part of the alien strategy, however, and they cannot be compared to any other part, and thus need to be compared to the similar linear strategies on the marine team. The top marine-centric feature in this list was in the 47th position, indicating that the strategies for the marines were more balanced. Comparing the linear sets of strategies of the two teams, it appears that the alien upgrades are a much bigger influence on deciding the outcome of a game, suggesting an imbalance between teams in that regard. In that this is an asymmetrical game, that may be an intentional design decision, but it remains a source of imbalance that should at the very least be considered.

4 Discussion

Supervised learning methods show promise for the analysis of metagames. I demonstrate that methods can predict the outcome of a round based on the strategies used, as well as identify problems with the balance of a game. This should generalize to the metagame of any game, assuming features are chosen appropriately. This provides a useful tool for both automating the study of metagames as well as revealing potential balance problems before they become an issue.

Suggested future work in this would be to consider a data set where player skill can be more easily factored out, allowing it to be considered entirely separately from strategy. Another suggestion is to apply these methods to a data set for a trading card game, as they can have highly diverse and strategically deep metagames.

References

[1] Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 (5), (Nov. 1958), 386-408. DOI: http://dx.doi.org/10.1037/h0042519

[2] Yoav Freund, Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37 (3), (Dec. 1999), 227-296.

[3] M. A. Aizerman, E.A. Braverman, and L. Rozonoer. 1964. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, (1964), 821-837.

[4] Leo Breiman. 2001. Random Forests. *Machine Learning*, 45 (1), (Oct. 2001). 5-32. DOI: http://dx.doi.org/10.1023/A:1010933404324

[5] Leo Breiman. (1996). Bagging predictors. *Machine Learning* 26 (2), (Aug. 1996) 123140.

[6] NS2 Stats. 2012. Retrieved November 2012 from www.ns2stats.com