Proceedings of the Federated Conference on Computer Science and Information Systems pp. 139–142
DOI: 10.15439/2017F568
ISSN 2300-5963 ACSIS, Vol. 11

Multi-model approach for predicting the value function in the game of Heathstone: Heroes of Warcraft.

Alexander Morgun
Email: alexander.morgun.usu@gmail.com

***Abstract*—This document describes the problem presented at AAIA'17 Data Mining Challenge and my approach to solving it. In terms of reinforcement learning the task was to build an algorithm that predicts a value function for the game of Hearthstone: Heroes of Warcraft. I used an ensemble of 85 models trained on different features to build the final solution which scored the 36th place on the final leaderboard.**

***Index Terms*—data mining competition; classification; ranking; faeature engineering; algorithm composition; hearthstone;**

I. Introduction

IN THE RECENT time, there are many pieces of research about applying machine learning algorithms to video games. The most famous of them led to the creation of a learning model for playing Atari 2600 games [1]. This work was followed by many others, including the ones about Starcraft [2] [3] [4] and DotA [5] [6]. In summary, the main point of these works is developing new machine learning approaches which can be applied to other areas such as algorithm trading [7] or robotic manipulation [8] later. Following this trend the task of AAIA'17 Data Mining Challenge was to create an efficient prediction model which would help in building an agent capable of playing the game of Hearthstone: Heroes of Warcraft.

A. Game description

Hearthstone: Heroes of Warcraft is a turn-based card video game. It is played by two players. Each player uses his own deck of 30 cards. The final goal of each player is to destroy the opponent's hero by setting his health points to zero. To do so each turn players can play cards limited by fixed amount of mana crystals. There are three types of cards:

- Minions - creatures that can be placed on the game board. Minions can attack enemy minions and the hero.
- Spells - single-use cards that cause various effects.
- Weapons - cards that allow the player's hero to attack enemy minions trading health for board advantage.

In addition, minions can have different abilities. The most notable of them is *taunt*. Players minions cannot attack enemy hero until there are enemy minions with taunt on the board.

Complete rules can be found on the official site [11].

B. Problem statement

The task of AAIA'17 Data Mining Challenge was to predict the probability of player's victory. The metric used for the evaluation was the ROC-AUC metric [9]. We can see that this probability can be used as a value function for building an agent capable of playing the game [10]. Each provided game state got data about minions played by player and opponent, cards in player's hand and state both of the heroes. Information about active secrets and the history of played cards was unavailable.

TABLE I
Comparison of the local cross-validation to the public leaderboard results. We can see that an improvement in the cross-validation results does not result in a better score on the public leaderboard and cannot be used as a reliable way of validation.

Local validation	Public leaderboard
0.883	0.788
0.837	0.7926
0.877	0.7924
0.810	0.7842
0.816	0.7754
0.824	0.7646
0.8125	0.7819

C. Related Work

Hearthstone: Heroes of Warcraft is not a well-studied enviroment. There are very little scientific publication about applying machine learning methods to this game [16]. This could be explained by the fact that the game's license agreement [17] explicitly prohibits the use of bots and modification of the game files. Regardless of this fact bots exist in the game but there is no publicly available description of their algorithms.

II. The proposed solution

A. Local validation

I used 5-fold cross-validation stratified by target labels. This method helped me in choosing hyperparameters for each algorithm but it was not very correlated with the public leaderboard. Because of that, I choose my final solution based on its public leaderboard score. My validation scheme was flawed because of the information leakage between train and test folds. This leakage was caused by states from the same game appearing in the train and test folds.

B. Bagging

Because of the technical limitations, I could not train models on all provided data so I used a technique similar to the bagging [14]. First, I created multiple different subsets of the training data by sampling randomly the examples, trained separate instances of the same model on this subsets and got a number of finally different models which predictions could be combined afterwards. Random subsampling allowed the final composition to use information from the whole dataset.

C. Composition methods

Due to unreliable local validation results, I decided to use a simple averaging instead of more complex techniques like stacking and blending. The idea was that I did not want to favor any algorithm because I did not know its true quality so I averaged highly correlated answers before final averaging to avoid higher weights of them in the final result. I used two methods of averaging [13]:

1) Averaging of probabilities: I calculated the arithmetic mean of probabilities returned by all models.

2) Rank averaging: ROC-AUC is a ranking metric and correct ordering of test states is more important than predicting precise probabilities. Averaging ranks instead or probabilities helps with different calibration of classifier probabilities.

D. Models

1) Tree-based models: Most of the used models were XGBoost tree ensembles [15]. I also used one random forest.

2) Linear models: I used two kinds of linear models: linear classifier trained by stochastic gradient descent with log loss and multilayer perceptrons with different numbers of hidden layers and neurons.

E. Features

1) Baseline features: Features provided by competition hosts was used to train baseline models and measure the quality of other features I tried. My first model was the XGBoost model trained on these features. Leaderboard scores showed that rank averaging of a few XGBoost models with different maximum tree depth performs better than the single best one (0.7926 vs. 0.788) and rank average performs exactly the same way on the public leaderboard but slightly worse on the local validation. Bagging improved result of this modest even further so I decided to use such composition of bagging plus rank averaging of three XGBoosts with a different maximum depth of the trees. I ended up using three instances of such blocks trained on different features.

2) Additional handcrafted features: I extracted a few more features from the provided game states based on my knowledge of the game and intuition.

- player.hand.hp calculated as sum of minion health only. For some reason in baseline features this sum included durability of weapons in hand.
- opponent.played.taunts
- opponent.played.total_taunts_hp
- opponent.played.max_taunts_hp
- opponent.played.poisonous
- player.played.taunts
- player.played.total_taunts_hp
- player.played.max_taunts_hp
- player.played.poisonous
- opponent.played.possible_attack - sum of attack for minions with flag "can_attack"
- player.played.possible_attack
- player.card_advantage as difference betwen number of cards left in player deck and in opponent deck.
- player.hand.nOfPlayableSpells
- player.hand.nOfPlayableMinions
- player.hand.nOfPlayableWeapons
- opponent.total_weapon_damage

$$\begin{aligned} opponent.total\_weapon\_damage = \\ opponent.hero.attack \\ \cdot opponent.hero.weapon\_durability \end{aligned}$$

- player.total_weapon_damage

Minions with taunt ability basically serve as additional hero health so the next group of features is linear combinations of hero health, total attack of enemy minions and total health of player minions with taunts. These features were supposed to help tree-based and we can see that their importance is quite high.

- player.max_hp_danger

$$\begin{aligned} player.max\_hp\_danger = \\ player.hp \\ + player.armor \\ + player.played.total\_taunts\_hp \\ - opponent.played.attack \end{aligned}$$

- player.current_hp_danger

$$\begin{aligned} player.current\_hp\_danger = \\ player.hp \\ + player.armor \\ + player.played.total\_taunts\_hp \\ - opponent.played.possible\_attack \end{aligned}$$

- opponent.max_hp_danger

$$\begin{aligned} opponent.max\_hp\_danger = \\ opponent.hp \\ + opponent.armor \\ + opponent.played.total\_taunts\_hp \\ - player.played.attack \end{aligned}$$

- opponent.current_hp_danger

$$\begin{aligned} opponent.current\_hp\_danger = \\ opponent.hp + opponent.armor \\ + opponent.played.total\_taunts\_hp \\ - player.played.possible\_attack \end{aligned}$$

I also calculated WOE (Weight of Evidence) for hero classes. For categorical feature f, object x and target label y

$$WOE(f)(x) = P\left(y(x) = 1 | f(x)\right).$$

WOE can be calculated for a set of categorical variables by considering all their combinations as separate categories. Basically $WOE(player.hero\_card\_id)$ is a win rate of player class, $WOE(opponent.hero\_card\_id)$ is a loss rate of opponent class and $WOE(opponent.hero\_card\_id, player.hero\_card\_id)$ is a probability that player class will win against opponent class. WOE can be estimated from dataset X using the formula

$$WOE(f)(x) = \frac{K \cdot mean + \alpha \cdot global\_mean}{\alpha + K}$$

where

$$\begin{aligned} K &= |\{i | i \in X \& y(i) = 1 \& f(x) = f(i)\}| \\ mean &= \frac{K}{|\{i | i \in X \& f(x) = f(i)\}|} \\ global\_mean &= \frac{|\{i | i \in X \& y(i) = 1\}|}{|X|} \end{aligned}$$

and α is a regularization parameter. I used $\alpha = 50$. In the result for each category we obtain a single WOE feature in contrast with multiple features produced by one hot encoding. This single feature is highly informative and tree-based models can utilize it effectively. This technique is well known among Kaggle [12] participants and was popularized in the Russian data science community by Stanislav Semenov.

- $player.class\_score = WOE(player.hero\_card\_id)$
- $opponent.class\_score = WOE(opponent.hero\_card\_id)$
- $class\_pair\_score = WOE(opponent.hero\_card\_id, player.hero\_card\_id)$

With addition of this features, I trained three groups of models. First of all the second ensemble of XGBoost described above. Then a group of linear models was added. Due to data size, I used stochastic gradient descent for training and bagging with result averaging to diminish randomness of the final prediction. Lastly, I added a random forest because it is usually a safe bet to try it.

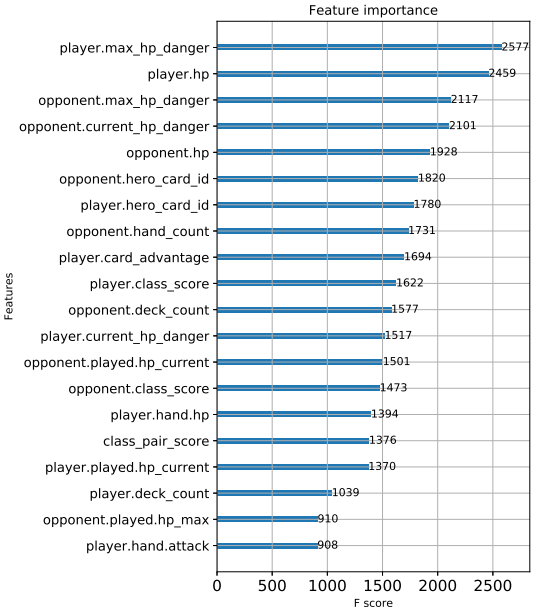

Fig. 1. The 20 most important features from one of XGBoost models. We can see many handcrafted features at the top.

3) WOE features: All features listed above can be treated as categorical and I calculated WOE for all of them. For example $player.hp$ is an integer feature with values between 1 and 30 so we naturally get at most 30 categories. Using this features I trained the third block of XGBoosts.

4) Logarithmed features: Sometimes linear models perform better on logarithmed data. So I calculated $\log(1 + x)$ for positive-valued features and replaced each of other features with two new ones: $\log(max(x, 0)+1)$ and $\log(max(-x, 0)+1)$. Using these new features I trained a group of linear models in the way described earlier. Because of the good result of these models on the leaderboard, I also trained a number of multilayer perceptrons with a different structure using the same features. There is no point in training separate XGBoost using this features because logarithm cannot change the ordering of feature values.

F. Final composition

At the end, I averaged ranks of all models from the previous steps. I also added two particular groups of XGBoost models directly to the final average because of the low correlation between their predictions and results of the corresponding XGBoost blocks.

III. Conclusion

During this work I constructed a number of features, used them for training multiple models and combined predictions of these models in order to improve quality of prediction over a single model. Almost all described methods can be applied to an arbitrary classification problem. Averaging method is simple enough to be safe from overfitting but requires manual selection of uncorrelated classifiers. It allowed to build model

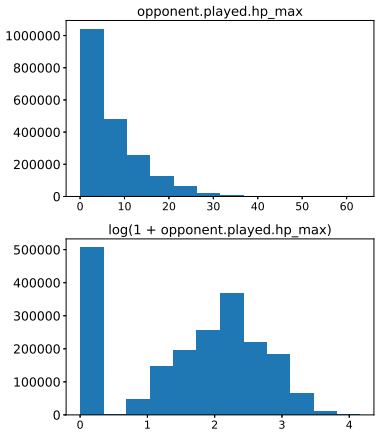

Fig. 2. Histograms of the feature "opponent.played.hp_max" before and after the logarithm transformation.

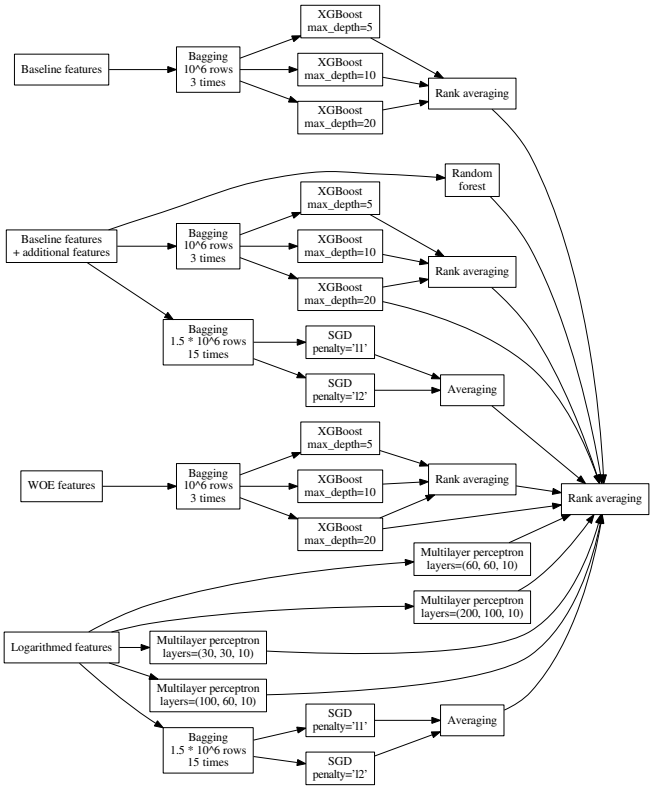

Fig. 3. Final composition scheme

that performs well even without reliable way of local validation. Problem-specific feature engineering also improved final score.

REFERENCES

[1] "Playing Atari With Deep Reinforcement Learning" Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, NIPS Deep Learning Workshop, 2013.
[2] H. Cho, K. Kim, S. Cho, Replay-based Strategy Prediction and Build Order Adaptation for StarCraft AI Bots, IEEE CIG, 2013.
[3] M. Stanescu, S. Hernandez, G. Erickson, R. Greiner, M. Buro, Predicting Army Combat Outcomes in StarCraft, AAAI AIIDE, 2013.
[4] Y. N. Ravari, S. Bakkes, P. Spronck, StarCraft Winner Prediction, AAAI AIIDE, 2016.
[5] K. Conley and D. Perry, "How Does He Saw Me? A Recommendation Engine for Picking Heroes in Dota 2", tech. rep., 2013.
[6] Kalyanaraman (2014). "To win or not to win? A prediction model to determine the outcome of a DotA2 match". https://cseweb.ucsd.edu/~jmcauley/cse255/reports/wi15/Kaushik_Kalyanaraman.pdf
[7] Du, Xin, Jinjian Zhai, and Koupin Lv. "Algorithm Trading Using Q-Learning and Recurrent Reinforcement Learning." CS229, n.d. Web. 15 Dec. 2016
[8] Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S. (2016). Collective robot reinforcement learning with distributed asynchronous guided policy search. ArXiv e-prints
[9] Tom Fawcett. 2006. An introduction to ROC analysis. Pattern Recogn. Lett. 27, 8 (June 2006), 861-874. doi:10.1016/j.patrec.2005.10.010
[10] Michael L. Littman. 2001. Value-function reinforcement learning in Markov games. Cogn. Syst. Res. 2, 1 (April 2001), 55-66. doi:10.1016/S1389-0417(01)00015-8
[11] Game guide, http://us.battle.net/hearthstone/en/game-guide/
[12] Kaggle, https://www.kaggle.com/
[13] Kaggle Ensembling Guide, https://mlwave.com/kaggle-ensembling-guide/
[14] Breiman, L. Machine Learning (1996) 24: 123. doi:10.1023/A:1018054314350
[15] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. Preprint.
[16] Elie Bursztein, "I am a legend: Hacking Hearthstone using statistical learning methods" https://cdn.elie.net/publications/i-am-a-legend-hacking-hearthstone-using-statistical-learning-methods.pdf
[17] Battle.net end user License Agreement http://us.blizzard.com/en-us/company/legal/eula.html