Proceedings of the Federated Conference on Computer Science and Information Systems pp. 127–130 ISSN 2300-5963 ACSIS, Vol. 11 DOI: 10.15439/2017F563

Predicting Unpredictable: Building Models Handling Non-IID Data, A Hearthstone Case Study

Dominik Deja
Polish-Japanese Academy of Information Technology
Warsaw, Poland
dominik.deja@pjwstk.edu.pl

***Abstract*—The following article is created as a result of the *AAIA'17 Data Mining Challenge: Helping AI to Play* Hearthstone. The Challenge goal was to correctly predict which bot would win a bot-vs-bot Hearthstone match based on what was known at the given time. Hearthstone is an online two-players card game with imperfect information (unlike chess and go, and like poker), where the goal of one player is to defeat their opponent by decreasing their "life points" to zero (while not allowing the opponent to do the same to oneself). Two main challenges were present: the transformation of hierarchically structured data into a two-dimensional matrix, and dealing with Non-IID data (certain cards were present only in test data). A way of how to successfully cope with those complications while using state-of-the-art machine learning algorithms (e.g. *Microsoft's* LightGBM) is presented.**

I. Introduction

A. Hearthstone

DEVELOPED and published by *Blizzard Entertainment*, *Hearthstone* (initially *Heartstone: Heroes of Warcraft*) is a free-to-play turn-based online collectible card video game. It was released on March 11, 2014, and it is available for Windows, Mac, iPad, Android and Windows 8 tablets, as well as iOS and Android mobile phones[1].

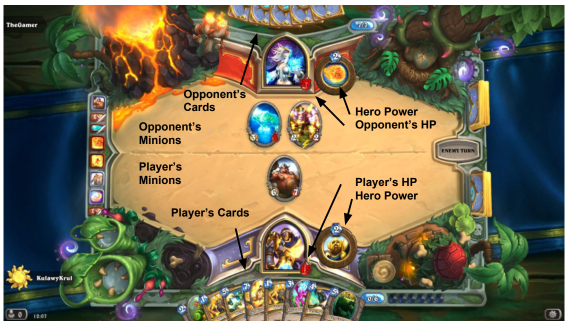

Fig. 1. Screenshot from an actual game

The game is a turn-based card game between two players (naming convention: "player", "opponent" will be used in this article), using constructed decks of thirty cards along with a selected hero. Each hero posseses a unique power which allows them to either draw a card, summon a minion, heal or deal damage. Furthermore, usable cards differ for each hero. For example, Mage class offers more spells, while Paladin has access to stronger minions. Players use their limited mana crystals (indicated by a hexagon, at the bottom right for the player in Figure 1, and top right counter for the opponent) to cast spells or summon minions to attack the opponent, with the goal to reduce the opponent's health to zero. Each spell has a unique effect such as: dealing damage to one or more minions, dealing damage to champion(s), changing the statistics of minions or champions, etc. Each minion can deal a certain amount of damage (indicated at the bottom left of its card), has a certain amount of hp (indicated at the bottom right of its card), posseses additional features (such as "windfury" enabling it to attack twice per turn, "charge" enabling it to attack the same turn it was cast, or "taunt" which makes a minion a priority target for the enemy's attacks), and can cast additional effects depending on other circumstances.

While the mechanics of the game is rather simple, a high number of available cards (by 2017, there are over 1000[2]), a wide range of possible, often unique traits possessed by each minion, and imperfect information (the player does not see the opponent's cards, decks are randomly shuffled, and random effects are common) increase the complexity of the game[3].

This complexity makes it a perfect case study for AI experts to try out new methods and approaches.

B. Contest

The *AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone* was a data mining competition organized by Silver Bullet Solutions and the Polish Information Processing Society (PTI) within the framework of the International Symposium Advances in Artificial Intelligence and Applications[4].

The goal was to predict a binary outcome (win/lose) of bot-versus-bot Heartstone matches. The cost function used for evaluating the participants predictions was AUC (Area Under Curve). There was a two-step score evaluation. First, AUC scores based on a fixed 5% of test data were provided for each contestant's set of predictions. Then, after the contestants

[1]http://us.battle.net/hearthstone/en/

[2]http://hearthstone.gamepedia.com/Card

[3]https://en.wikipedia.org/wiki/Hearthstone_(video_game)

[4]https://fedcsis.org/2017/aaia

shared their reports, a final leaderboard (based on the whole test set) was provided.

For data preprocessing, the author used Python 2.7 (IPython Notebooks). For the rest of this work R version 3.3.3 (RStudio) was used. The author used Windows 8.1 Pro, Intel i7-4710MQ 2.50 GHz (4 cores), 32GB RAM, NVIDIA GeForce GTX 870M.

II. Data Processing

Data for this contest was generated by Peter Shih's Hearthstone simulator[5]. From each match, random snapshots were taken, aggregated, and saved as JSON files. The creators provided two datasets a training dataset and test one(2000000 and 750000 snapshots respectively) formatted as multiple JSON files.

For each game, short overall statistics, player and opponent statistics, statistics on cards played (by both player and opponent) and cards at hand (only for player) were provided. For the training set, 90 unique cards (78 at hand, and 42 played), and 12853295 cards in total (8996725 at hand, and 3856570 played) were used. Cards at hand are the ones owned by the player which they can cast (in case of spells), or summon (in case of minions). Cards played are minions summoned and still living. Interestingly, there are 38 new unique cards in the test set (38 at hand and 22 played). In total, 642985 (417607 at hand, and 225378 played) out of 5500047 (3264847 at hand, and 1592215 played) cards in the test sets are new. This mean, that 11.69% of cards present in test set are new, and they are present (to various extents) in 415793 out of 750000 games (55.44%) played using the test set.

The fact that over 55% of observations from the test set contained new cards dismantled the assumption of identical distributions of data and played an important role in data processing and modeling.

A. From Attribute-Value to Matrix Format and Feature Engineering

In order to construct a two-dimensional matrix, where each row is a snapshot and each column is a different feature, JSONs were processed one by one.

First, all the statistics on each game and participant were extracted (final outcome, turn, participant's hero type, hp left, armor, crystals left, crystals in total, #cards at hand, #cards played, etc). This produced 26 columns.

Then, the counts of the players cards at hand were saved in separate columns (one for each card type). For the cards played, as they consist of minions only, the sums of their hp were saved in unique columns (per minion type and player/opponent). The rationale behind it is that the minion's health can be changed by both participants during a match and it highly impacts how much influence a minion will have on the outcome of a game. This produced 162 columns.

In order to overcome the fact that new, unseen cards were present in the test set, features based on aggregates were added. They included respectively for each participant's minions overall hp and attack situation: max, min, sum, product, mean, median, and counts for the minions special characteristics, such as ”charge”, ”taunt”, or ”freeze”, additional features such as ”max damage doable to an enemy in this turn” amongst them. Because a player usually has more minions that can be summoned than they can afford to summon, a knapsack problem was solved in order to find an optimal configuration of minions to summon in order to maximize damage done to an enemy champion (the ”taunt” trait was taken into an account in its simplified form - instead of solving an optimization problem of finding the best way on how to attack minions and then the hero, the ”taunt” minions hp was subtracted from the maximum damage doable to an enemy in the same turn). Unsurprisingly, this feature came out to be one of the strongest predictors (all models agreed on this) of the final outcome. Yet, it wasn't sufficient to simply check whether a player can decrease an opponent hp to values equal to or less than zero (in the same turn), as the relation between the game status and the outcome turned out to be more complex (or bots are not as smart as we would like them to be). This produced 42 columns.

Depending on a run, the cards from the test set which were non-present in a training set, where either mapped to their closest neighbour (using Euclidean distance on their crystal cost, hp, and attack for hand, as well as current hp and current attack for played card), or were omitted. Additionally, a couple of diffs were provided (player hp - opponent hp, maximum potential damage to enemy - enemy hp, and so on).

B. Final preparation

Since constructing this many features results in introducing collinearity into data, additional measures were taken to minimize the negative consequences of feature engineering.

Thus, constant features, highly correlated ones (> 0.95), and those which could be presented as a linear combination of others were deleted. This resulted in 241+1 final variables used for training models. Finally, the data was scaled in order to improve the efficiency of algorithms (especially logistic regression with regularization).

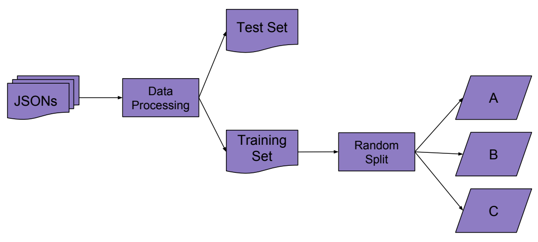

Fig. 2. Final Model

In order to obtain reliable results and avoid overfitting, training data was split into three parts (as shown on Figure 2):

- A : First layer data (1500000 observations)

[5]https://github.com/peter1591/hearthstone-ai

- B : Second layer data (400000 observations)
- C : Internal test data (100000 observations)

Splitting it into three parts (instead of the usual training, and validation set) helped to train a more complex, two-layered model.

III. Modeling

Amid various methods of improving the overall efficiency of modeling, such as obtaining good understanding of data (via solid explanatory analysis done before applying actual models), feature engineering, or adjusting models to directly optimize cost function, one of the most common and important is stacking. It is based on the observation that combining predictions of several good, uncorrelated models will result in an even stronger prediction [1].

A. Initial Two-Layered Model

Therefore, the first model used consisted of a number of machine learning algorithms stacked in two layers. All models are shortly described in Table I. The pipeline was as follows: first, each of the algorithms from the first layer was trained on part A of the training data and provided predictions for part B and part C. Then, using first layer predictions for part B as an input, the LightGBM model from the second layer was trained and provided predictions for part C. Predictive power was compared between each single algorithm, and the overall model using part C.

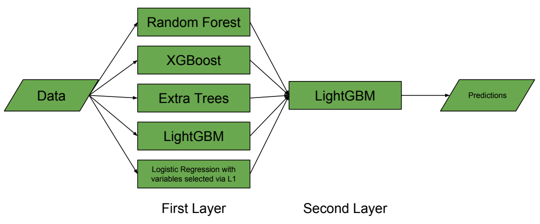

Fig. 3. Typical Two Layered Stack Model

Since most of the algorithms used have a number of parameters that need to be optimized, a Bayesian approach was used to estimate them as proposed by Snoek, Larochelle, and Adams [2]. In short, parameter tuning can be seen as a Bayesian optimization problem, in which a model's performance is modelled as a sample from a Gaussian process. Since the posterior distribution induced by the Gaussian process is traceable, we can efficiently use information from past runs to optimally choose new parameters worth trying.

As training a number of models tends to take a lot of time, the author first checked for optimal ranges of parameters using a small subset of part A of training data (from 10000 to 100000 observations), and then ran it on bigger chunks (up to 1000000 observations). A Bayesian optimization package in R as proposed by [2], usually takes less than 30 iterations (in case of 5 or less parameters).

TABLE I
Models used for stacking

Algorithm	Specification
Random Forest	One of the best tree-based algorithms(using bagging)as invented and implemented by Breiman [3]. Run two times. The first time, it was trained on whole data, the second time it was trained only on 100 top features (their importance was asserted by the first model). This allows to reduce noise and increase prediction power. ntree = 1000. The number of features used in the second forest can also be optimized.
XGBoost	A still fairly new, tree-based algorithm (using boosting) created by Tianqi Chen [4]. Basic parameters to be optimized: *eta, colsample_bytree, subsample, max_depth, min_child_weight*.
Extra Trees	Extremely Randomized Trees, as proposed by Geurts, Ernst, and Wehenkel [5].
LightGBM	One of the newest and strongest algorithms accessible via R. *Microsofts* LightGBM as a part of their Distributed Machine Learning Toolkit[6]. Alike XGBoost it's a tree-based boosting algorithm with multiple parameters to be optimized. Using bins instead of vectors, and optimizing the code, LightGBM is one of the fastest and strongest algorithms available.
Logistic Regression with variables selected via L1	Logistic regression trained on variables selected by logistic regression with L1 penalization.

B. Dealing with Non-IID Data

Since test data is Non-IDD, and over 55% of observations contain new cards, the more optimized the parameters are, the worse the score obtained on the test set is.

From a statistical learning theory point of view, it can be shown as a bias-variance trade-off dilemma [1]. The more we try to optimize our models, the more their complexity and variance will increase, and thus, their sensitivity to any shifts in data distribution. By finding the right set of parameters, one may decrease the bias and improve the models effectiveness, but it is always done at the price of the models stability.

Tables II, and III show this phenomenon using LightGBM algorithm trained on 1000000 observations from part A (learning rate = 1). As the number of leaves increases, the model becomes more complicated. As the number of minimal observations required in each leaf increases, the model is pushed to be simpler. Here, as it is shown only for illustrative purposes, data used for obtaining scores comes from part B, and C (for iid data), and from the test set (for non-iid data).

TABLE II
AUC for IID Data

#Leaves	Min #Observations in Leaf: 1	10	100	1000	10000
10000	0.9073	0.8594	0.9395		
1000	0.8624	0.8119	0.8883	0.9414	
100	0.8231	0.8100	0.8434	0.9181†	0.8915†
10	0.8044	0.8001	0.8146	0.8483†	0.8331†

†Algorithm didn't converge in 3000 iterations and could be further trained.

Two main things are interesting here. The first one is that parameter optimization is a non-convex problem. While a model with 10000 leaves is over 0.9 AUC for both, 1 and 100 minimal observations in a leaf, for 10 it drops to 0.86 AUC.

The second one is that the more optimized the model is, the more sensitive it becomes, and that different parameters indicate a different "threat" to the stability of the model. While a minimal number of observations in leaf seems not to be strongly linked a to model's sensitivity, the number of leaves plays a major role in making a model fine-tuned.

TABLE III
AUC FOR NON-IID DATA

#Leaves	1	10	100	1000	10000
10000	0.7261	0.709	0.7537		
1000	0.7016	0.7293	0.7262	0.7439	
100	0.7347	0.7471	0.7361	0.7354†	0.7401†
10	0.7754	0.7807	0.7667	0.7482†	0.7631†
	Min #Observations in Leaf				

†Algorithm didn't converge in 3000 iterations and could be further trained.

C. Final Conditional Model

This knowledge was used to build a final model. In fact, two models were built, and depending on whether new cards were present in any given snapshot, the adequate model was used. For snapshots where IID seemed to hold (no new cards present), a fine-tuned stacked model was used for predicting. For observations where data was non-IID, a conservative model was run to provide predictions.

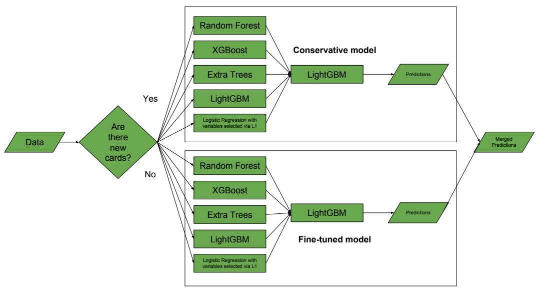

Fig. 4. Final Model

The internal test set obtained 0.967 AUC. The test set managed to obtain a stable 0.795 AUC score (the best solution obtained was 0.802 AUC).

D. Additional Possibilities

Thankfully, each data science contest has a deadline. Without one, there would be always a new hypothesis to test. Here as well, a number of things can be tried:

- Instead of using minion hp only, doubling (or tripling) the number of columns to add information on attack and count
- Using deep neural networks
- Adding information on what cards an opponent probably has at hand
- Adding more features based on player/opponent possible decisions
- Checking the lot's logic - why some games where a player can win in one turn are not won
- Further optimization of the models' parameters
- Defining and constructing a card space where each card has its representation so that one does not have to rely on actual cards

IV. CONCLUSION

Even five years ago, many scientists predicted that we are still a decade from constructing an AI capable of winning a Go game with an average professional. Yet, in March 2016 *AlphaGo*, an AI developed by *Google DeepMind* beat Lee Sedol, one of the best players worldwide, 4:1 [6].

Since then, and due to the renaissance of deep neural networks, AI research got a second breath. Currently, the effort is to create a bot capable of playing more human-like games (usually in real time) with imperfect information (at the time of this article being published, games like Doom [7] are probably already conquered, therefore Starcraft would be a good example of AI researchers' next target).

As part of this ongoing effort, this article helps to sort out what the most practical approach is to structuring non-relational data into a form usable for machine learning, and how to cope with non-IID data (or when a shift in feature distributions is expected to happen).

REFERENCES

[1] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2001, vol. 1.
[2] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.
[3] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[4] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.
[5] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.
[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[7] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," *arXiv preprint arXiv:1609.05521*, 2016.