Proceedings of the Federated Conference on Computer Science and Information Systems pp. 143–148
DOI: 10.15439/2017F567
ISSN 2300-5963 ACSIS, Vol. 11

Use of Domain Knowledge and Feature Engineering in Helping AI to Play Hearthstone

Przemysław Przybyszewski, Szymon Dziewiątkowski, Sebastian Jaszczur, Mateusz Śmiech, Marcin Szczuka
Faculty of Mathematics, Informatics and Mechanics, The University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
Email: {pp332493,sd359113,sj359674,ms361208}@students.mimuw.edu.pl, szczuka@mimuw.edu.pl

***Abstract*—This paper describes two approaches to the AAIA'17 Data Mining Challenge. Both approaches are making extensive use of domain/background knowledge about the game to build better representation of classification problem by engineering new features. With newly constructed attributes both approaches resort to Artificial Neural Networks (ANN) to construct classification model. The resulting solutions are effective and meaningful.**

I. Introduction

The past three years saw an increased interest in online collectible card games. One of the most popular games of this type is Hearthstone® created by Blizzard Entertainment [1]. Because of its popularity and simplicity the game has gained attention in the streaming and e-sport community. Although the most popular game modes involve competition between two human players, it's possible to play against AI. Unfortunately, the AI is no match for a skilled player. Hence the question of improving computer players arose. In order to achieve the best results, players have to be able to judge possible outcomes, predict consequences of their actions, take into account random factors (after all, it's a card game and the randomness is embedded into its core), and much more. The decisions can be reduced to classifying possible plays or game states as good or bad and these predictions may then guide the player to choose the best possible play. In connection with the International Symposium on Advances in Artificial Intelligence and Applications (AAIA'17) a competition was organised with use of KnowledgePit platform [2]. The goal of this competition – called AAIA'17 Data Mining Challenge – was to estimate the state of the game and decide which player has a higher chance of winning.

Hearthstone is a turn based card game where the goal is to reduce opponent hero's health to zero. Players may play *spells*, which have an instant effect or *minions* that remain on the *battlefield*. Cards are played from hand and a new card is drawn from deck at the beginning of each turn. The deck consists of 30 cards (see Figure I) and each card may be present in at most two copies. Hand size is limited to 10 and board size (number of minions present at any point) is limited to 7 for each player. The enemy hero may be damaged using minions and some spells. Minions have certain *attack* and *health*. All cards have cost which is paid in *mana crystals*. At the beginning of each turn the current player refreshes all their previous mana crystals (thus regaining mana they spend in the previous turn) and gets a new one with the upper limit of 10 mana crystals per player.

The input data represented various states from a large collection of games played between AI players. Game states consist of both players' health, armour, spell damage bonus, remaining deck size, number of mana crystals, hero class and a brief description of each card on hand (current player only) and on the battlefield (both players). This description contained card's name, cost, and its attack, health, and some special traits, if applicable. Because some card effects weren't described, we had to resort to card databases and our knowledge of the game to ensure that all aspects are covered.

The overall goal in the competition was to predict which of two players has a higher chance of winning in the given state of the game. The problem was an example of a binary classification. Solutions were compared by measuring the Area Under Curve (AUC) for the Receiver Operating Characteristic (ROC) curve [3]. This metric allows for interpretation of classification model results in terms of trade-off between true positive and false positive rates. The metric was chosen so that false positives would be eliminated as they have a potentially disastrous effect on the outcome - incorrectly classifying a bad play as a good one may result in choosing that play and losing in the final outcome.

This article describes some of the features used by the authors in their submissions to the competition. Two different approaches are outlined - one employed by team consisting of Szymon Dziewiątkowski, Sebastian Jaszczur, and Mateusz Śmiech (team "jaszczur") and the other created by Przemysław Przybyszewski (team "pp332493"). Both teams independently developed interesting features and the end product - the classifier - was somewhat alike in both cases as it was based on an Artificial Neural Network (ANN). The similarities may be seen in utilisation of domain knowledge to estimate card value (or usefulness) to allow for more precise classification. The final score for team "jaszczur" was AUC of 0.7930 which placed them 35th place and AUC of 0.7950 for team "pp332493" which came 18th.

This paper describes solutions developed by both teams. First, the data processing and feature engineering on which both teams put a particular emphasis in their work is described (Section II). Then there is a description of models used in final solutions along with other tested approaches (Section III)and the most interesting findings and conclusions (Section IV).

Fig. 1. Three examples of Hearthstone® cards. Left to right: Acidic Swamp Ooze, Flamestrike and Core Hound. The information contained in each card:
Acidic Swamp Ooze - the lower left corner contains Attack (3) and the lower right corner contains Health (2). Presence of these two traits indicate that it's a minion. The 2 in the blue hexagon in the upper left corner indicates mana cost of the card. The Battlecry is an effect that takes place immediately before the minions enters the battlefield after being played from hand.
Flamestrike - lack of Attack and Health indicate that it's a spell. Its devastating effect takes place immediately after casting it. The blue edge indicates that it's a class card, available only to Mages, as opposed to grey-bordered cards which are neutral and available to all classes.
Core Hound – 7 mana, 9 Attack, 5 Health minion that is additionally a Beast – a minion subtype. Cards belonging to the same subtype usually synergise well with each other.
Hearthstone® card designs are property of Blizzard Entertainment.

II. Feature Engineering

The contestants were given two distinct forms of data. The first and more basic one was raw data in JSON format, where an entry was a single game state of a particular game. The organisers processed and aggregated this data in tabular Comma Separated Value (CSV) format. The CSV set consisted of 45 attributes, which are shown in the Table I below (their names are rather self-explanatory).

The training dataset contained 3,150,000 labeled samples. Organisers also provided 750,000 unlabeled samples, of which 5% was used for validation visible to participants during the competition and the remaining 95% was used for final evaluation

Although the CSV dataset contained several useful attributes, the JSON format proved to be much more informative, a quality that was exploited by both teams in their solutions. When connected with domain knowledge, that raw data was especially helpful in gaining an advantage in the competition.

The following subsections show the two approaches to construction of new, meaningful features as applied by the respective teams.

A. Approach of the team "jaszczur"

The majority of work was devoted to the feature engineering based on raw JSON data. The aim was to use those features later on in an Artificial Neural Network (ANN), therefore only numeric features were investigated. From the input data 796 input columns were extracted, corresponding to 29 different features. A single column represents a single value of a feature.

The major limitation of this approach is the number of columns which had to be constant for every sample. Unfortunately, this doesn't reflect the structure of game states, as those can have a variable number of cards on the table or in hand. To take care of that a maximal possible number of columns per feature was introduced. For example there were 14 columns created for a feature describing minions' attack on the battlefield, as there is an in-game limit of 14 minions on the table, 7 for each player.

Some values were undefined. This happened mostly in cases when a feature was applied to multiple cards (e.g. attack of each minion on the battlefield). In such cases an additional column was added for every possible undefined column. This new column took binary values and represented whether the corresponding value was defined. All undefined values were then replaced with zeroes.

There are cases with multiple possible options with no obvious numeric interpretation, like Hero Class. In such cases an adequate number of binary features was defined, e.g., 18 columns for Hero Classes, one for each player/class pair, with two corresponding features set as 1.

Tiers and values: One of Hearthstone's game modes is Arena. It differs from Constructed mode profoundly, mostly in terms of strategies and decks that yield best possible outcomes. These two modes share the card set, of which Standard (cards

TABLE I
ATTRIBUTE NAMES IN THE TABULAR PART OF DATA SETS.

gamestate_id	decision	turn	opponent.armor	opponent.attack
opponent.hero_card_id	opponent.weapon_durability	opponent.special_skill_used	opponent.hp	opponent.crystals_all
opponent.crystals_current	opponent.deck_count	opponent.fatigue_damage	opponent.hand_count	player.crystals_all
player.armor	player.attack	player.hero_card_id	player.hp	player.special_skill_used
player.weapon_durability	opponent.played_minions_count	player.crystals_current	player.deck_count	player.fatigue_damage
player.hand_count	player.played_minions_count	opponent.played.nOfCards	opponent.played.attack	player.hand.nOfPlayable
opponent.played.hp_current	opponent.played.hp_max	player.played.nOfCards	player.played.attack	player.played.crystals_cost
player.played.hp_current	player.played.hp_max	player.hand.nOfMinions	player.hand.nOfSpells	player.hand.nOfWeapons
player.hand.nOfCards	opponent.played.crystals_cost	player.hand.attack	player.hand.crystals_cost	player.hand.hp

used in the competition) is a small (ca. 16%) subset. In Arena instead of creating the deck from all available cards, players draft cards, i.e., the system chooses three random cards, of which the player selects one that will be added to the final deck. This process is repeated 30 times, resulting in the same deck size as in Constructed. Because of this deck-making scheme, Arena is generally considered less synergy-oriented and more value-oriented than Constructed.

Although this is very situation-specific, some cards are generally considered superior (see Figure I). There are numerous tools that help players gauge the value of cards in Arena, one of which is Lightforge [4]. The fundamental differences between Arena and Constructed wane in the light of AI's weak style of play. Having analysed Lightforge's tier juxtaposition, it was decided to utilise it as it is - without adapting it to Constructed.

Lightforge divides the cards into the following tiers: *Great, Good, Above Average, Average, Below Average, Bad,* and *Terrible*. They are referred to as tier categories. Furthermore, it also assigns numerical values to cards so that they can be compared within tiers. They are referred to as *tier values*.

Fore example, for the cards presented in Figure I:

- Acidic Swamp Ooze – tier *Good*. Base stats are standard for its cost but on entering the battlefield it irrevocably destroys the opponent's weapon which costs anywhere from 1 to 5 mana.
- Flamestrike – tier *Great*. It is a costly spell available only for Mages. Serves as an Area of Effect (AoE) card capable of clearing the board. If used properly it is very likely to turn the tide for the player who uses it.
- Core Hound – tier *Bad*. Despite the high Attack, this minion's value is considerably low because of its minute Health.

Description of attributes used by "jaszczur": The 29 constructed attributes were put on the list and ordered with respect to accuracy achieved by Logistic Regression that learned only on that attribute on 750,000 samples with test to validation ratio of 80:20. All features except those annotated with an asterisk (*) were computed for both players. The annotated ones were created only for the current player because of the limited information available (i.e. we don't know the opponent's hand). There is also a brief explanation of choosing specific features based on authors' Hearthstone experience. The final list of constructed attributes is as follows:

1) 0.6569 - Number of cards on the battlefield that with certain cost. Basic set consists of cards of cost between 0 and 8 and 10 (there aren't any cards costing 9 or more than 10). Usually the higher the cost, the better and more impactful the card is, and the more substantial threat it poses.
2) 0.6558 - Battlefield state - a single column for every possible card (133 uncollectible and 17 collectible) containing number of copies of this minion on the battlefield. Some cards, like Stormwind Champion or Healing Totem have an additional effect that was not included in the short description the input data provided but have a veritable influence on the game. It was tried to create features tailored to single cards (e.g., Stormwind Champion has greater impact if the board consists of more minions), but initial results were disappointing - the accuracy of 5 of those features combined together was negligibly higher than random guessing.
3) 0.6558 - Sum of costs of all minions on the battlefield.
4) 0.6512 - Sum of attack of all minions on the battlefield.
5) 0.6508 - Attack of every single minion on the battlefield, along with information whether or not it is present.
6) 0.6453 - Tier value of each card on the battlefield.
7) 0.6432 - Tier category of each card on the battlefield.
8) 0.6403 - Number of minions present on the battlefield.
9) 0.6396 - Health of each minion on the battlefield. Because minions with 0 health die instantly, there was no need to create additional columns to indicate their presence or absence.
10) 0.6307 - Sum and average number of received damage of all minions on the battlefield.
11) 0.6227 - Health points of heroes with added armour.
12) 0.6193 - Health points of heroes after using hero power.
13) 0.6153* - Tier category of each card on hand.
14) 0.5828 - Number of cards in deck, on hand and on the battlefield. For prolonged games this number often represents the advantage a player has. Generally, forcing the opponent to trade multiple cards for your one card gives you an advantage later on. This feature quantifies this advantage in a simple manner but only in the context of opponent's total number of cards.
15) 0.5385 - Number of special traits (Windfury, Taunt, Divine Shield etc.) that minions on the battlefield have. These traits help guard the hero and valuable minions

(Taunt), make favourable trades (Divine Shield) or put more pressure on the enemy hero (Windfury). Aggregating them yielded better results because of their rarity - it's uncommon for multiple Divine Shield minions to be present on the board at the same time whereas any two traits are much more likely to occur.

16) 0.5359* - Hand state - a single column for every possible card containing number of copies of this card on hand. It is similar to board state but also includes spells. Some Area of Effect spells (Holy Nova, Flamestrike, Consecration) or so-called hard removal spells may turn the tide for the casting player.
17) 0.5252* - Number of cards costing X on hand, for every possible value of X existing in the data.
18) 0.5251 - Number of cards on hand.
19) 0.5207* - Tier value of each card on hand.
20) 0.5124* - Sum of costs of all cards on hand.
21) 0.5094* - Sum of damage from spells that can affect the enemy hero (Fireball, Holy Nova, Kill Command etc.) and bonus spell damage (Dalaran Mage, Kobold Geomancer etc.).
22) 0.5066 - Fatigue damage.
23) 0.5062 - Number of special traits minions on hand have - Charge and Battlecry in addition to those considered by similar feature regarding the battlefield.
24) 0.5062* - The approximate number of possible plays in the current turn, depending on cost of all cards, available mana crystals and number of possible targets. Usually the more choices a player has, the more likely there is a good (or even an outstanding) one.
25) 0.5049 - Attack and durability of the currently equipped weapon.
26) 0.5017 - Hero class (Mage, Warlock, Shaman etc.).

The remaining three (out of 29) constructed attributes were irrelevant and therefore omitted. Logistic regression ran on the features mentioned above gave a result of 0.7830 AUC.

B. Approach of the team "pp332493"

The majority of work of "pp332493" was also spent on the feature engineering part. Apart from the data provided by competition organisers external data from the website HearthPwn [5] was also used. On this website a ranking of decks can be found. Each entry in this ranking consists of deck name, deck type, mana, class, rating, viewcount, comments, and cost. The higher the evaluation score for a deck the higher its chance to win. By knowing the deck name, one could find all the cards it consists of and their respective attributes. Of the deck-related information gained this way only part of the card features, such as mana cost and deck rating, were actually utilised. They were used to generate the 'average card strength' attribute. Only records having rating higher or equal than 60 were taken into account. This value was chosen because decks with this rating were among the 0.5% best decks ever evaluated on this site. As there is no full information about the whole deck of the player and the opponent the approach was to estimate the value of 'strength' attribute for each minion. These estimates were further summed up in order to get the average strength of the player's and the opponent's played cards as well as the player's hand. Computing the 'strength' attribute for a single minion card was conducted in the following manner:

1) Iterate over all decks and retrieve all minions that were present in those decks.
2) Iterate over all minions. For each minion create temporary variables 'power_sum' and 'power_count' and iterate over all decks. If this minion is in the given deck, then compute its share in a given deck. This share can be represented by the ratio of crystal cost of the minion to the sum of crystals costs of all cards multiplied by the number of appearances of this minion in the deck. Then add the number of occurrences of this minion in the deck to 'power_count' variable and its percentage share multiplied by the deck evaluation to the 'power_sum' variable.
3) The power of a single minion is computed by dividing 'power_sum' by 'power_count'.
4) In cases of minions in the training data set, that were not found on the ranking page [5], the median of all computed strengths was taken instead.

Other features that were used in the classification were derived from the data provided for the competition (both tabular and JSON).

Description of attributes used by "pp332493": Both provided datasets were used to generate the training data for chosen models. In the feature selection phase attributes first the identifiers that carry no specific information were removed. Those are: 'gamestate_id', 'opponent.hero_card_id', 'player.hero_card_id'. The rest of the variables, apart from the decision, is used to create the training data set, as all of them seem to have an influence on the decision value. In the end 41 variables are chosen from this the original 45 in tabular part of data (see Table I).

After checking the part of the data in the JSON format it was noticed, that there are some influential variables that are not present in the tabular data. Those are the special features of a single minion card, such as: Taunt, Charge, Stealth, Freezing, Shield, Poisonous, and Windfury. Sum of each special ability for player's hand and cards played by both player and the opponent are also added to the training data. Additionally, a dedicated variable called 'special' was created. 'special' is derived as a sum of all aggregated special traits. Value of this attribute can be understood as an expression of interaction between the features. Moreover, pair of attributes called 'able_to_perform' are generated for both the player and the opponent. Those are binary variables indicating whether a player (or opponent) is able to perform at least one move using the cards that are already on the table. The aforementioned 'average strength' attribute is added to training sample for player's hand and cards played by both adversaries. Features taken both directly from the original data and engineered make the final data set that contains 70 attributes.

III. Classification Models

Both teams experimented with various classification models and at the end selected the one giving the best results on their engineered attributes. Chosen models are described below, followed by a brief description of alternative approaches that were investigated for the purpose.

A. Final model of the team "jaszczur"

The best model for constructed attributes turned out to be an Artificial Neural Network [6]. It had 796 inputs nodes, all of which were corresponding to numeric variables that were normalised to have mean of zero and standard deviation of 1. The network consisted of four fully-connected, hidden layers - with 100 neurons in the first and 50 neurons in each of the subsequent hidden layers. The output layer had a single neuron with sigmoid activation.

Neurons in hidden layers used Rectified Linear Unit(ReLU) [7] as the activation function. There was also batch normalisation (see [8]) between each pair of adjacent layers. The overall network error (loss) was calculated as cross-entropy, with batch size of 100 and Adam optimiser [9].

Contestants were provided with 3,150,000 labelled samples, but some of those were from the same games (but different turns). Unfortunately, there was no information about which samples were coming from which game. Because of this, after splitting labelled data randomly into training and validation set, the samples from the same game could go into both sets, what caused the model to recognise specific games instead of general patterns. Because test data was extracted from a completely different set of games, model's result on the validation data turned out to be a really bad predictor of result on test data. The authors were unable to mitigate this problem with training/validation split, so it was decided to use each labelled sample in training only once (that is, there was a single training epoch) to prevent overfitting (recognising specific games). This approach also yielded the best result on 5% of test data available during competition.

Neither L1 nor L2 regularisation yielded better results. That is probably because each sample was shown only once and because batch normalisation already regularises the network.

The model was implemented in Python/Keras [10] with TensorFlow backend [11]. First prototypes and some of data processing were done using scikit-learn module [12].

Training this model took about 20-30 min. to train, out of which at least 15 were devoted to loading the data. It was run on GPUs and total RAM used was about 40GB. The final score for neural network based on constructed attributes (see Subsection II-A) on the test set was AUC of 0.7930 which placed this solution on 35th position.

B. Final model of the team "pp332493"

The chosen model belongs to the class of ensemble-based classifiers, which is a soft voting classifier consisting of three other classifiers. First of them is logistic regression with the default value for the inverse of the regularisation strength. Second one is an ANN with ReLU activation function and three hidden layers with 25,15, and 5 neurons, respectively. Last but not least is an ANN with logistic activation function and three hidden layers arranged as previously (25,15,5 neurons). Both ANN models used the Adam optimiser, the L2 regularisation term 0.0001, and the initial learning rate of 0.001. The attempt to run an exhaustive search over a range of possible values for the regularisation terms for those three models was made but finally abandoned due to excessive computational cost. Additionally, to better suit ANN training, the data was normalised by removing the mean and scaling to unit variance which is recommended as a measure to better fit the network model.

The model selection was based on the principle of getting the AUC score as high as possible at the same time minimising its standard deviation. The estimation of the score were derived using the 10-fold cross-validation on the training data set. The score for the finally chosen model was AUC of 0.79652 with deviation of 0.01283. The final model was trained on the training data set, which consisted of two million observations. Predictions made on the test set, which consisted of 750,000 observations, allowed to achieve the AUC score of 0.79508952, which is a bit below the validation result. The discrepancy between validation and actual testing result may suggest that the chosen model was also a bit overfitted.

Model construction and data processing were done in Python using the scikit-learn module [12]. Training of this model took up to 30 min., two-thirds of which was spent on loading the data. It was run on the quad-core CPU and total RAM used was about 16GB. The final score for this classifier based on constructed attributes (see Subsection II-B) on the test set was AUC of 0.7950 which placed this solution on 18th position.

C. Other classification models tested

Both teams have tested several classification algorithms before arriving at the final solution presented above. The alternative classifiers checked were mostly drawn from the tool box of the scikit-learn [12] library for Python and associated tools.

Team "jaszczur" has tried to use scikit-learn methods such as: K Nearest Neighbours (k-NN), Support Vector Machines (SVM), Decision Trees, Random Forest, and Extra Trees. Unfortunately, k-NN and SVM models could not compare with ANN and tree-based classifiers in this case.

The majority of testing was related to scikit-learn's methods RandomForestClassifier and ExtraTreesClassifier. Both of them resulted in overfitting in addition to excessive memory (RAM) consumption. The best Random Forest solution trained on 1,500,000 examples with maximal tree depth of 23 consisted of 20 trees and achieved AUC = 0.7589. Higher depths show an increase in accuracy and AUC on our validation data, but with a simultaneous decrease in AUC on organisers' test data. Members of the "jaszczur" team were unable to tweak parameters of scikit-learn's DecisionTreeRegressor to yield result above 0.71 AUC.

Alternative models that were investigated and tested by "pp332493" team included Logistic Regression, AdaBoost, Extra Trees, and Artificial Neural Networks with higher number of hidden neurons in each layer than the finally chosen one. Additionally, a Voting Classifier – which consisted of a mixture of previously mentioned ones – was trained and validated. Most of these alternative models were not classifying the data well enough compared to the finally chosen solution, often not reaching 0.79 for AUC in the cross-validation phase. Only the ANN with more hidden neurons yielded the value of AUC metrics nearing 0.8 on 10-fold cross-validation. Unfortunately, checking this model on the provided test set clearly demonstrated that it was significantly overfitted.

IV. Interesting Findings and Conclusions

Some results from the previous section (Section III) are non-trivial and may indicate interesting aspects of the game. Below are the most probable consequences and authors' explanations of the aforementioned results.

- Features derived from hand state have on average 0.12 less accuracy than their equivalents derived from battle-field state. This is due to the fact that the cards that have been played and are unaffected by summoning sickness (inability to attack in turn they are played) have more measurable impact on the game than cards that are yet to be played because of mana limitations.
- It's widely recognised that some classes have favourable, neutral, and unfavourable matchups. For example Hunters are more likely to lose to Priests and Warriors (due to their healing ability) – unfavourable matchup, but more likely to win with Warlocks (due to their utilising life as a resource) – favourable matchup. Using just class options for both heroes yielded 0.5017 accuracy – negligibly more than random guessing. This shows that either Basic set is well-balanced or that the AI didn't utilise the class they played to its full potential.
- Features revolving around card tier or value yielded scored on average 0.01 worse than those based solely on mana cost. Given that a lot of experienced players agree that the tiers and values we employed are a good estimate of card usefulness, this indicates that cost of cards is well-balanced and reflects their strength.
- Minions' attack is slightly more significant than their health. Experienced Hearthstone players disagree with this result. Health is a little bit more important as it potentially allows for multiple trades and card advantage. This result shows that the AI played too aggressively and could probably be improved by making more trades instead of prioritising damaging the enemy hero.
- Spells have considerably lower impact on game than minions. There are two possible explanations for this result. Although the total number of spells is similar to the total number of minions, all spells are class-dependent. Because there are nine classes, the number of spells to choose from after a class has been selected is an order of magnitude lower than the number of available minions. Additionally the variance in usefulness of spells is higher, resulting in very few good spells for each class. This, in turn, affects their participation in deck (spell to minion ratio) meaning high number of samples with features corresponding to those spells set to zero (no such spell drawn). Another explanation is that the provided AI didn't learn how to utilise these spells to their full potential.

Given the final assessment of models on test dataset it can be said that they generalise quite well. The difference between the cross-validated AUC score and that computed on the test dataset was almost negligible. It is apparent that feature engineering and domain knowledge played a major role in the final outcome and that they greatly improved the solutions. Although the differences were profound, the non-obvious affinity led to similar conclusions and results. Leaving implementation details aside, resorting to the opinion of the game community which was shared between the teams led to significantly better outcome and some interesting findings. Thus, it appears mandatory for designers of AI in games, Hearthstone® being just a good example, to factor-in the knowledge accumulated by gamers' communities.

References

[1] "Hearthstone official game site," http://us.battle.net/hearthstone/en/.

[2] A. Janusz, D. Ślęzak, S. Stawicki, and M. Rosiak, "Knowledge Pit - a data challenge platform," in *Proceedings of the 24th International Workshop on Concurrency, Specification and Programming, Rzeszów, Poland, September 28-30, 2015.*, ser. CEUR Workshop Proceedings, vol. 1492. CEUR-WS.org, 2015, pp. 191–195. [Online]. Available: https://knowledgepit.fedcsis.org/

[3] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. doi: 10.1016/j.patrec.2005.10.010

[4] "LightForge – Hearthstone Arena tier list," http://thelightforge.com/TierList.

[5] "HearthPWN – Hearthstone database, deck builder, news, and more!" http://www.hearthpwn.com/.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[7] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. doi: 10.1038/nature14539

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://jmlr.org/proceedings/papers/v37/ioffe15.html

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[10] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195