Mobile game (Clash Royale) match result prediction using multiple machine learning algorithms

Zheng Li[*], **Yiming Wang**[+] and **Fangwen Wu**[+]
Civl and Environmental Engineering Department, UIUC[*]
Electrical and Computer Engineering Department, UIUC[+]

Abstract

Clash Royale is a worldwide popular card-based strategy mobile game. It consists of more than 70 battle cards and allows the player to choose 8 cards as the battle deck. Different cards combination and distinctive play style bring the complexity and outcome of the game. In this project, multiple algorithms are implemented using 180,000 top 1000 players' real matches. Among all experiments, we found that AdaBoost over decision tree achieves the highest accuracy of 63.7%. The results show that we have reached a fairly reasonable result and it also shows that it is difficult to find a cards combination that dominates the game.

1 Introduction

Clash Royale[1] is a worldwide popular real-time card-based strategy mobile game. It won the best mobile game of the year 2016 and has more than 70 million downloads on App Store. The game consists of building decks by choosing 8 out of 75 battle cards and battle with opponents in real-time. Cards are characterized by their types (troop, building, spell), rarities (common, rare, epic, legendary) as well as levels. Given the different combination of cards and its fast-paced mechanism, it brings complex card interactions and outcome of the game.

Given the popularity and complexity of the game, we can propose a research question by asking does it exist a deck that performs better than another one? Does the cards combination itself provides enough information for us to tell whether it will win against its opponent? Before we dig further, we acknowledged that human plays a very important role in this game and the same deck with different play skills will yield significant contrasting outcomes. In this project, we neglect the effect of human factors by selecting worldwide top 1000 players match records as our training data. Because all those players are good at the game and therefore the skill difference is relatively subtle. In addition, top 1000 players always have maximum level cards which eliminate any bias caused by cards level differences. Thus, we are able to exclude unrelated factors (human skills, card levels, etc) and focus on the combination of cards that determines the match outcome: win, lose and draw. In order to simplify this task, we exclude draw scenario and only consider matches that result in win or loss.

In this project, we will be focusing on the following aspects:

1. Define and find features and label space.
2. Propose algorithms given the defined feature and label
3. Implement proposed algorithms and do a comparative study

Detailed algorithms selection and implementation will be discussed in section 2 and section 3.

2 Methodology[3]

2.1 Feature matrix

One important part of this project is to generate a feature matrix to represent the data. The rows of the

feature matrix represent the examples we have and the features are in columns. All the elements in the feature matrix are in binary. We need to convert the features with continuous values, such as the average card elixir cost to the nearest integer and then represent them in binary.

2.2 Algorithms

The goal of this project is to predict which player is more likely to win the game given the cards they use. Hence, this is a binary classification problem (win and lose). The methods that commonly used for binary classification are decision trees, random forests, AdaBoost, support vector machines, neural networks and Naive Bayes. In this project, we implement these algorithms and evaluate their performance in our data set. A brief summary of the algorithms used in this project is listed below.

2.3 SVM

SVM is a linear maximum margin classifier which can be represented by the following object function:

$$\begin{aligned} \textbf{min} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{j=1}^{m}\xi_i \\ \text{s.t } y^{(i)}(\mathbf{w}\cdot\mathbf{x}^{(i)}+\theta) \geq 1-\xi_i \\ \xi_i \geq 0, \forall(\mathbf{x}^{(i)}, y^{(i)}) \in D \end{aligned}$$

In addition, we can choose a different kernel to blow up the feature space and apply SVM in the new feature space to separate the data set that is not linearly separable in the original feature space. The kernels that can possibly be used in our project are the polynomial and RBF kernels.

2.4 Adaboost

AdaBoost is a boosting algorithm which requires choosing a distribution over the data, learning a weak hypothesis based on the data as drawn from that distribution and choosing a new distribution such that mistakes are weighted more heavily from which a new weak hypothesis can be learned. This process can be repeated for any arbitrary number of weak learners.

2.5 Neural networks

The neural network is a non-linear classifier and a neural network with one hidden layer together with one input and one output layer is able to represent any functions. Since the data in our case may not be linearly separable, the neural network is likely to outperform the linear classifiers. Parameter tuning is an important step in the training phase and the weights updates during the training process are achieved by the back propagation algorithm.

2.6 Decision tree

Decision tree is a hierarchical data structure that represents data through a divide and conquers strategy. Decision tree is a nonlinear classifier and it worth noting that decision tree always gives the local optimum.

2.7 Random forests

Random forest is a bagging algorithm which we draw a number of bootstrap samples of data and draw the sample of available attributes at each split, train trees on each attribute and get a number of trees and average prediction of trees on out of bag sample.

2.8 Naive bayes

Naive Bayes is a generative learning algorithm. It learns the most probable hypothesis by calculates the posterior probability of all possible classifications. It exploits the assumptions of the conditional independence that each feature value is conditionally independent, given the label. This assumption is applicable in this context.

3 Implementation and optimization

3.1 Data preparation and collection

StatsRoyale (*https://statsroyale.com*)[2] is the website that provides real-time match records of the entire game. So training and testing match records are retrieved from a website called *StatsRoyale* where top 1000 players's match records are updated on daily basis.

Then we use *BeautifulSoup* package to retrieve match records from the website based on the record of top 1000 players' profile. We collect data at the end of each game season (2 weeks per season) because the end of a season is the time when players use different decks and try their best to win. So the data could potentially be more expressive. So

far more than 180,000 match records have been collected from the website.

3.2 Feature generation

We generate feature based on our background knowledge of this game. First, we list all the cards and their attributes including the type, the rarity and the cost to use them and store them into an excel file. We sort the cards in the alphabetic order and each card is assigned a unique id. There are four aspects that may affect the match result.

3.3 Average card elixir cost

The stronger the cards, the more elixir they will require to play, so the average elixir cost reflects how strong your card. It is worth-noted that elixir regeneration is the same both players, so the average cost of a deck reflects how frequent they can use the card. Consequently, we are able to differentiate decks by group them based on average elixir cost.

3.4 Cards rarity

The more rarity of the card, the stronger and harder to collect will it be. So this feature also reveals how strong a user is. We show the number of cards for each rarity (common, rare, epic, legendary).

3.5 Cards class

Each card has a class (troop, spell, building) and each class has their function. Some for defend and some for offend. The better combination of card class will indicate that a user is more balanced. We show the number of cards in each class for this feature.

3.6 Card composition

This is the most direct aspect that may affect the final result. What users choose and what their opponents choose is important. If a certain card is used, then this column will be 1, otherwise, it will be 0.

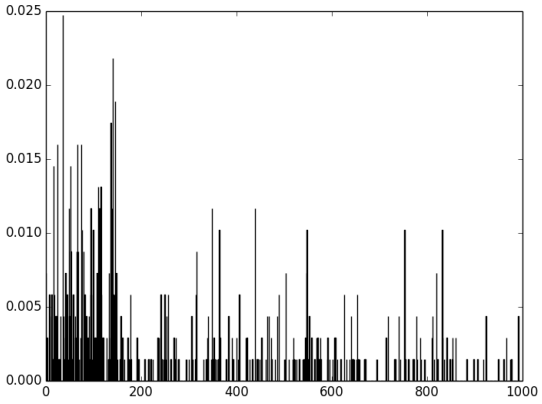
Next step we use the data mining algorithm, *Apriori*, to find the frequent cards combination from battles. There must be some reason for the combination like they may compensate the weakness of each other or some for attack and some for defense. By setting a different minimum threshold, we can find which are the most frequent card combinations from all match history. For example, if we set the threshold to 200, we can get 717 frequent patterns. Below is the screenshot of frequent pattern with their frequent.

```
1074 2 [fire_fireball minion]
974 2 [fire_fireball hog_rider]
819 2 [minion order_volley]
808 2 [giant minion]
747 2 [building_inferno hog_rider]
741 2 [minion musketeer]
736 2 [hog_rider musketeer]
735 2 [hog_rider minion]
725 2 [minion skeleton_horde]
717 2 [minion skeletons]
```

After combining all features, we can get a matrix with 1003 columns and over 180,000 rows. We save this matrix in a document.

3.7 Feature selection

Our feature matrix is really huge which may result in not only the dramatically slow speed of learning but also the overfitting like issues. So we need to find a way to reduce the dimension of our matrix. We use *xgboost* package to make feature analysis and selection. We first draw the feature importance plot to see which feature matters most.



In the plot, first 148 columns are cards composition, then comes the average cost, cards type, and cards rarity. The frequent pattern follows at the end. From the plot, we can find out that the cards composition and cards type contribute heavier to the final result than other features. And the average cost, cards rarity and frequent pattern also have a certain weight.

However, for the sake of speed and not encounter overfitting issues, we regenerate the feature matrix.

We use *xgboost* to find the 150 most important features for further learning.

3.8 algorithm implementation

For this part, we use *scikit-learn* package for algorithm implementation. We divide the whole dataset into two parts. 80% for training and 20% for testing. We tried different algorithms. The result is shown in the following table.

Algor Name	Accuracy
Random Forest	0.597
SVM	0.6324
Adaboost	0.6256
Logistic Regression	0.6279
Naive Bayes	0.5961
Neural Network	0.5778
Adaboost+Decision Tree	0.632

From the table, we can see that the average accuracy of all algorithms is about 60%. The best algorithm that fit this model is *SVM*. Then comes *Adaboost+Decision Tree* and *Logistic Regression*. However, the accuracy gap between algorithms is not that large and the prediction accuracy for such binary-class classification problem need to be further improved.

4 Discussion

This project is a binary-class classification problem. So if we make an arbitrary guess, the probability is 50%. Our result is above this probability to some extend, around 60%, but it is not very satisfying. We think it may be due to the following aspects.

4.1 Item selection

As this is one of the most important aspects that contribute to the success of the final accuracy, we need to choose proper and valuable features to generate the feature matrix. In this project, we generate the feature matrix based on our knowledge of the game. However, this knowledge may not be correct for prediction. So it may limit the improvement of prediction accuracy.

4.2 Amount of data

If the data amount is not enough, the noise may be large and the variance of prediction may be unstable. We only have 180,000 battle records now. This may not be large enough for a better prediction. So If we have chance to get more data, we may make a better prediction.

4.3 Game design

This mobile game is popular and we believe that game designer hope the game can be fair and not let some players be too strong to unbeatable. So in order to do this, they may change the ability of cards or introduce more features to decrease the gap of winner rate of strong players and weak players. Therefore, the prediction accuracy is may not be stable under such circumstances.

4.4 Other issues

Other issues like algorithm selection, parameter setting for different algorithms may also affect the prediction accuracy of the model. We need to conduct more experiment if we want to achieve a better result.

5 Conclusion

In this project, we make match result prediction for a mobile game, Clash Royale. We focused on top 1,000 players and generate features like cards composition, average elixir cost, cards frequent pattern, etc. We also use feature selection methods to decrease dimension. We use multiple algorithms to fulfill the supervised learning. The result reveals best prediction accuracy is about 63.2% using SVM. This result is relative good. There may have several aspects that limit this accuracy and several aspects that can improve this prediction accuracy. We need to conduct more experiments to achieve a better result.

6 Future work

For the future work, we can consider from the following three aspects:

6.1 Extend to multi-class classification

Now, we exclude tie battles, so it's a binary classification. However, if we want to better analyze the game, we should take this situation into consideration. We should extend the project to a multi-class classification problem.

6.2 Better feature extraction

Feature extraction is one of the most important steps in machine learning projects. There are still a lot of possible valuable features that we haven't considered. So for future work, we may consider features like cards sequence in a battle, cards ability change, popular cards deck, etc. Besides, we may generate player based features that sort and manage data according to user's battle history.

6.3 More application extension

Since this mobile game is popular nowadays, we may create web applications to show our result and attract users. We may connect our program with database and create the application like recommendation system.

7 Reference

[1] Lehtonen M J, Harviainen J T. Mobile games and player communities: Designing for and with clans[J]. Design Management Review, 2016, 27(3): 20-26.

[2] Stats Royale website: https://statsroyale.com/top/players

[3] UIUC CS446 lecture notes *http://l2r.cs.uiuc.edu/ danr/Teaching/CS446-17/*