Big Data Analytics
Report
Mobile Game Data Analytics: Clash Royale

Sergio Cristiá & Ken Oshiro

I. Introduction

Using the results of millions of ranked matches, we trained a model predicting the win/loss of a game based on the player's deck of cards.
It was an interesting experiment to see if the deck building part of the game was important enough to allow an accurate prediction of the match's results.

Clash Royale is a free to play game in which players compose their deck of 8 cards out of more than 100. Each card spawns a particular unit or spell. A player wins by destroying the enemy castle.

Because of the size of the dataset we used CSC through Valohai.

Summary of technologies used:

- Puhti
- Pouta/Allas
- Filezilla
- Cyberduck
- Jupyter Notebooks
- Visual Studio Code
- Github
- Valohai

II. Dataset

The dataset has been acquired from Kaggle. It consists of the results of 37 millions ranked matches from the mobile game Clash Royale (21 Gb uncompressed).
The format is csv and the collection took place from December 2020 to January 2021.

The data contains the following:

- Players information such as name, rank, clan.
- Deck information: 8 cards ID, cards level, number of card types (troops/spells/siege), number of each rarities (common to legendary)

- Match information: arena, duration, remaining hitpoints of buildings, average elixir.

III. Workflow

The raw data is located in Kaggle in the following link. The dataset is composed of a total of ten csv files, accounting for a total of 20Gb. The data was downloaded first locally and then to Puhti via Filezilla.

Having chosen Valohai as the tool to preprocess the data and train our model, it is required the use of one of their compatible data storage services. In our case, our choice was the OpenStack Store Object or Swift since it is readily available in CSC via Pouta as an Allas container.

The first option was to push the data from Puhti via command line. Seemed to be working reasonably well however, we ultimately carried the operation via Cyberduck. The reason behind is that CSC created an empty object in the Allas container of destination as well as duplicating the container. That seemed off at the time and so Cyberduck seemed more reliable.

With the data in place, the only thing left was to prepare the Valohai project. We decided to carry steps in Valohai: preprocess and training.

- Preprocess: fetch the csv, merge them into one, extract features and labels and save them as intermediate files (numpy .npy files).
- Training: fetch features and labels using Valohai datum URIs and train the model. The model can be saved through Valohai back to Allas.

IV. Pre-processing

We chose the following features:

- 8 Cards ID forming the deck
- Total cards level
- Number of each cards rarity (from common to legendary)
- Number of troop cards
- Number of spells cards
- Number of siege units cards

The label was win/loss, which was respectively 1 and 0.
Due to the nature of the dataset, we had to split each match into two, assigning 1 for the winner and its deck and 0 for the loser and its deck. We then split the data into training and testing.

V. Exploratory Data Analysis

We ran locally a sample of 20k data points on a notebook to see if the model would be of any value.
With a simple K Nearest Neighbours, the accuracy was 0.68.
A gridsearch found that the best model was SVC with a C of 1 and Gamma of 10.

VI. Modelling

The modeling work was carried out in Valohai. With the help of the tutorial example from class, we created the Valohai yaml configuration file and the other scripts. The goal was to develop a somewhat organized piece of code split in two parts, one for the preprocessing and another one for the training.

First, we tried to continue with the SVC model. After noticing it would take a very long time to train a SCV model that big, we decided to try both KNN and RandomForest models. The results of the modeling were not very successful with an accuracy of around 55%. We tried cross-validation but Valohai, for some reason, was not able to converge and we stopped the calculation after some time. The low score value of the model could be due to several reasons of which we could highlight:

- The features are not suitable predictors of whether the match will be won or lost.
- The model is inappropriate and/or requires hyperparameter optimization.

In addition to the Valohai configuration files and scripts, we created our own docker image to be able to install all the requirements our project needed.

VII. Code

The code can be found in https://github.com/sergiocristia/clahroyale.git.

VIII. Final thoughts

The documentation about CSC was very good and helpful. For example, pushing data to or from Allas was well explained and easy, as well as how to use Cyberduck and other additional services.

The experience with Valohai was bittersweet. Valohai is a great and powerful tool and we are sure it helps to standardize the MLOps process of training models and putting those on production. Yet, the documentation is not as rich and the process of using Valohai is not trivial. We believe we could have spent a good amount of time on improving the model, instead setting up Valohai turned out to be quite time consuming.

All and all, it was a rich and valuable experience and sure the efforts were rewarding.