Alexander Palomba
Artificial Intelligence
Dr. Rivas
Final Project Writeup
12 December 2016

Alexander Palomba – Game Algorithm

Abstract

Many consumers purchase games either based on critical reception alone or based on the "hype" surrounding the game at its release, only to have all but abandoned the game after a few weeks or even days of play time. The goal of this project is to produce an algorithm that can reliably determine if a user should purchase a game based not only on the game's critical merits but also based on the user's purchase history. After testing on a library of 65 games, the algorithm proves moderately effective, however, additional development and research are needed to prime the algorithm for practical application.

Introduction

If you've been playing video games for long enough, it's likely you've accrued at least a few games that you later regretted purchasing and which are now collecting dust on a shelf or in a box somewhere. In the modern game market, there is a great deal of pressure on the consumer to buy a game knowing very little about it. Pre-order bonuses, review embargoes, and misleading pre-release advertisements all contribute to this. After a time, a gamer may get to the point where they are no longer willing to gamble $60+ on a game that may or may not even be worth their time.

The goal of this project is to create an algorithm that can predict will decent accuracy whether or not a user should purchase a new game. It will not only take into account the game's critical scores, as those can sometimes be misleading, but also a comparison to the user's game library. Games that are more similar to those that the user owns and plays frequently will receive higher final scores, because it

is assumed that the user will enjoy games of those genre even if there are games from other genres with higher scores.

Method

Step One: Acquiring Data

In order to properly construct an algorithm for new games, it would be necessary to first construct a library of games to simulate what the user may have already purchased. Ideally, an algorithm such as this would have several hundreds or even thousands of games' worth of data to train with. Although Steam Libraries tend to vary in size, they typically contain anywhere from a handful of games to a few hundred at most, depending on how often the user plays video games and how often they are able to purchase games to add to their library.

The training set used for this algorithm is based on an existing library of 65 games. While this does provide a more realistic example of a semi-casual gamer for the algorithm, less data will likely mean less accurate results.

Once the library is assembled, each game in the library is categorized by genre: Action, Adventure, Casual, Indie, Massively Multiplayer, Racing, RPG, Simulation, Sports, or Strategy. After all games have been given a genre, they will be scored based on how many other games of that genre are present within the library and on how many hours those games have been played out of the user's total play time. The games will also receive marks based on the ratings given on Metacritic.com – critical score, user score, and publisher score.

So each game within the library is weighed based on the user's library, and based on three separate review scores. For the purpose of training the dataset, each game within the library is scored based on the average of these scores. The algorithm is designed to output a number between 1 and 4,

with 1 indicating that the user should not purchase the game, 2 indicating that the user should explore other alternative games, 3 indicating that the user should only purchase the game at reduced price, and 4 indicating that the user can confidently purchase the game. How the games in the library scored was based on their average. The games all averaged between 3 and 7 during preliminary scoring, so the final score was determined by making games that averaged below a 4 received a 1 for their final score, games that averaged between 4 and 5 received a 2, games that averaged between a 5 and 6 received a 3, and games that averaged above a 6 received a 4.

Step Two: The Neural Network

When a user starts up the algorithm, they are asked for the genre of the game in question, and given a list of options (the ten categories listed above). Based on the genre given, the algorithm will look through the library to find the percentage of pre-owned games in that genre, and the percentage of hours played in that genre. These percentages are used to generate the first two scores. Then, the user is prompted to enter the game's Metacritic scores – critical score, user score, and publisher score. Not all games will have sufficient critical reception to accommodate all three scores, so if there is no data in any of these three categories, the user will be asked to enter a 5, indicating an average score.

The algorithm then needs to access the library once again, separating the data into training sets and target output. This data is put into an MLP (multi-layer perceptron) regressor, which determines what final score the game should receive based on the final scores of the other games in the library. The score it outputs is a number between 1 and 4; and for simplicity's sake, the number is rounded to the nearest integer for the final result (though the non-rounded result is still shown, so the user gets an idea of how close to making/not making the cut the game was).

Experiments

The original concept for the algorithm involved using an SQL search in order to accurately acquire the first two scores. However, this approach was dropped when it became apparent that there would not be enough time to implement it. Instead, a pseudo switch-case method using if statements was used to assign values to the first two scores based on the fact that the programmer knew what those scores should be (since it is assumed that games of the same genre will have the same scores for these categories).

```
if gameGenre == 0:
    genreScore = 3.7
    hourScore = 4.4
elif gameGenre == 1:
    genreScore = 1.1
    hourScore = 0.7
elif gameGenre == 2:
    genreScore = 1.2
    hourScore = 0.7
elif gameGenre == 3:
    genreScore = 0.9
    hourScore = 0.2
elif gameGenre == 4:
    genreScore = 0
    hourScore = 0
elif gameGenre == 5:
    genreScore = 0
    hourScore = 0
elif gameGenre == 6:
    genreScore = 1.1
    hourScore = 1.1
elif gameGenre == 7:
    genreScore = 0.6
    hourScore = 0.2
elif gameGenre == 8:
    genreScore = 0.2
    hourScore = 0
elif gameGenre == 9:
    genreScore = 1.2
    hourScore = 2.6
else:
    genreScore = 3 #average genreScore value
    hourScore = 2 #average hourScore value
```

During testing, it was determined that a neural network of about 20 neurons produced the best, most consistent results. This is after testing several different amounts of neurons between 3 and 200. However, in every case, the algorithm would occasionally produce results that had ridiculously low testing scores, going as low as -34.

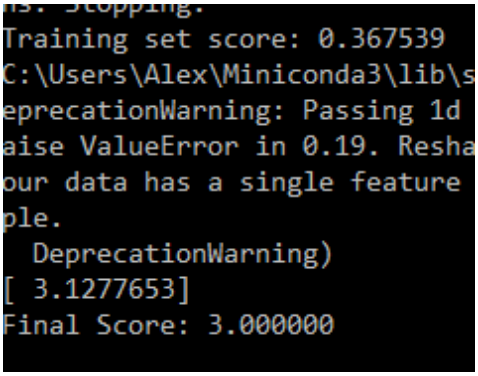
```
Training loss did not improve mo
hs. Stopping.
Training set score: -5.089771
C:\Users\Alex\Miniconda3\lib\sit
eprecationWarning: Passing 1d ar
aise ValueError in 0.19. Reshape
our data has a single feature or
ple.
  DeprecationWarning)
[ 0.59805171]
```

The solution to the problem was to construct a loop that would continually execute the testing until the algorithm reached a result that scored positively. After implementing this, the majority of future tests scores averaged around 0.7, with a few occasional outliers as low as 0.2.

```
87    mlp.fit(X, y)
88    while(mlp.score(X,y) < 0):
89        mlp.fit(X,y)
```

```
Iteration 4, loss = 0.20721805
Iteration 5, loss = 6.28226985
Iteration 6, loss = 10.39353243
Iteration 7, loss = 8.67476585
Training loss did not improve more than
hs. Stopping.
Iteration 1, loss = 0.16642650
Iteration 2, loss = 80.31401956
Iteration 3, loss = 6.50663939
Iteration 4, loss = 11.00647615
Training loss did not improve more than
hs. Stopping.
Iteration 1, loss = 3.57488126
Iteration 2, loss = 9.77828035
Iteration 3, loss = 1.18951486
Iteration 4, loss = 1.97528851
Iteration 5, loss = 2.13662439
Iteration 6, loss = 0.32526532
Iteration 7, loss = 0.82074887
Iteration 8, loss = 0.17761098
Iteration 9, loss = 0.62057789
Iteration 10, loss = 0.18452847
Iteration 11, loss = 0.10589204
Iteration 12, loss = 0.16124771
Iteration 13, loss = 0.13280148
Iteration 14, loss = 0.09196213
Iteration 15, loss = 0.11382043
Iteration 16, loss = 0.14028907
Iteration 17, loss = 0.12065506
Training loss did not improve more than
hs. Stopping.
Training set score: 0.778276
C:\Users\Alex\Miniconda3\lib\site-packag
eprecationWarning: Passing 1d arrays as
aise ValueError in 0.19. Reshape your da
our data has a single feature or X.resha
ple.
  DeprecationWarning)
[ 2.68786247]
```

In order to further correct for this, another final step was added to the algorithm to have it round the prediction result to the nearest integer (1, 2, 3, or 4).



After testing this version of the algorithm with examples taken from the library (whose target scores were known), the algorithm proved relatively efficient at providing dependable results.

Conclusions

The algorithm, in its current state, is a very basic and inherently flawed product. Despite the fact that it was able to consistently produce the desired results in testing, there are a large number of ways it can be improved upon in the future.

Firstly, the algorithm relies on the fact that the user library, and all the data within, is pre-constructed. The user has to manually input all of their games into a spreadsheet (two, actually), manually look up their scores on Metacritic, and manually figure out the percentages for the library-based scores. A much more complete version of the algorithm would be able to query this data automatically from the user's Steam library and attain the necessary reviews from Metacritic by means of web scraping.

Secondly, the algorithm cannot be properly trained with only one library's worth of data. In order to properly ensure consistently reliable outputs with minimal outliers, thousands of games would be required for training. Even for those who professionally play and/or review games for a living, such a library is not realistically feasible. To accommodate for this, it is possible that a different method of

machine learning would be more optimal, such as a random forest, but additional time would be needed to explore such options.

References

http://www.metacritic.com/game – source for critical reception data

https://www.python.org/ – source for Python package info

http://stackoverflow.com/ – troubleshooting info

http://machinelearning.org/icml.html – references for write-up format

Dr. Pablo Rivera-Rivas – general information, troubleshooting, and assistance with core concepts