What are strong factors predicting prices in steam applications

Benjamin Gerritsen s1019202 (1), Nikolai Schauer s1017181 (2)

(1) Radboud University, Artificial Intelligence, Nijmegen, 6533XJ, Netherlands
(2) Radboud University, Artificial Intelligence, Nijmegen, 6537DD, Netherlands

Abstract

For consumers it is important to know how much one would pay for a product of a certain type. Additionally for companies it may be important to know how they might have to adjust the prices of their applications in order to fit their industries standards. In this paper we look at certain openly available attributes of steam games/applications to determine what the usual price of a certain application would be. This allows the user to judge whether he is paying too much or whether it is a good bargain. Companies on the contrary can adjust the prices of their games if they find that similar applications cost more or less money or can predict a good price for their game given certain other predictable attributes.

In order to do this we use a Multi-Layer-Perceptron with input attributes being average number of downloads, with and without average hours played, median playtime, total time played respectively and genres.

Additionally we use clustering on the genres to find stereotypical genre-price combinations and double check the MLP's results.

Previous Work

Unfortunately not many scientific reports go specifically into this branch, as it is a rather competitive industry and research is conducted by companies that would probably want to keep their advantageous information secret. However are there various informative blog's available which inspect larger datasets and infer plausible trends. A couple are referenced below.

"Using Steam data to tell you if your game will sink or swim" (Michael Treneny, June 28, 2017, venturebeats) is an analysis of games and what determines their success. The author analyses 4200 games on attributes like price, release date, player base and genre and infers that multiplayer games, 3D games and open world games tend to sell the best. (1)

“Predicting video game hits with Machine Learning” by Ignacio Chavarria on the webpage “towards data science” analyses a multitude of game attributes such as developer, critic and release year using machine learning and visualizes his results and other data metrics. (2)

Data:

Our data is a precleaned dataset of 27.000 steam applications and 18 attributes by Nick Davis from kaggle. However most of these are games. It was last updated 7 months ago, thus being fairly recent. Their attributes include name, release date, user ratings, genres, approximate number of downloads and more. The dataset is provided as a CSV-file.
A link to the kaggle page is provided in the appendix.

Preprocessing

Preprocessing of the data was not necessary except for two instances; genre and approximate number of downloads. Genre is a string in which all genres of a game are stored in, separated by a semi-colon. This needs to be converted to a numerical representation to make it classifier friendly. Each genre will become a separate dimension, with either value 1 for “is present” or 0 for “is not present”. Approximate number of downloads is also a string and gives a range, similar to “100000-150000”. We split the string on the minus and take the average of the two numbers, in this case 125000.

MLP-Classifier approach:

Attributes

We picked a couple attributes to run the Multilayer-Perceptron classifier on. These are the approximate number of downloads (the dataset only provides a rough estimate) of a game, the average hours played of the game, the genre, positive user ratings and negative user ratings. We also trained the classifier with combinations of these attributes, namely average hours played and downloads, genres and positive and negative user ratings. This should give us a diverse foundation and will allow us to find patterns easily. The best way to predict the price of a game can be determined by the lowest error rate for each attribute.

Procedure

MLP Classifier tuning

The performance of the MLP is heavily determined by the amount and the size of hidden layers and the solver. To find a good combination of amount and size we create a powerset of a “input set”, e.g. (1,2,3,4). For each set in the powerset we run the classifier once. We track the results in an excel sheet. First we used the lbfgs solver, later we switched to the adams solver which performed significantly better, both in speed and in accuracy.

Improving generalizability

To make sure we get generalisable results we use the K-Fold cross validation with 10 folds. For each fold we compute the error rate of the classifier and return an average error, as well as the variance. This lets us quickly rate the performance of the current combination of amount and size of the perceptron layers. Our code also allows for a “smoothing” operation, where we run multiple versions of K-Fold and average over these as well. We ended up not using it because most results were already satisfying and it increased the runtime by a lot.

Storing results

We use a Python library to write directly to an Excel sheet. For each combination of attribute and MLP-configuration we save the average error and the biggest error deviation. There is also an entry called smoothing, it keeps track of the amount of smoothing iterations done to the data but we ended up not using it, as explained in the previous paragraph.

Evaluation

Average number of downloads

This proved to be the most reliable way of determining the price of a game when looking at the whole data set. With an error rate of around 5,2 it predicted the price of a game almost independent of the configuration of the MLP classifier.

Average hours played combined with number of downloads

This was second to the number of downloads when it comes to performance over the whole dataset. There were a couple MPL-C configurations that worsened the performance by quite a bit, e.g (1,2,4) with an average error rate of 5.4. This is interesting because it suggests that the introduction of average hours played actually worsened the result.

Genres

Against our expectations, genres don’t perform very well, averaging at around an error rate of 3.4. We assume that this score can be improved by combining it with another attribute, but as the genre-data is processed in a binary way (one attribute for one genre, 1 or 0 indicating whether the game has this genre or not), it takes quite a bit more time to compute. Due to time constraints we decided to not investigate further. It's noteworthy though that the average error is higher than other attributes, while the variance is quite a bit lower than the rest.

The whole dataset vs the first 500 entries

The data set that we are using is sorted by the approximate number of downloads. This is slightly problematic as there are a lot of cheap or even free games with barely any users or ratings. This distorts any attempt of classifying the data. When running our program on only the first 500 entries, the performance increases drastically. The best performing classifier yields an error rate of only 3.086 with the attribute average hours played and downloads (50,5), closely followed by 3.088 user ratings (30,5).

Implementation

After loading the data from the dataset, we extract the columns with the attributes we want to use. To automate that, we have a list that stores tuples containing the name of the attribute and its corresponding location in the dataset. We loop through that list of tuples; for each iteration we have a nested loop that iterates over a predefined list of tuples that contain the size and amount of layers for the MLP classifier. That means for every attribute we loop through all MLP configurations and run the classifier with 10-K-fold. This lets us crunch many configurations at a time and allows us to run the computation overnight. For each classifier that we train, we also compute its performance, measured by the average error and its variance. These get stored in an excel sheet, which we write to via a library called xlsx-writer.

Conclusion

Increasing the number of hidden layers and increasing the number of units makes the classifier perform slightly better, but this boosts performance only so far. What type of attribute and the combination with other attributes really made a difference. From this follows, that the price of a game seems to be more dependent on some attributes than on others. This was expected. Noteworthy is also that the overall performance, regardless of attribute increased by a lot when reducing the amount of data points. We ran some iterations of our code on only the first 500 data points instead of the full 27000 and the average error dropped from 5,3 to 3,1. We assume that this happened because our dataset is not complete and especially games with just a few hundred downloads don't have entries for reviews or average playtime. So when we only consider the first 500 games, we can make more or less sure that all data points are complete.

Clustering approach:

Procedure

Preprocessing

Since clustering requires a numerical input, we first transform the string format of genres into a list of 0’s and 1’ for whether a genre is or is not included in the applications description.
This list contains 29 genres going from “Action” to “Photo Editing”.
As a thirtieth entry we add the price of the respective game. Afterwards we perform clustering on the resulting data.

Clustering

To perform clustering we used the sklearn.cluster K-means. With 10 up to 100 clusters. Initial clusters are chosen by the built-in k-means++ function. Every clustering operation is repeated 10 times and the best cluster configurations are chosen.

Post Processing

After the clusters have been calculated, we extract the centroids. It is observed that the centroids have values between 0 and 1 in the first 29 dimensions containing the genre information. Since this is an unusable format, we converted these numbers to the closest integer, such that all values are either 0 or 1 by means of a threshold of 0.5.
The validity of this approach was tested by the “representativeGenres” class, by comparing the representativeness of the centroids against the one by the top occuring genre combinations.
To compare them we calculated the average difference between the most similar centroid or genre combination and each game’s genres.
The results showed that the centroids are marginally worse to the top genre combinations at any amount of centroids and top combinations, with a threshold of 0.5 netting the best results.

Obtaining output

Due to the low complexity of the output it was sufficient to obtain the results in the form of print statements and denote them here directly.

Results

The results obtained are inconclusive. For any amount of clusters between 10 and 1000 the price of all centroids is 7.19 euros. The average error from this result is about 4.944, which is considerably worse than the prediction by the MLP.

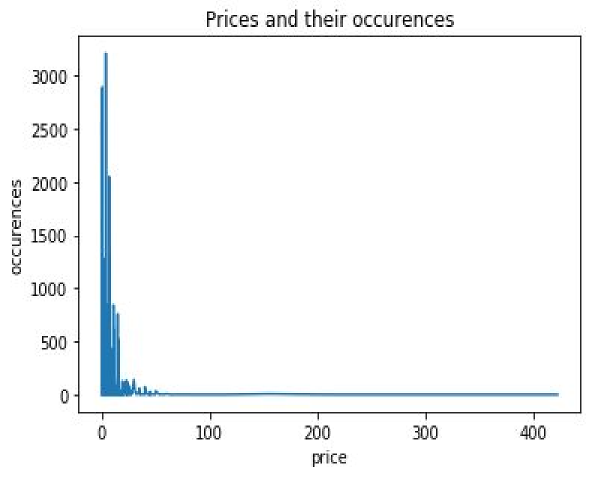
Conclusion

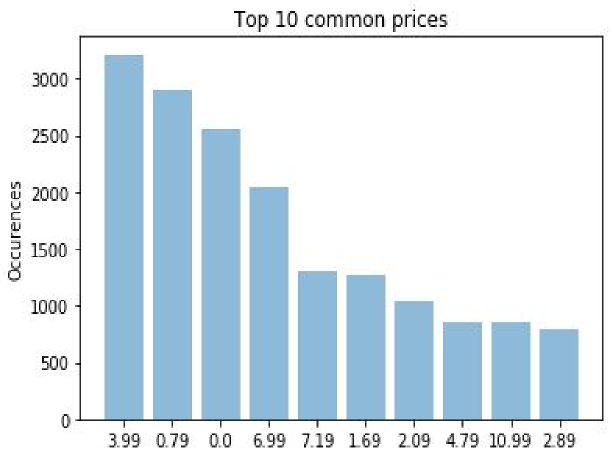
An explanation is that firstly genres may not be a good indicator of prices in general. Secondly it is evident that clustering is inferior to the MLP given the task at hand. Observed in the price distribution analysis section we can see that 7.19 lies within the top 10 occuring prices, with the average price being close to that as well. It is likely that the estimated price originates from the average price, from which it only deviates by 1.11 euros.
However why exactly 7.19 is chosen over other frequent prices is inexplicable.

Analysing attribute distributions and further discussion

Price

The price range in our data ranges from free to 421.99 euros. The average price is 6.08 euros and the median price is 3.99 euros.
As visible in the graph below, there are very few applications that are very expensive and the price range is not very evenly distributed.

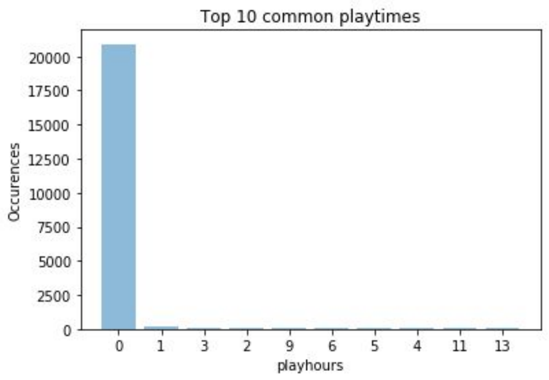


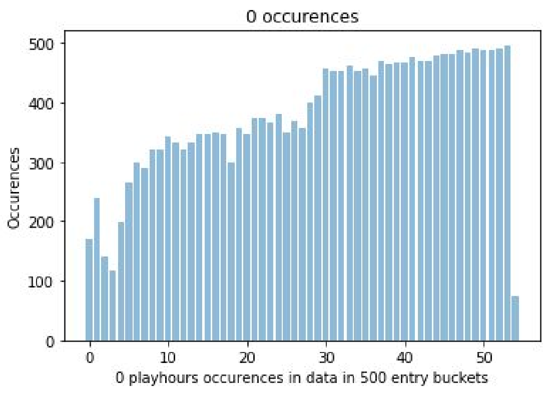


Average playtime

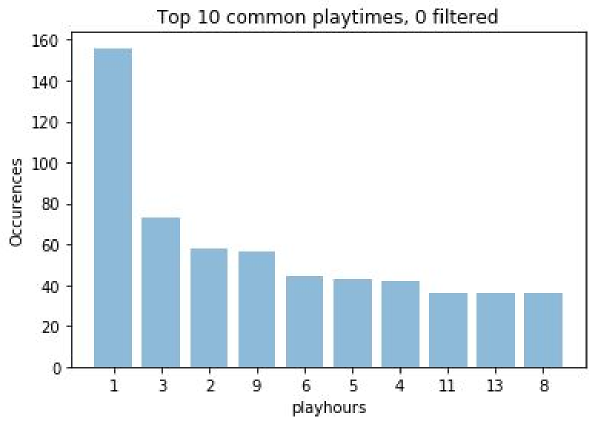
The average playtime range in our data ranges from 0 to 190625 hours. The average average playtime is 149.81 hours and the median average playtime is 746 hours.

From the first bar chart below it is evident that 0 hours makes up a huge chunk of our average playtimes, which, by more making up more than 74% of the data, makes classification using just this attribute on its own impossible. The second bar chart shows a strong tendency for zero playhours in later entries, strongly explaining the positive effects of only analysing the first 500 applications, since if many entries have the same playhours, 0, they are hard to distinguish.



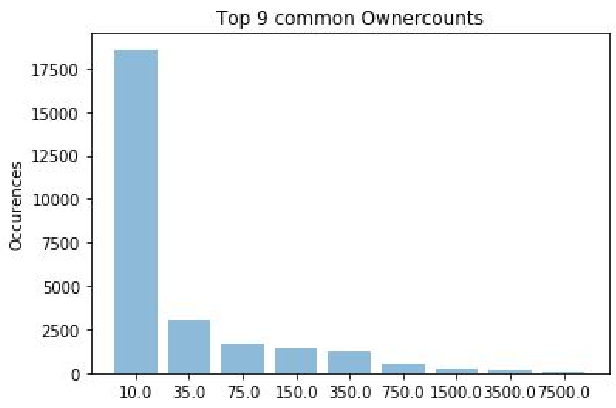


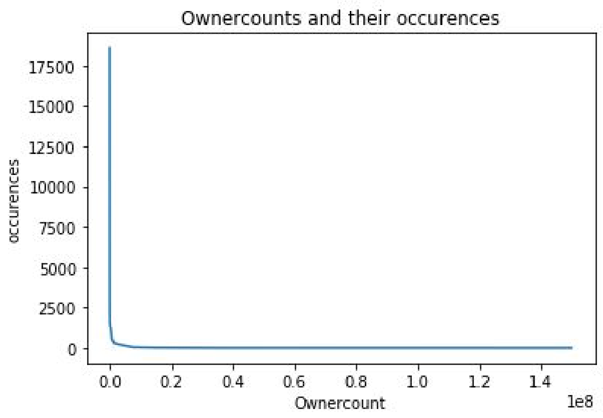
Below we have a filtered version of the playtimes with median and average of 748 and 657.37 hours respectively. The average playtime has been impacted strongly. The new global playtime distribution is still quite dense, but the barchart now shows the expected logarithmic distribution of playtime occurrences. The global ratio between range and occurrences is still extreme. This broad range most likely causes our MLP to overfit, since prices can be assigned to single playtimes.



Average downloads/owners

Owner counts range from 0 to 150000000. The median owner count lies at 10000 while the average owner count of an application is 134090.49. From the below chart it is evident that the occurrences are not very evenly distributed given the range, however not as bad as the playtimes without zero filtering.





Given that our data is sorted by owner count and we can observe that the higher owners counts occur much less, we can conclude that there is much more variance in the earlier applications owner counts, making games more distinguishable and thus giving another explanation for why the MLP is more accurate with the initial 500 applications as isolated dataset.

Genres

Genre distribution is omitted due to the exceeding page number for this report.

Appendix

(1) https://venturebeat.com/2017/06/28/using-steam-data-to-tell-you-if-your-game-will-sink-or-swim/
(2) https://towardsdatascience.com/predicting-hit-video-games-with-ml-1341bd9b86b0

Kaggle page for source data:
https://www.kaggle.com/nikdavis/steam-store-games