Recommending Next Five Games

Hossein Zandinejad

Abstract

A recommender system [1], or a recommendation system [2] [3], is a subclass of information filtering systems that seeks to predict the ”***rating***” or ”***preference***” a user would give to an item. However, when the datasets being classified are large and highly complex and when the set of potential classes is large as well, these models could be improved by incorporating more information about each user's previous preferences. In this paper, I examine several models and attempt to find the most efficient one.

1 Introduction

- The goal of this project is to recommend the next five games based on previous choices the user has made. The ability to recommend the user's future choices is highly used in App Stores, Video Streaming Websites, Online Shops, etc.

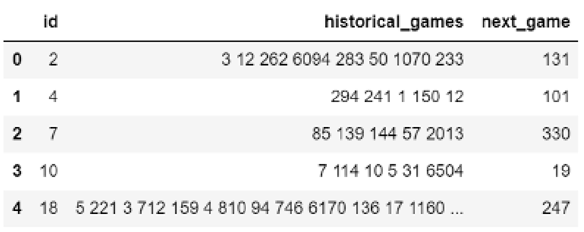
- Here, I first describe the data I use and how I process it. Then, I explore whether these questions can be answered using techniques for classification. Finally, I run multiple Machine Learning models and choose the most accurate model.

- One of the main problems that I faced, was related to the CSR Matrix. The problem with the CSR Matrix is that the number of features varies per training example. This makes learning hard, and in particular, makes it hard to extract information on exactly what the algorithm is using to “learn”. To address the difficulties of working with CSR Matrix, I tried to use ***feature reduction*** methods which I will get into in part II.

2 The dataset and features

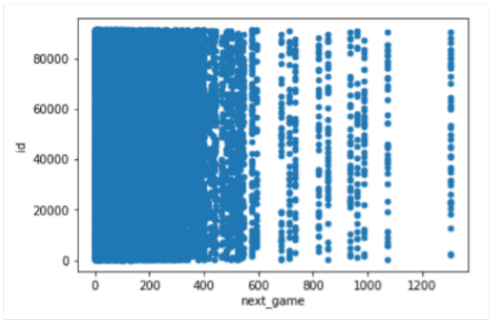
For this project I use the SummerCamp 1400 CafeBazaar dataset.

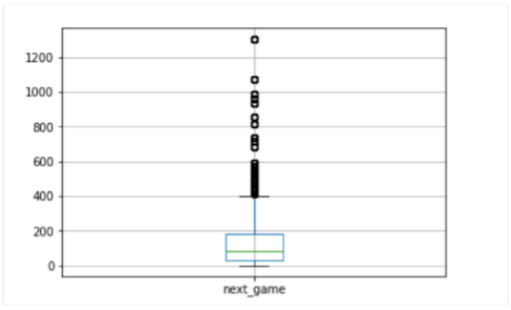
This dataset consists of three features named ”***id***”, ”***historical_games***” and ”***next_game***”. The ”***id***” feature shows the ID of each 30588 user which is not of obvious importance in this project, With that being said, I deleted it and used the other features in constructing the models. The ”***historical_games***” feature is the history of each user's previously installed games which is a string of game IDs split by space. The ”***next_game***” feature is an integer that represents the ID of the next game that the user will install.

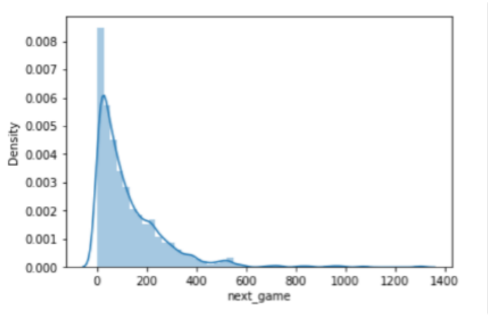


	id	historical_games	next_game
0	2	3 12 262 6094 283 50 1070 233	131
1	4	294 241 1 150 12	101
2	7	85 139 144 57 2013	330
3	10	7 114 10 5 31 6504	19
4	18	5 221 3 712 159 4 810 94 746 6170 136 17 1160 ...	247

From the dataset and the ”***historical_games***” column, we get that game IDs range from 1, 7489 but the vast majority of game IDs in the ”***next_game***” range from 1, 600. So it is possible that I will overfit this segment of the scale. The minimum ID game in ”***next_game***” is 1, the maximum is 1304 and the 3rd quantile is 183. (See the scatter and the boxplot below)







The above statistics show that the "***next_game***" column is like a search result because %25 of users next game IDs range from 183 to 1304. (In more than 7000 game IDs)

I changed the original dataset to a ***Compressed Sparse Row Matrix*** (CSR-Matrix) using the ***SuperML*** library in R (Or ***Scipy.sparse*** library in Python) to construct considered models.

Now, the new dataset (***CSR***) has 7736 features. Just before implementing model selection I wanted to check the effect of the ***Principle Component Analysis*** (PCA) so I also ran many models with ***PCA*** as well but the accuracy decreased. It might be because of several reasons:

- For regression problems, I think that this occurs because the new features space are linear combination of original features. Therefore, may there be a loss of information.

- Using ***PCA*** can lose some spatial information which is important for classification, so the classification accuracy decreases.

- ***PCA*** should be used mainly for variables that are strongly correlated. If the relationship is weak between variables, ***PCA*** does not work well to reduce data. Refer to the correlation matrix to determine. In general, if most of the correlation coefficients are smaller than 0.3, ***PCA*** will not help.

I also attempted using ***AdaBoost*** but this did not really improve the model's performance either.

To increase the speed of my models, and duo to the fact that this does not decrease the accuracy, I deleted the game IDs higher than 2000. (To reduce the features based on the statistics and plots) (Saved as train11)

3 Methods

I. Random Forrest: [5] After completing preliminary exploratory data analysis, we shifted our focus to various supervised learning methods, turning first to tree-based methods, which segment the feature space of a dataset into distinct, non-overlapping regions using a greedy approach known as recursive binary splitting.
Prediction of a test example using a classification tree is performed by determining the region to which that example belongs, and then taking the mode of the training observations in that region. A classification tree is "grown" by successively splitting the predictor space at a single cutpoint which leads to the greatest possible reduction in some chosen measure. For our project, we chose as our measure the Gini index (G), which is often thought of as a measure of region purity:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \qquad (1)$$

Where K is the number of classes and $\hat{p}_{mk}$ the proportion of training observations belonging to the k_{th} class in the mth region. Thus, for every iteration of recursive binary splitting, the splitpoint s is chosen so as to minimize the Gini index and thus maximize the purity of each of the regions. This splitting of the feature space continues until a stopping criterion is reached. While one single classification tree might not have extraordinary predictive accuracy in comparison to other machine learning methods, aggregation of decision trees can substantially improve performance. For my project, I chose to use a random forest of classification trees as one of my prediction methods. Random forests are constructed by building a number of decision trees on bootstrapped observations, where each split in each tree is decided by examining a random sample of p out of n features; in this way the random forest method is an improvement upon simple bootstrap aggregation of trees as it decorrelates the trees. For the purposes of my project, I used python's sklearn package to construct a random forest on my test data set.

I achieved 152 points (%15.2), using RandomForrestClassifier with the following parameters:
$n\_estimator = 100$,
$min\_samples\_leaf = 50$,
$max\_depth = 50$

II. Logistic Regression: [4] My baseline discriminative model was L2-norm Logistic Regression, which minimizes the following cost function:

$$\begin{aligned} J(\omega, c) &= \frac{1}{2}\omega^T\omega \\ &+ C\sum_{i=1}^{n} \log\left(\exp\left(-y_i(X_i^T\omega c)\right) + 1\right) \end{aligned} \tag{2}$$

Where C is a hyper parameter determining the balance between fitting the model and penalizing overfitting with the L2-norm. To construct this model, I used python's ***sklearn*** package. I gained 167 points (%16.7) by these parameters:
$C = 0.15$ and $max\_iter = 300$
I also tried making model with the L1-norm, saga solver and elasticnet penalty which did not even finished running on the dataset provided after 7.5 hours.

III. SVM: An ***SVM*** minimizes the following cost function:

$$J(\omega, b_\gamma) = \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\gamma_i \tag{3}$$

Subject to:

$$y_i(\omega^T x_i + b) \geq 1 - \gamma_i \tag{4}$$

I also used python's ***sklearn*** package to implement this model.

4 Future Works

Given more time I would work to develop or improve on several ideas. The first idea I had was to XGBoost model and the second one was to run Random Forrest Models with different parameters. The third method I wanted to try was an ensemble method to combine classifiers. Some of the classifiers had high bias, such as Linear Discriminant Analysis, and some had high variance, such as Random Forests. An interesting topic that I have not had time to explore is combining these models using an ensemble method, such as Bayesian Model Averaging (BMA). Unfortunately, I did not have the time to code an implementation of BMA.

5 Conclusion

I did an ***Exploratory Data Analysis*** (EDA) first, then I made the sparse matrix for the dataset. After that I tried applying feature reduction using ***PCA*** on my sparse matrix. The next step was to try different models to check which one has the best performance. ***Logistic Regression*** model was the best of all in predicting the next five games.

References

[1] Kim Falk. *Practical Recommender Systems*. Manning, 2019.

[2] Baptiste Rocca. Introduction to recommender systems, 2019.

[3] Aditya Sharma. Beginner tutorial: Recommender systems in python, 2019.

[4] Sklearn. Logisticregression.

[5] Sklearn. Random forrest.