CS591 Final Project report

Recommendation System for Amazon Games

Leshen Sui, Zheming Sun

12/12/2016

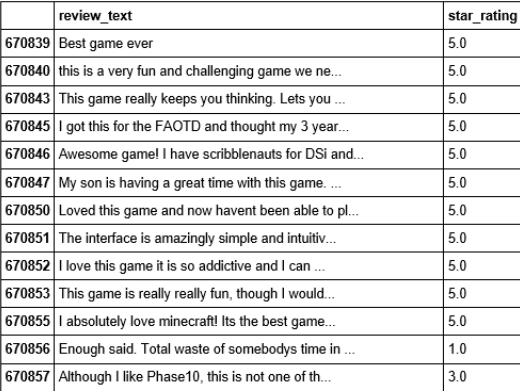
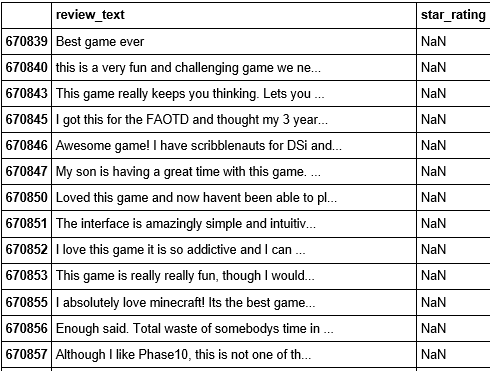
**Abstract**

In this project we are aimed to build a recommendation system for Amazon game apps. For users who give a high star-rating on a certain app, we are going to do recommendation based on two approaches. The first approach is clustering, and the goal is to group games with similar description together and recommend best of them to users. The second approach is to solving a matrix completion problem(fill in a user-item matrix based on known entries), and our expected outcome is users will give high star-ratings for games we recommended. Also, we will use our second model to evaluate the output of our first model by predicting the users’ rating to our recommendation.

**Dataset and data preparation**

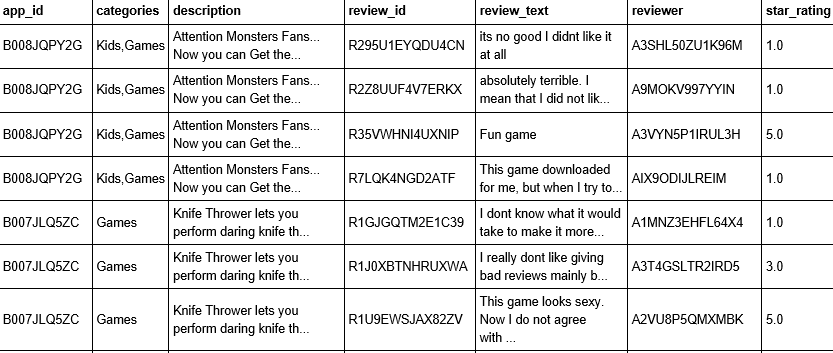
Our dataset is a subset of a huge Amazon app dataset that contains information about apps and users and meta-data about them, including description for apps, categories of apps, developer, reviews, star-rating, review time...For our project purpose, we select all apps whose category is related to game, and keep six columns that potentially useful, which are “asin”(app id), description, reviewer id, review text, user id, and star rating. There are about 500000+ records in our dataset, including 8000 unique game apps and 100000 reviewers, the most popular game has 1181 reviews and one user at most reviews 57 games. For records with missing reviews, we simply delete it, but for records with missing star-rating, we use classification techniques to predict the missing values by stemming, removing stop-words and TF-IDF vectorizing reviews text. First we pick out all records with non-missing star-rating, and then split those records into training set and validation set(7:3) and tried different classifiers. After fitting the model we do evaluation based on the average difference between the true star\_rating and predicted staring\_rating. The intuition is we want our predicted star\_rating is close to the true star\_rating; for example, predicting a 5 star to a review when the true rating is 4 star is acceptable, but we want to avoid that predicting 1 star rating as 5 star. Therefore avg(|ytrue - ypredicted|) should be as small as possible. Among our experiments, Naive Bayes, Random Forest, Linear regression and Logistic regression gives the final outcome; SVM and KNN are too slow so I give them up. Here is the result:

It turns out that logistic regression gives highest accurate score and lowest average error, The average error of Logistic regression model is 0.58, which means that if I predict a star\_rating as x star based on logistic regression model, the true value should fall in the range (x - 0.58, x + 0.58) on expectation, which is acceptable. Also, the accuracy score is 0.65, which means 65% of the test data are prediction exactly correct. Therefore, we are going to use logistic regression to predict our missing star-ratings:



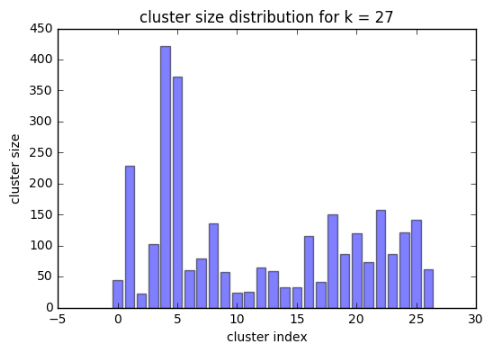
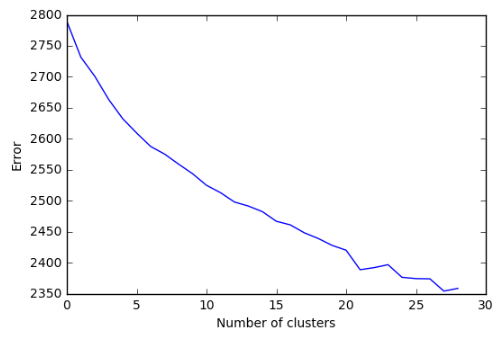
Notice that most of the predictions make sense. For example, “Best game ever” is predicted as 5 star, but “total waste of somebody’s time” is predicted as 1 star.

Our final dataset looks like the following(without missing value):



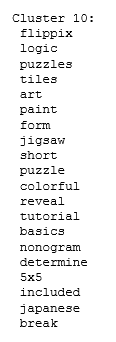
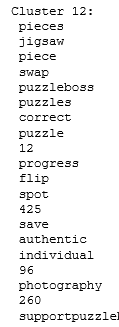
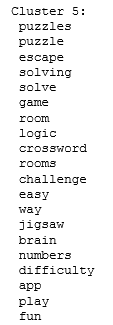
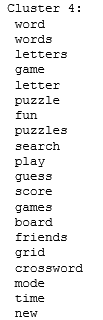
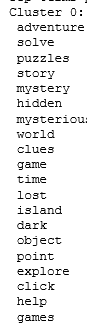
**Recommendation based on clustering**

Our first approach is to cluster games by their descriptions, because games with similar descriptions belong to same type. For example, if a reviewer rates highly for FIFA 17, then he might be a sports fan so recommending good sports game to him would be reasonable. Again, we use some normal text processing like TF-IDF vectorizing and words selection to construct a document-term matrix, and apply clustering algorithm. we choose k-means algorithm because it doesn’t require too much time complexity and generates clusters with meaningful labels. Compared with k-means, sizes of clusters generated by hierarchical clustering is highly skewed. There are always a very huge cluster and some small clusters, which increase bias for recommendation. In order to apply k-means, the first step is to choose a reasonable k value. We utilize “elbow method” to find the k value that minimize the square sum of error. It turns out that k = 27 is the best choice.

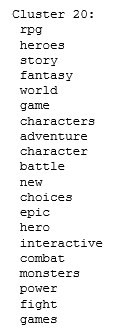
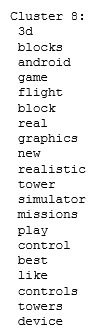
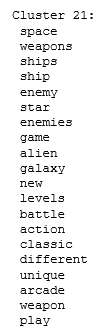
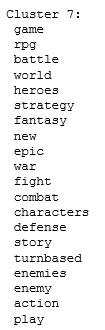


It’s ideal that all clusters have similar number of games, so we plot the distribution of clusters’ sizes. Most of the clusters have a size between 50 and 200.

Then we look deep into each cluster, trying to assign some meaningful labels for them. First we find some clusters that related to puzzles and words game:



We also find some RPG games related to war:



Other types of games involving trivia games, animal hunting games, kid games and cards games. Most of clusters have top terms that indicate game types. Therefore, we are pretty successful in clustering similar descriptions together. Then, we are going to rank apps in each cluster by the average star-rating. The algorithm for recommendation is as following: for every user we find the games he reviewed before, and then find clusters those games belong to. Therefore we are able to rank games within whose clusters and select top 10 games as our recommendations. If we don’t have enough information for a user(a user only played one game and rated 1 star, for example), we can simply recommended the most popular games among all clusters.

**Model Analysis**

The advantages of this recommendation algorithm lie in two aspect: first, the games we recommended are similar in types. We’ll avoid cases such as a reviewer provides a high star rating on sports game but we recommend him angry bird. Second, the games we recommended have high quality. Only games that have high star rating will show up on the recommendation list.

However, this recommendation approach is over specialized. Users are only recommended with games in same categories they played before, and only games that have high overall rating have chances getting recommended. If a user doesn’t show interests to some categories, best games of that categories will never get recommended to him. For this consideration, we design another approach focusing on predicting how a user will rate for any unknown game.

**Recommendation based on regression**

Utilizing the networkx package, we build a bipartite graph consists of users and games, and a weighted edge represents the corresponding star rating. Our goal is to predict the weights of edges which currently not exist. Intuitively a star rating depends on two aspects: the “harshness” of users in terms of rating and the quality of the item. So we take the average of that user’s past rating and the item’s average rating as two importation features. Besides, we also take consideration of the popularity of a game, reflected by the number of users who reviewed this game; and the loyalty of a user towards Amazon games, reflected by how many times the user wrote reviews. Intuitively, popular games tend to receive high star ratings, and loyal users tend to give high star ratings. Still, we split our dataset into two parts, one part for training(70%) and one for validation(30%). Again, we construct 4 models: Random forest, naive bayes, logistic regression and linear regression, and evaluate the performance of our models by the MSE score:

In this case we prefer linear regression to other models. On the one hand, the R^2 is almost 0.8, meaning that 80% of variation are explained by current features. On the other hand, linear regression generates lowest MSE value, because that’s the only method which can output decimal numbers(Other methods are classification techniques whose prediction output are categorical, which are integers from 1 to 5).We also use cross validation to ensure the reliability of this regression model(see Model Evaluation part).

**Model Analysis**

Then, it’s able to use the model to predict all missing entries of our user-item matrix. If the predicted rating is high, the game would be a good choice for recommendation. This method makes it possible to jump out of certain categories when recommending games to users, and because the MSE is acceptable, the result of recommendation should be reliable.

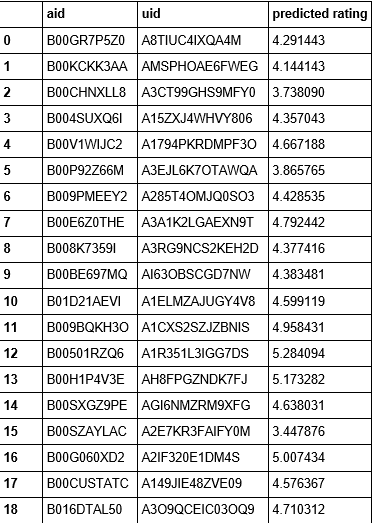
One of the drawbacks of this method is it doesn’t take the user bias towards different types of games into account. For example, a user who tends to rate highly on sports game may give low star ratings on trivia games just because he thinks trivia games is boring(nothing with the game itself). Therefore just taking the users’ average might be too general. And one way to improve this model could be taking advantages of the result of clustering in our first approach, because each cluster represents a category. In this way, we can calculate users’ bias towards different types of games, and add some adjustment terms for our regression formula.

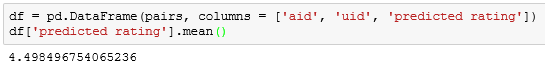
**Model Evaluation**

In order to evaluate our regression model, we split our training set into 10 piles, and each time we use 9 piles to train our model, and the rest one pile for testing. We use MSE(mean square error) to evaluate the performance:

The average test MSE is 0.79, which is pretty good.

Evaluating our first approach is a little more difficult, because it’s impossible to know how a user will actually rate the item we recommend to him. Therefore we utilize our regression model to predict the star rating for the recommendation outcome generated by clustering and ranking. Our expectation is the average rating should be much higher than the global average rating. In total we generate about 35000 user-item pairs, every user gives a 5 star rating for a game so we find the cluster that contains that game first, and then pick one or two top-10 games inside the same cluster:



 The average of users’ rating to our recommendation is 4.4985, which is higher than the global mean(4.04437).

**Conclusion and Future Works**

Up to this point, we are pretty confident that our two models achieve our objective to a large degree. Our first model groups similar games together and rank games by average rating, and we predict that on average users will rate our recommendation about 4.5 stars. Our second approach is also reliable in terms of MSE score, and it’s easy to understand and work fast in practice.

We also tried to predict unknown star ratings based on matrix factorization, but the user-item matrix is huge and sparse so it takes forever to run our algorithm. In future we will explore more methods for dimensionality reduction of such matrix. One possible way would be clustering all games on description, and then predicting how a user will rate a cluster of game.

Another approach we don’t have time to try is collaborate filtering. By clustering descriptions it’s able to get different categories, so that we can compute user’ bias towards various types of games, and the predicted rating would be global average plus user bias.

Through this project we practice most of techniques we learned in class, including clustering, classification and regression. We also experience the workflow of solving real problem with big data, from data collection and preparation to model construction and evaluation. This project is meaningful and instructive for our future development.