Xbox Game recommendation system - Implementation & Analysis

Animesh Grover
College of Engineering
San Jose State University
San Jose, CA
animesh.grover@sjsu.edu

Kashika Jain
College of Engineering
San Jose State University
San Jose, CA
kashika.jain@sjsu.edu

Watcharit Maharutainont
College of Engineering
San Jose State University
San Jose, CA
watcharit.maharutainont@sjsu.edu

Vishwanath Patil
College of Engineering
San Jose State University
San Jose, CA
vishwanath.patil@sjsu.edu

Abstract— **This paper is based on a recommendation system project to predict the games a user will be interested in based on some features provided like their interests and transaction history. The focus of this paper is to provide a detailed description of the machine learning techniques and algorithms implemented to build the recommendation system to recommend new and trending games to the users. The prediction is based on the items(XBOX games) that are queried by the users and subsequently clicked. A recommender system is a subclass of information filtering that checks the importance of an item to a user and accordingly preference for that item is decided for the user. Recommender systems are huge today and are used in all kinds of applications including movies, music, document search, and general products marketing. The two most common types of recommendation system are content-based and collaborative filtering. In this paper, we have built predictive models based on both these recommendation system techniques.**

Keywords—Recommender system, collaborative filtering, content-based filtering.

Introduction

The objective of the project is to build a recommendation system based predictive model to expose an item to the users based on their interests and historical games searched by them. The idea is to train the data based on the transaction data available and supply the user with a list of new and trending games. Recommendation system helps the companies make big bucks getting the interests of the users right and marketing it to them in the perfect way. There are 2 classic ways to perform recommendation systems which are collaborative filtering and content-based recommendation, however collaborative filtering has been more successful in giving higher accuracy when it comes to prediction. In this project, we have used three approaches to build a game recommendation system which are - i) SVM based recommendation, ii) KMeans based recommendation and iii) Programmatic approach and then we compare them with the solution used by the winning team. In this paper, we have provided a detailed description of all these approaches.

Background

This problem was originally part of a Kaggle competition, therefore a lot of material was available to understand this problem deeply and also look at the winning solution and understand their approach to the problem. The winning team mainly relied on several different string comparison techniques and using the timestamp information provided in the dataset. A query comparison is made using the entire query string and also perform a word comparison where the query is split into words. A score is provided for each comparison, query and word, and then they are merged together with some customer history to compute results.More detailed implementation of this solution will be described later in the report.

For Implementing and analyzing different algorithm and approaches to this issue, few of the journal related to the topic were read and understood first to better grasp the history and understand all the challenges in this problem. The dataset in this problem was mostly categorical value and in order to effectively use this type of data we consulted and took inspiration from a few journals which helped us guide and try different approaches for this problem. Below is a list of few conference papers and journals we consulted or found most useful in finalizing our approach to this problem:

- Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values[6].
- The Xbox recommender system[7].
- Context-Aware SVM for Context-Dependent Information Recommendation[8]
- Application of Dimensionality Reduction in Recommender System - case study[9]
- new user similarity model to improve the accuracy of collaborative filtering[10]

I. Dataset Description

The dataset that we have used for the purpose of our project has been taken from a Kaggle competition. It consists of the following features-

User Token - User ID
SKU Clicked - A Stock Keeping Unit that users click on
Search Query - Category of each SKU
Click Time - The total time the SKU was clicked on
Query Time - Total Time each query was run

The dataset consists of 42,365 training elements. This comes from over 38,000 users. In addition, The dataset also consists of a separate list of about 870 Xbox 360 related SKU's and some metadata such as product description, price history about these items. This list has multiple duplications, and after culling these, we find that this list contains 437 unique entries The test data has the same information as the training data except for the SKU, which needs to be predicted.

Exploratory Data Analysis

The initial steps performed on that data involved exploring the data to understand the various features and understand the different relations and correlations among variables in the data. As the primary objective of a recommendation system is to provide a user with a list of games that he might be interested in based on the kind of queries he has done. To make it easier to understand similar games, we found using correlation, the similar games for a user query. Similarly, users similar to each other was also found.
A user-item filtering will take a particular user, find users that are similar to that user based on similarity of items, and recommend items that those similar users liked. On the other hand, the item-item filtering will take an item, find users who liked that and explore other items those users or similar users also liked. Item-Item Collaborative Filtering gives the users who liked this item also liked some other item. User-Item Collaborative Filtering gives users who are similar to you also liked something else. In both cases, we create a user-item matrix which built from the entire dataset.

A distance metric commonly used in recommender systems is cosine similarity, where the ratings are seen as vectors in n-dimensional space and the similarity is calculated based on the cosine angle between them. Cosine similarity for users a* and *m can be calculated using the formula below, where you take the dot product of the user vector uk and the user vector ua and divide it by multiplication of the Euclidean lengths of the vectors.

$$s_u^{cos}(u_k, u_a) = \frac{u_k \cdot u_a}{||u_k|| \, ||u_a||} = \frac{\sum x_{k,m} x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

To calculate the similarity between items m* and *b you use the formula:

$$s_u^{cos}(i_m, i_b) = \frac{i_m \cdot i_b}{||i_m|| \, ||i_b||} = \frac{\sum x_{a,m} x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

The steps performed after this is to make predictions. After we have user_similarity and item_similarity matrices, the prediction is done by applying the following formula for user-based CF:

$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum_{u_a} sim_u(u_k, u_a)(x_{a,m} - \bar{x_{u_a}})}{\sum_{u_a} |sim_u(u_k, u_a)|}$$

In this case, we use root mean square error(RMSE) as the evaluation matrix to check the prediction accuracy-

$$RMSE = \sqrt{\frac{1}{N} \sum (x_i - \hat{x}_i)^2}$$

The user similarity calculated above is useful in game recommendation as we can expose similar items to the users. The image below shows correlation between different games and based on high correlations, items can be recommended to similar users.

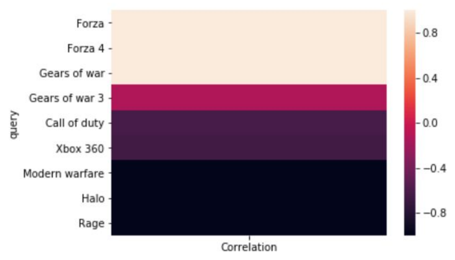


Fig: Correlation between Xbox games

II. Model Implementation

We have implemented this project by building 3 different models, firstly using the SVM based approach we were able to achieve a model accuracy of 61 %. Using the K-means a total accuracy of 63% was accomplished. Finally, an accuracy of 40% was achieved on the test dataset by adopting the programmatic approach. Clearly, the SVM and the K-means models are the winners here. Below is the description about working of each model-

1. SVM Approach -

In general, SVM is not the best method to be implemented for a recommendation system. However in our case when combined with a grid search algorithm, SVM gave an accuracy of prediction as good as KMeans. The idea here is that in recommendation systems, we use SVM. The algorithm repeatedly corrects the missing values in the user-item matrix and improved the model prediction.
The gridsearch algorithm is used to improve the prediction accuracy. Finding the right parameters (like what C or gamma values to use) is a tricky task. The gamma parameter in SVM defines how far the influence of a training example goes where a low value of gamma means 'far' and high-value means 'close'. The C value helps in maximizing the SVM decision boundary.

This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a GridsearchCV. The CV stands for cross-validation which is the GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The most common way of defining a grid of parameters is through a dictionary, where the keys are the parameters and the values are the settings to be tested. The biggest advantage of using GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC and creates a new estimator, that behaves exactly the same - in this case, like a classifier.

2. K-means Approach

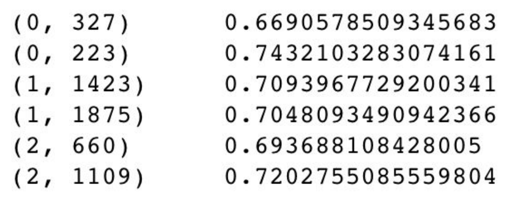
In this approach, we recommend products *(SKUs)* based on the similarity between queries. One thing we can clearly see is that we cannot categorize queries into the same cluster just by comparing whether the query strings are the same to each other since the same meaning of queries can vary in many forms. To illustrate this, consider the following queries:

1. *batman arkham city*
2. *Arkam city*
3. *Arkham city*

These queries are meant to get the same result but users may not type in the full name of the game and there is always a chance of typo error. Hence, in order to categorize the queries, we measure their similarity instead.

K-means is a popular algorithm to cluster objects as the algorithm is easy to implement and fast, with the time complexity $O(n * K * I * d)$ where n is the number of objects, K indicates the number of clusters, I represent the number of iterations, and d is the number of attributes. The algorithm accepts numerical value as input so we need to convert all of the queries into numbers first. This is where Tf-IDF algorithm comes in.

Tf-IDF algorithm reflects how important a word is to a document in a collection or corpus. The result after applying this algorithm to the queries is shown in Fig. Tf-Idf Result.



```
(0, 327)      0.6690578509345683
(0, 223)      0.7432103283074161
(1, 1423)     0.7093967729200341
(1, 1875)     0.7048093490942366
(2, 660)      0.693688108428005
(2, 1109)     0.7202755085559804
```

Fig. Tf-Idf Result: Results of the first three queries after applying Tf-IDF algorithm to all query data. For example, The first two lines show the values of each word in the first query.

One thing to consider before using the K-means algorithm is that the algorithm needs a number of clusters K before it can begin processing. From our dataset, we do not know the number of clusters beforehand since the queries can come in many different meanings and forms. Fortunately, the appropriate K value can be selected by choosing the K value with minimum error. The error of this algorithm can be calculated by the sum of squared distances of all queries to the nearest cluster center. Starting from K = 500, we plotted an Elbow curve graph between error and value of K as shown in Fig. Elbow Curve Graph. K value can be selected when the graph begins to converge, in this case, K = 3500 is selected.

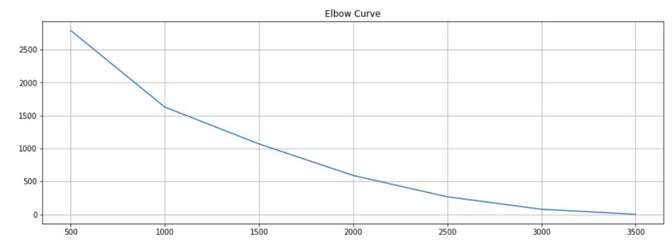


Fig 1. Elbow Curve Graph: A graph between the sum of squared distances of all queries to the nearest cluster center *(y)* and the number of clusters *(x)*. Incremented by 500 per iteration.

Now we have data and algorithm ready to solve the problem. Fig. Step by Step shows an overview of the implementation used to recommend *skus* to users.

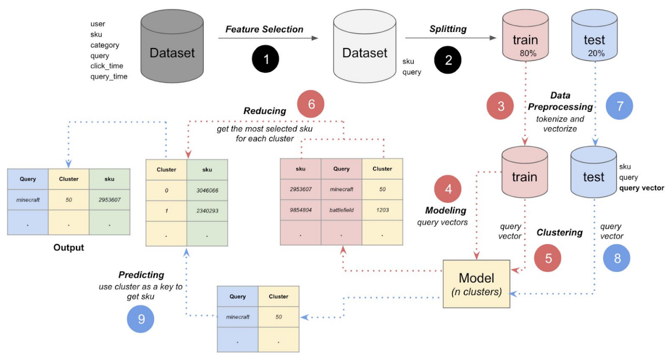


Fig 2. Step by Step: Overview of the approach to solving the problem.

After we finished with *Feature Selection*, we then split data into a training dataset and test dataset with a ratio of 8:2 respectively. The test dataset is kept away and the process continues solely on the training dataset. For the next step, all the queries are converted into numerical values by the Tf-IDF algorithm and we now have *query vector* which will be an input to the K-means algorithm. In step number 4 in Fig. Step by Step, we input all the vectors into the K-means algorithm. As a result, we have a model which accepts query vector as an input and outputs the cluster number. Move onto step number 5, we categorize all query vectors from the training dataset by feeding the vectors into the model we got after step number 4. For each cluster, we count all of the selected *sku* and we choose the most selected one and map it with the cluster number, as shown in step 6. By using this mapping table we can predict *skus* to users by just using the cluster number. Proceed further, we do the same to test dataset until before step 9. The data we have now is the test dataset which has queries map to cluster number. For the last step (step 9) we just have to use mapping table between cluster number and *skus* we mentioned earlier to predict *skus* for each query in the test dataset.

The accuracy of this approach is calculated by comparing the predicted results with the actual results in the test dataset. It turned out that by using this approach, we got the accuracy around *64.4%*.

3. Programmatic Approach

For the Programmatic approach, we have eliminated some fields such as clicked time, query time from the train data. The search query has many spelling mistakes and also consists of many redundant queries. Thus for the purpose of removing such redundant data, we have used Levenshtein distance, a built-in library of python for calculating the similarity between two strings. Queries are considered same if the similarity is above a mentioned threshold.

A mathematically Levenshtein distance between two strings a,b is given by -

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j)=0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j)+1 \\ \operatorname{lev}_{a,b}(i,j-1)+1 \\ \operatorname{lev}_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

The working of the model is as described below -

- The training dataset is filtered and pre-processed using Levenshtein library as described above. Once the input queries have preprocessed a mapping between each user query and the respective SKU are created using the concept of dictionaries.
- We also have a function to find the most frequently occurring SKU’s which can be useful for padding the SKU prediction when the input user query generates only one SKU from the dictionary.
- Once we done with the above two steps we load the test dataset and again here we are dropping the unwanted fields such as click time and query time.
- Now each input user query from the test data is compared with the already existing queries in the dictionary. Once we find the query in the dictionary, returning the key value of the dictionary (i.e the SKU’s from the train data) gives us the prediction.
- Also as mentioned above if a particular input query gives us only one SKU from the model. We use the most frequent SKU’s to fill in the gap for the predictions.
- Thus in this way we are able to recommend five SKU’s corresponding to games for each user input query.

4. Winning Team approach

In this approach, as mentioned above, several different string comparisons are made along with the timestamp information to compute and recommend the user a particular product. A score is provided based on these comparisons and result is computed based on this and some other user data. First, the following tables were created:

- SKU array mapping for the entire training set time window(query)
- SKU array mapping for each day (query)
- SKU array mapping for the entire training set time window(word)
- SKU array mapping for each day(word)
- Item array mapping
- customer queries within a day of each other

4.1. Query Matching

when adding the training data, each query is compared with ones added from the product catalog and merged any misspellings with the Levenshtein distance threshold. This algorithm is very conservative in matching. only very similar queries and words are merged from the product catalog and a new entry is created.

When test data queries are compared with the tables that were generated (mentioned above), we look for an exact match at first and if it is found we return that SKU otherwise, from the query, the largest word is searched in the word table that exists in the query and looks for any word matches in the remaining string. All the words found within the query are added with their SKU rows and returned. Rows from the query per day table are also used to get a week-long data (3 days prior, current and 3 days after) and Gaussian is applied to the weight the current day the most as compared with farther days.

4.2. Word Matching

Words within the query are compared to the words in the word table. If the word is not found, a combination of word similarity algorithms (cosine, Jaccard, Euclid, Jaro-Winkler, Soundex, etc) to get the similarity score and select the best match. Again like before, SKU rows are added for all the words and a week's worth of data is used and Gaussian is again applied to weight the current day the most.

4.3. Adding result

According to the results, combining the scores from the query matching and word matching provides much better accuracy than any other single matching.

III. Results

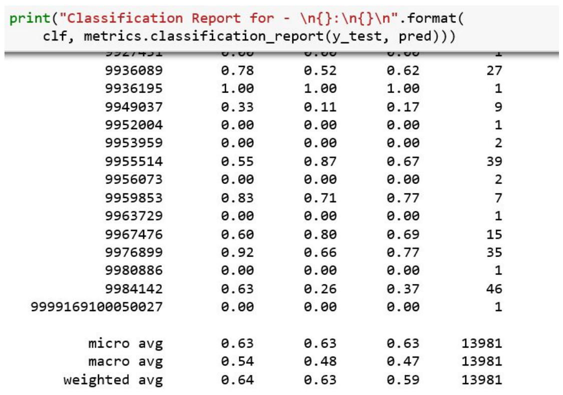
Evaluation

All the above-mentioned algorithms were evaluated in a way to get as accurate and consistent measure as possible because running these algorithms on the whole dataset was not something our testing system could handle in a reasonable time. A different number of records were used to test the accuracy not only to compare the models with each other but also to infer which models accuracy was improving and going down with the increase in training/ data-set.

For evaluation of the SVM model, we have considered two evaluation criteria -

Classification Report - The SKUs predicted for users using the SVM gives us i) Precision - 64%, ii) Recall - 63% and iii) F1-score -59%

Confusion Matrix - gives the performance of a classification model on a set of test data for which the true values are known. It gives a table for comparison of True Positives with False Positives

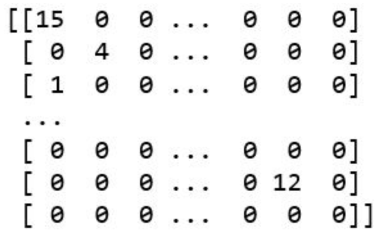


```
print("Classification Report for - \n{}:\n{}\n".format(
    clf, metrics.classification_report(y_test, pred)))
```

```
          9936089       0.78      0.52      0.62        27
          9936195       1.00      1.00      1.00         1
          9949037       0.33      0.11      0.17         9
          9952004       0.00      0.00      0.00         1
          9953959       0.00      0.00      0.00         2
          9955514       0.55      0.87      0.67        39
          9956073       0.00      0.00      0.00         2
          9959853       0.83      0.71      0.77         7
          9963729       0.00      0.00      0.00         1
          9967476       0.60      0.80      0.69        15
          9976899       0.92      0.66      0.77        35
          9980886       0.00      0.00      0.00         1
          9984142       0.63      0.26      0.37        46
 9999169100050027       0.00      0.00      0.00         1

        micro avg       0.63      0.63      0.63     13981
        macro avg       0.54      0.48      0.47     13981
     weighted avg       0.64      0.63      0.59     13981
```

Fig : SVM Classification Report



```
[[15  0  0 ...  0  0  0]
 [ 0  4  0 ...  0  0  0]
 [ 1  0  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0 12  0]
 [ 0  0  0 ...  0  0  0]]
```

Fig 3.: SVM Confusion Matrix

For K-means model, we considered Precision and Recall as a measure of accuracy. Below is the result:

Classification Report - The SKUs predicted for users using the K-means gives us i) Precision - 64.4%, ii) Recall - 65%

For the programmatic approach, we got an accuracy of close to 42% (precision) and about 31%(recall). In this case, we had to infer a low accuracy on large dataset as with only 2000 records we were getting such low accuracy.

Comparison

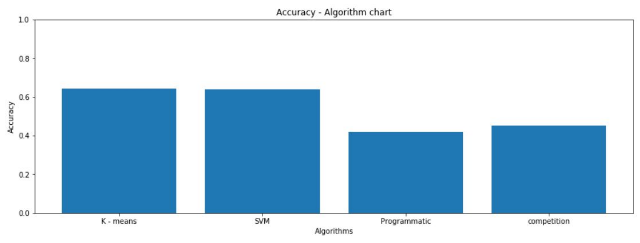


Fig 4. Accuracy comparison between all algorithms

From Fig. 4, we can see that the K-means and SVM are outperforming the programmatic approach and the solution used by the winning team. The winning team solution works really well with small data and gets a decent precision and recall score but when working on large dataset its accuracy and performance drop a lot and this is where K-means and SVM are more efficient. They are trained to work with this type of large dataset not only producing more accurate results in comparatively less time.

Conclusion

Based on the data and the accuracy we calculated, it was observed that with a large dataset K-mean was the best performing model for this kind of prediction with the best precision@k rating. Although the programmatic and the winning team approaches were custom tailored to the dataset and worked really well with a relatively small dataset, the accuracy starts to drop as the dataset gets bigger along with memory and time complexity.

This is a very relevant and ongoing issue and the accuracy for the current state of the art algorithms are still not perfect and new approaches still need to be considered as we get more complex and very large dataset to compute and recommend items based on a similar dataset. In order to achieve better accuracy and efficient use of memory and time complexity, some new state of the art and hybrid models should be also analyzed for this kind of issue. One of the popular algorithms should be considered is singular-value decomposition (SVD). there are some hybrid SVD algorithms like SVD++ and other which have improved the accuracy of the original SVD algorithm as proposed in the Netflix competition. Using K-means and SVD together in a complementary way would be an interesting approach to try in the future (K-SVD).

Work Distribution

Animesh Grover: Worked on and explored various algorithms used in the Kaggle competition. Worked on Preprocessing and feature extraction. Worked with algorithms to compare the accuracy of the different algorithms implemented. Also implemented the programmatic and hybrid approach.

Kashika Jain: Performed Exploratory data analysis and Implemented SVM algorithm to work with this dataset to yield better results, fine-tuned the algorithm to get reasonable accuracy in a short amount of time. Implemented correlation and similarity search algorithms to get user and item similarity to facilitate recommendation of games.

Watcharit Maharutainont: Came up with an approach as well as implemented program to solve the problem, from data preprocessing through prediction. Worked on queries vectorization using TF-IDF algorithm, implemented the K-means algorithm to work with this dataset, and fine-tuned the algorithm to get good accuracy.

Vishwanath Patil: Worked on the programmatic solution for this problem, did the data pre-processing using the python libraries such as Levenshtein. also found the best parameter to work with the given dataset. Compared the accuracy of the built model for 2000 data items, and also for the entire dataset using a logical approach.

Project source code GitHub link

https://github.com/grovera/CMPE-257-Xbox-game-recommendatiion-system

References

[1]https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed

[2]https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-2-deep-recommendation-sequence-prediction-automl-f134bc79d66b

[3]https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

[4]Noam Koenigstein and Ulrich Paquet. 2013. Xbox movies recommendations: variational Bayes matrix factorization with embedded feature selection. In Proceedings of the 7th ACM conference on Recommender systems (RecSys '13). ACM, New York, NY, USA, 129-136. DOI: https://doi.org/10.1145/2507157.2507168

[5] https://www.kaggle.com/c/acm-sf-chapter-hackathon-small

[6]Huang, Z. Data Mining and Knowledge Discovery (1998) 2: 283. https://doi.org/10.1023/A:1009769707641

[7] Noam Koenigstein, Nir Nice, Ulrich Paquet, and Nir Schleyen. 2012. The Xbox recommender system. In Proceedings of the sixth ACM conference on Recommender systems (RecSys '12). ACM, New York, NY, USA, 281-284. DOI: https://doi.org/10.1145/2365952.2366015

[8]Kenta Oku, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. 2006. Context-Aware SVM for Context-Dependent Information Recommendation. In Proceedings of the 7th International Conference on Mobile Data Management (MDM '06). IEEE Computer Society, Washington, DC, USA, 109-. DOI=http://dx.doi.org/10.1109/MDM.2006.56

[9]Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of Dimensionality Reduction in Recommender System -- A Case Study. https://apps.dtic.mil/dtic/tr/fulltext/u2/a439541.pdf

[10]Knowledge-Based Systems, ISSN: 0950-7051, Vol: 56, Page: 156-166 Publication Year: 2014.