INF 558: Building Knowledge Graphs
Game and Requirements Knowledge Graph: Play What You Like

Ravi Kiran Selvam, Rijul Vohra

May 2, 2020

1 Introduction

Games are an ideal way of entertainment for people of all ages. One of the important problems all the people encounter is to figure out the games they want to play next based on their interest and previously played games. So, the most common way is to use the game recommendation system to get suggestions for the games related to the previously played games. The problem with existing game recommendation systems is that it does not provide user-device based recommendations. Most of the time, the games that are recommended to the user would not work on his device. This is a problem we tried to solve in our project by creating a Knowledge Graph which takes into account the user's device configuration and recommends games that work on the user's device. [1]

2 Challenges Faced

We started off our project by crawling the games information from IGDB.com, the game requirements, and the seller information from G2A.com. The information about all the CPUs and GPUs was crawled from Techpowerup.com. One of the major challenges of crawling IGDB.com was to handle infinite scrolling pages. We managed to solve this problem by identifying the Ajax request and simulating the Ajax request to the server. Another challenge was to identify the drop-down options of a form in Techpowerup.com and simulate filling the form to crawl the CPU and GPU information. We used Scrapy and Beautiful Soup for data extraction.

Once the crawling was done, we extracted the CPU and GPU model names and numbers, since a single game might have many CPU and GPU information and it is unstructured. So we used regular expression rules to extract the graphics card and processor model names and numbers from individual game requirements. We also normalized the units for various attributes like Clock Speed (normalized to MHz), Memory and Disk Space (normalized to MB) so that we can use them directly in our queries.

Moving on to the Entity Resolution, we had to link the Games information from IGDB.com (which had 58,606 records) to its corresponding requirements and seller information from G2A.com (which had 34,759 records). The total number of pairwise comparisons is huge, so we used Tri-gram Blocking of the Game Name attribute to solve this problem. We used a combination of Jaro Winkler and Levenshtein similarity. We did two more ER tasks which were linking the games processor and graphics information to the Techpowerup CPU information and GPU information respectively. Evaluating the task was another challenge and we manually labeled 100 examples to evaluate them. The evaluation metrics for the ER are mentioned in Table 1

One of the innovations is to compare the different CPU and GPU models. Since they have a lot of features (both numerical and categorical), it is very hard to manually implement a comparison function. We found benchmark performance scores for some of the CPUs and GPUs online at ⟨https://www.cpubenchmark.net⟩ and ⟨https://www.videocardbenchmark.net⟩ respectively. We used this as the ground truth data and trained our Random Forest Regressor model to predict the benchmark score for both CPUs and GPUs. We evaluated our model using two types of tasks. The first one is a regression task where we predict the benchmark score for a given CPU (or) GPU and this is evaluated using the R^2 metric. The second one is a classification task where we classify the given pair of CPUs (or) GPUs into two classes which indicates if the first one is better than or similar to the second one (or) if the first one is not better than the second one and this is evaluated using the F1 measure. The results can be found in Table 2 and Table 3.

Another important task at hand was to come up with a good ontology to link games with their requirements and the seller information. Apart from using already defined classes and properties in schema.org, we defined 4 custom classes and a lot of custom properties to come up with meaningful triples.

For building the game recommendation system, we need a way to represent the game nodes in the vector space where related/recommended games are near to each other. We could not train the graph embeddings for our KG (which had around 1.2 million triples) because of the lack of adequate computing resources. So, we came up with an innovative way to use the 300-dimensional fastText pretrained text embeddings. First, we computed the individual text embedding for the important game attributes like game

[1] The code for our project can be found here: ⟨https://github.com/ravikiran0606/Game-and-Requirements-KG⟩

name, game summary, genre, theme and game mode. Then, we computed the game node embedding by using a weighted average of these individual attribute embeddings. For recommending games we first filter the games having a rating above a certain threshold. Second, we filter the games that can work on the user device. Third, we compute the cosine similarity between the selected game and all other filtered games and use those scores to rank them and recommend the top-5 games to the user.

The system UI was also a big challenge for us where we created a Flask web application that is connected to the Apache Jena Graph database and used SPARQLWrapper for querying. The logical flow of using our system is as follows: First, the user can enter his device configuration and the system at the backend would map the processor and graphics to a score respectively. The user can even search our knowledge base using 14 attributes and can also filter games that are supported by his system. The Game page displays all the attributes of the game and also has a link to the cheapest seller for that game. It also has links to the Top-5 recommended game pages. We also build a visualization Page for various properties of Game class and other classes using Plotly.js.

3 Conclusion

The project has been a great learning experience for us to be able to apply all the concepts learned in class innovatively and practically. We understood that crawling is time intensive and it is important to build focused crawlers. We also understood the way blocking can massively decrease the computation time. We realized that the ability to perform complex queries heavily depends on a well-defined ontology. Finally, having worked with relational databases in the past, we realized how linked data is so much easier to navigate, query, and the amount of knowledge it can store.[2]

4 Appendices

Table 1: Evaluation Metrics for ER

ER Task	C1	C2	Match	Accuracy	F1 Score
$IGDB : Games- > G2A : Games$	58606	34759	17773	0.84	0.76
$G2A : CPUs- > Techpowerup : CPUs$	34759	2084	13522	0.83	0.74
$G2A : GPUs- > Techpowerup : GPUs$	34759	2655	13317	0.82	0.78

Table 2: Evaluation Metrics for CPU and GPU Benchmark Score Prediction (Regression)

Task	R^2
CPU Score	0.9420
GPU Score	0.9244

Table 3: Evaluation Metrics for CPU and GPU Comparison (Classification)

Task	Precision	Recall	F1 Score
CPU Comparison	0.9444	1.0	0.9714
GPU Comparison	0.8798	0.9339	0.9061

[2]The demo video for our project can be found here: ⟨https://www.youtube.com/watch?v=wa_C4xqBmjo⟩