iOS App Data Collection and Analysis

Matthew Hjørnevik-Saunders, 3230600078

1 Summary

In this project I collect and analyze data on iOS apps, focusing on the mechanisms of the 'You Might Also Like' feature in the App store. Using the python library BeatifulSoup, I collected metadata from apps released on the 8th of July 2023. The analysis consists of, categorizing apps, identifying similarities and differences and understand the app recommendations. App collections are based on direct relationships and similarities, and analyzed between different dimensions like genre, language, size, and age rating.

2 Background

This project focuses on analyzing iOS apps and their recommendations relationships through three nodes, root (the first collected apps), leaf(children of the root) and leaf_leaf (children of leaf). By examining the collected apps that launched on 8th of July 2023, we can try to discover patterns in the recommendation and data distributions across the dimensions.

3 Data Collection Methodology

I started by collected a list of app IDs from the Qimai Data Platform on apps that launched 8th of July 2023. With this .csv file (collectedapps.csv) I can begin scraping data from the app store.

I designed a web crawler in python to gather the following metadata:

1. App name
2. APK size
3. Supported Languages
4. Category
5. Recommended user age

For each app I also collected the data from the 'You Might Also Like' (recommended apps).

The data was collected through three iterations:

1. Root apps: Using the initially collected apps.

2. Leaf Apps: Using the recommended apps from the Root Apps.
3. Leaf Leaf Apps: Using the recommended apps from Leaf Apps.

For scraping the data from the web page, I used python libraries *requests* (retrieves the HTML code from a website) and *BeautifulSoup* (Parsing the HTML code). By examining the source code of the App Store, I determined where in the HTML code the desired values are and extracted them using BeautifulSoup. However, due to the dynamic nature of the app descriptions I was not able to retrieve them using BeautifulSoup. I initially planned to switch to a library that supports more dynamic content (e.g Selenium), but that drastically increased the runtime, so I decided to skip the description extraction to prioritize runtime. The crawler I used for the first iteration, *crawler.py,* used approximately 20 minutes gathering the metadata from 800 apps. Considering the number of apps that would be in the last iteration I had to think of another solution (estimated 100 hours of runtime). For the two next iterations I switched to a concurrent solution using python library, *concurrent.futures*, where I utilize threads to fetch and process the HTML data. Some of the apps found through Qimai no longer exists on the app store.

4 Dataset Description

From the three iterations I collected three sets of data stored separately in a .csv file.

1. The root apps (root_apps.csv, consisting of 510 entries)
2. The leaf apps (leaf_apps.csv, consisting of 1547 entries)
3. The leaf leaf apps (leaf_leaf_apps.csv, consisting of 6084 entries)

I have removed duplicates from the files, so the total number of entries mentioned above is after the removal of duplicates (stored multiple parent IDs into one entry). For storing relationships, I chose to store the parent's ID in the last column of the entries, where the parent ID is the app that the recommendation came from. By doing this I can look at both the parents of any given node and its children. For the construction of collections used for analysis, I made two types of collections. First direct relationships, where I put all nodes connected to each other in one collection, I achieved this by taking

the value of the parent, children, and self from the apps in *leaf_apps.csv.* The second type of collections is if two sets of recommendations contained 5 of the same app.

- Direct Relationship Collections: 1546 collections where the average size is 12.
- 5 Same Recommendations: 4890 collections with an average size are 11.

5 Data Analysis

For the data analysis I used the following python libraries.

- *MathPlotLib,* for creating visualizations of the data.
- *Pandas,* for manipulating and analyzing the data.
- *Scikit-learn,* for analyzing the keywords using NLP.

5.1 Dimension Distributions

Category

The analysis of the collected metadata reveals the distribution of apps across the category dimension. Figure 1 illustrates the percentage distribution of the categories.

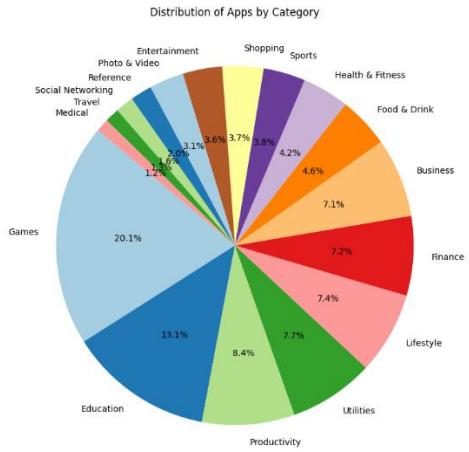

Figure 1: Pie chart showing the percentage distribution for each category, filtered out categories under 1 percent.

The distribution shows that the most dominant form of apps are Games, Education and Productivity, comprising nearly 40 percent of the total distribution.

Language

The analysis of the collected metadata revealed that English is the most dominant language across the apps. Apps only supporting English consisted of 63% of the total apps, and over 90% of the apps had English as a supported language. The second most dominant language is Simplified Chinese.

Size

The analysis of the collected metadata revealed the following about the statistics of the size dimension.

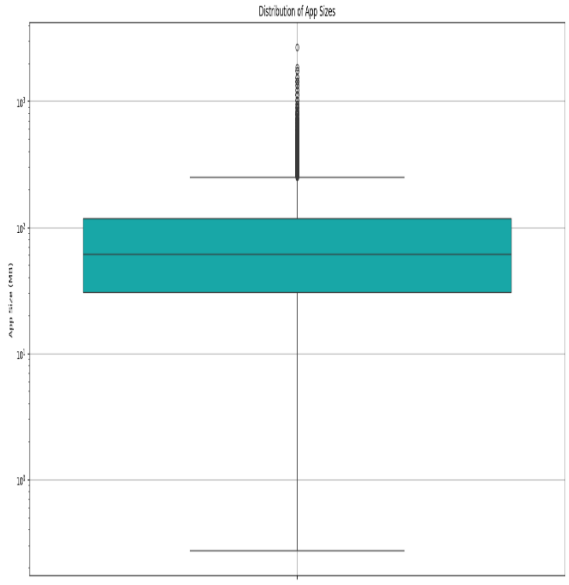

Figure 2: Box plot illustrating the distribution of app sizes, highlighting the median, quartiles, and potential outliers.

Statistic	Value
Mean Size	~ 102.75 MB
Median Size	~ 60.9 MB
Standard Deviation	~ 144.3 MB
Minimum Size	~ 0.27 MB
Maximum Size	~ 2662 MB

Most of the apps collected in the project are relatively small, with some few outliers that are significantly larger than the rest.

Age Rating

From the collected metadata the most dominant Age Rating is 4+, compromising 75% of the total apps. This could indicate that apps are more inclusive to lower age groups to target a broader range of users. The second most dominant Age Rating is 17+, making up 13% of the total distribution.

5.2 Keyword Analysis

I used NLP to analyze the top 20 keywords from the collected metadata. The top 3 keywords from all apps are:

1. Simulator, 187 instances
2. Game, 181 instances
3. App, 177 instances

As seen above the most dominant category was Games, therefore it makes sense that both simulator and game would be the most used keywords from the collected apps.

Top 3 keywords from the Game category:

1. Simulator: 180 instances
2. Game: 164 instances
3. Games: 125 instances

Top 3 keywords from the Photo/Video category:

1. Photo: 36 instances
2. Video: 30 instances
3. Camera; 23 instances

As we can see the keywords are highly dependent on which category the apps belong to, and therefore we can conclude that the most used

keywords would be from the most dominant category.

6 Findings

6.1 Largest Collection

The largest collection of apps gathered by grouping them as mentioned above in section 4. consists of 83 apps.

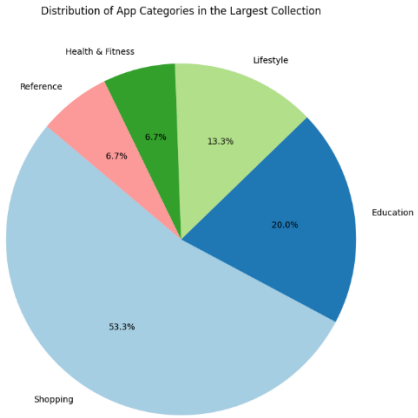

Figure 3: Category distribution among the largest collection.

As seen in figure 3. The apps don't all belong to one category, even though shopping is the most dominant category in this set, the Shopping category is only 3.7% of the total distribution for all apps. By analyzing the other dimensions and the keywords from this collection, I was able to discover the following reasons as to why they were grouped together. The average app size is ~ 20 MB which is consideriably lower than the average from all the apps. Another reason for why they could be grouped so heavily together is how similar the app categories are. Health & Fitness and Education are linkable with Lifestyle, and all the present categories with Shopping. The most common keyword in the collection is English which was also the dominant language, which could be another reason for the collection. Comparing the keyword frequency from this collection to a collection more homogenous, shows that keyword frequency is directly linked with category. Meaning the higher percentage the dominant category is, the higher the frequency of keywords belonging to that category are.

6.2 Node With Most Parents

I decided to analyze the node with the most parents, to try and understand why so many apps where recommending it. *Real Truck* has a considerably larger amount of parents than the rest, 141 parents. I decided to make all parents a collection and analyzing their dimensions and keywords.

Category. They all belong to the Game category

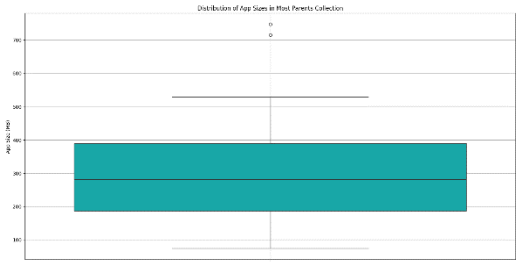

Figure 4: Box Plot of size for the collection with most parents.

Size: The average distribution of the app size is around 300 MB. Considering the amount of apps that are in this collection, the box plot (see figure 4.), seems to be very compact, meaning that the apps are similar in size.

Languages: All the apps are only supported by English.

Ages: The most dominant age is 4+.

Keywords: All the top keywords are related to either simulation, or transportation of some kind, like truck, ship, bus or road.

From the analytics above we can conclude that the parents of *real truck* are games that only support English, around 300 MB and are some sort of simulation game with focus on vehicle of some sort.

I analyzed the root apps that these recommendations came from. The root app collection consists of 29 out of 510, and out of all of their 10 recommendations, 4 were present in every one of them. The majority of the apps in the root app collection are simulation games with a vehicle theme, consistent with the recommendations from both leaf and leaf leaf nodes.

7 Miscellaneous

The biggest challenge of this project was to try and figure out a solution to minimize the run time of scraping data from the App Store. I was lucky to have had a course about concurrency and threads and applied the techniques I learned from that course. I could have also used an asynchronous approach with the get requests, but decided to opt for threads. I apologize for my messy and unelegant code, if I would have more time I would have made it more scalable and readable. If you have any questions regarding the code feel free to ask.