**Introduction**

Kharty is a way of learning where you exercise information on diagrams. It's a mobile phone application that helps people enjoy mastering diagrams of any subject, from geography and anatomy to history and languages. People who think alike may enjoy things alike. Thus, this project will aim to recommend Kharty users their most relevant diagrams based on different Information Retrieval approaches.

**Data**

Data is constructed from two different files. The first file *"maps"* contains maps information as illustrated in Table (1). This data contains 2348 maps with 26 relevant features' data. Some of the features are dictionaries of other features such as coordinates of points on maps. The second file "users" contains user data with only two features "downloaded maps" and "User Id". Downloaded maps column contains lists of all downloaded maps for each user including users who never downloaded any maps. Number of users is 1821 users with 1399 users who have at least one downloaded map or more.

| **objectId** | **categories** | boxes | **points** | isRandomOrder | user | reviewRatingTotal |
|---|---|---|---|---|---|---|
| reviewRating | mapImage | **title** | userObject | previewImage | **tags** | downloadCount |
| **userId** | description | thumb | updatedAt | translatedFrom | version | reviewCount |
| language | originalTags | **public** | createdAt | **subject** | | |

*Table (1)*

**Data cleansing**

***Collaborative Filtering***

To be able to provide good recommendations to users, it's important to count for most relevant data. After a close look at data examples, I decided to keep 8 features for the study as shown in bold in Table (1). Review rating is the average rate per map. Thus, it wasn't needed and I preferred to use download counts for each user as a way to recommend in my first approach; which will be discussed later.

Some maps were private, therefore some maps had to be filtered to keep only public maps as others only available to one user and can't be downloaded. Also, some maps data were corrupted and they do not contain all chosen features. Therefore, they had to be excluded. Furthermore, points on each map had to be extracted from a dictionary of points' dictionaries, where each point is represented by an ID and a dictionary of coordinates and description, then concatenated to a list of points for each map. These points represent different items or titles on a map. The number of maps became 1155. Number of total downloads is 13255 with an average of 11 downloads for each map and 7.27 downloads for each user.

In the second dataset "users", users who never downloaded maps were assigned empty lists instead of Nan values to match the structure of downloaded maps feature. Using map IDs from" maps" dataset and user IDs from "users", I created a matrix of all users and maps exist, then, populated them with their corresponded downloads. Basically, each user is a vector of all maps that contains ones and zeros. Furthermore, I populated these users with the maps they created because these users can't download what they create. These created maps should count in the recommendations as a user behavior of interest and they were given 1s values in the user downloads matrix.

### *Content-based Filtering*

Using maps data frame to collect information about each map which may be relevant to describe each map most, I gave titles and subjects of maps double the weight so they can help navigate better to relevant maps during the recommendation process. Then I concatenated points, tags, subject, title, and description for each map in one feature "names". Double weight for subjects and titles means that every word in each feature is replicated or mentioned twice since subjects and titles are most relevant to better navigate to a map through terms.

Because, maps were created in different languages such as Spanish, more stop words had to be added rather than only English. After tokenization and stemming processes of names using Natural Language Toolkit NLTK platform in python, corpus defined names for each map as list of stemmed terms. The number of unique terms through all maps is 10933 terms. These terms will be assigned as column names for maps-terms association matrix (1155 maps and 10933 terms). I built a function to track each term in each map and populate the corresponding maps-terms matrix with term frequency for each map.

## Recommendation Methods

### *Collaborative Filtering*

Using the user downloads matrix, maps recommendations will be given to target users based on preferences of most similar users' behaviors. This social learning approach uses number of mutual maps downloads between a target user and other users to obtain users who have the highest number of mutual downloads with the target user we want to recommend to.

I built an algorithm that checks for similarity between users by tracking their download behaviors. This algorithm looks for mutual downloads between a target user and all other users in the dataset excluding itself or equal users. I tried to use Pearson correlation as a way to measure similarity yet this gave many zero covariances which then count as Nans for Pearson correlation coefficients. Pearson doesn't seem to work well on similar vectors with similar ratings. For example, if two users downloaded the same similar items or rated them the same, these two users have similar behaviors yet Pearson correlation coefficients between their mutual vectors will be divided by zero since there is no covariance. Also, in case of similarity measures such as cosine, when comparing two equal vectors of same ratings, similarity will be 1. This won't favor how many items are in share between the two users. The similarity between a target user and a user sharing 10 mutual ratings all five stars is equal to a user sharing 1 mutual rating five star (Both are 1). Thus, I used the number of mutual downloaded maps as a way to measure similarity between users.

The first function *Corr* will pass a set of users, a set of target users, number of nearest neighbors to predict from, and a print scores function (1 or 0). This function measures the similarity between target users and all other users, print the indexes of all users that share at least one downloaded map in addition to respected number of downloads for each user. This function returns a dictionary of target users with values of lists of sorted indexes of N nearest neighbors or similar users to each target user.

A function *recommend* will take the same previous parameters in addition to the main data set to acquire titles. This function takes the N most similar users and checks for all maps that a target user did not download among these nearest neighbors. Then, this function recommends the maps in a sorted way

according to their repentances among these N similar users. After calculations of recommendations to all users using the specified parameters, a table of recommendations will be saved into a data frame to be used to retrieve recommendations and provide to the user. A function *recommend_toUser* can then be used to take a userId and return titles of recommended maps sorted or a table of their IDs.

A function MAE was created to check for the Mean Absolute Error using different parts of the user downloads matrix. This function takes a set of target users and a set of users represented as users-maps matrix. Then, these matrices will be split using a split parameter into training and testing sets for targets and users. It will use the training set of columns to obtain most N similar users to each target user. Then, it will calculate the errors of all predictions in the testing set for all target users according to these most N similar users and averages the errors. I used a subset of 500 users and 1155 maps columns. I used 10 of them as target users to recommend to, and I used the 490 other users to recommend from. I used a split of 0.8 to split columns into training and testing sets. 924 maps will be used to train and find most 5 similar users for each target user. 231 maps will be used for testing to obtain predictions from these obtained N similar users and compare them to the actual values of the test target users. The MAE is 0.00134.

***Content-Based Filtering***

Using the contents in the second matrix that represents raw term frequencies of all maps, different similarity measures to find most similar maps to each map in the data set. Maps to maps matrix then is created using the dot product of the maps-terms matrix with the transpose of itself to get maps-maps associations. Each entry corresponds to the dot product of corresponding maps. Since dot product favors more frequent terms, associated scores were normalized using the following normalization equation:

$$s_{ij} = \frac{c_{ij}}{c_{ii} + c_{jj} - c_{ij}}$$

Then, *nsimilarMaps* function was built to take (the original set of maps, the normalized matrix, and the number of most similar maps retrieved) to recommend. This function iterates over the two matrices and return two tables. The first table represents IDs of maps that are most similar to each map ID in the index. The second shows the title of each map and the titles of N most similar corresponded maps sorted by their normalized scores in a descending order.

The second approach using content-based technique is to use Cosine similarity among maps in terms of terms. This measure will still return relevant maps using the raw term-frequency matrix yet it still favors for more frequent terms. Therefore, a *tf_idf* function algorithm was built to determine term frequency – inverted document frequency factor that helps normalize weights. Next function *W* takes the original term frequency matric of original weights and tf_idf scores for all maps to populate weights for all maps-terms associations. This is an expensive process yet computed data is included and can be imported using provided code with file link.

Using this matrix of normalized weights, I used Cosine similarity to fine the maps-maps association matrix that won't favor term frequency and will have normalized judgements that provide better association results. Then, using the *nsimilarMaps* with N of most similar maps, we could retrieve the N most similar maps to each map in the dataset as two tables of titles and IDs.

***Expand a query***

This query uses the maps-terms matrix to derive two different similarities between terms. Using the do product similarity between terms, this query will fine the most N similar words to each word in the query and add it to the query after the query is tokenized, stemmed, and cleaned. This query could also use the Euclidian distance as a measure of similarity to expand a query with N most similar terms. This query takes 4 parameters, the query to expand, number of similar words to each word, terms-terms association matrix, and similarity type ('Euc' or 'dot'). This query will return the expanded query as it's shown in the test code attached.

**Conclusion**

In conclusion, using different techniques in recommendation systems may help boost the downloaded maps and create more patterns among users. These approaches may be presented in different occasions to users as the application is being used. The first approach may be used as recommendations to users while they are browsing maps. On the other hand, the second approach could be presented to users as relevant maps during a session of a similar map. Collaborative filtering turns out to work well in this data and may help engage more data as recommendations become more relevant every time users find more relevant maps and generate more behaviors to learn from.

**Future work**

In this project, I used different approaches to study different techniques that help recommend users most relevant Kharty maps. However, content-based approach still needs to be evaluated to see how effective this approach is and build a comparison between different approaches and adjust parameters. In addition, more relevant features such as ratings or history of querying can be used as factors of recommendations. These algorithms were built to be easily tweaked to engage ratings in case of any future ratings feature added.