RegionNet

Region-Based Recommender System

Supervisor: doc. RNDr. Iveta Mrázová, CSc.

Student: Angie Aslhy Granda Nunez

September 2025

Table of Contents

Introduction

This Individual Software Project [NPRG045] develops a hybrid recommender system to evaluate how grouping users by region affects prediction accuracy, measured by Mean Squared Error (MSE) and Mean Absolute Error (MAE). This work was also presented for the course Programming in Python [NPRG065]. The system combines content-based filtering with model-based collaborative filtering using a modified Singular Value Decomposition (Funk MF) that incorporates user-neighbor predictions for data augmentation. Principal Component Analysis (PCA) was applied to reduce the dimensionality of hotel features, which originally included up to 300 one-hot encoded variables.

Motivation

- Provide the top 5 hotel recommendations for a specific user
- Analyze how the information on reviewers' region improves the results of the recommendation system.
- Enable hotel owners to assess their hotel ratings based on guest nationalities, which could help attract specific demographics.

Data

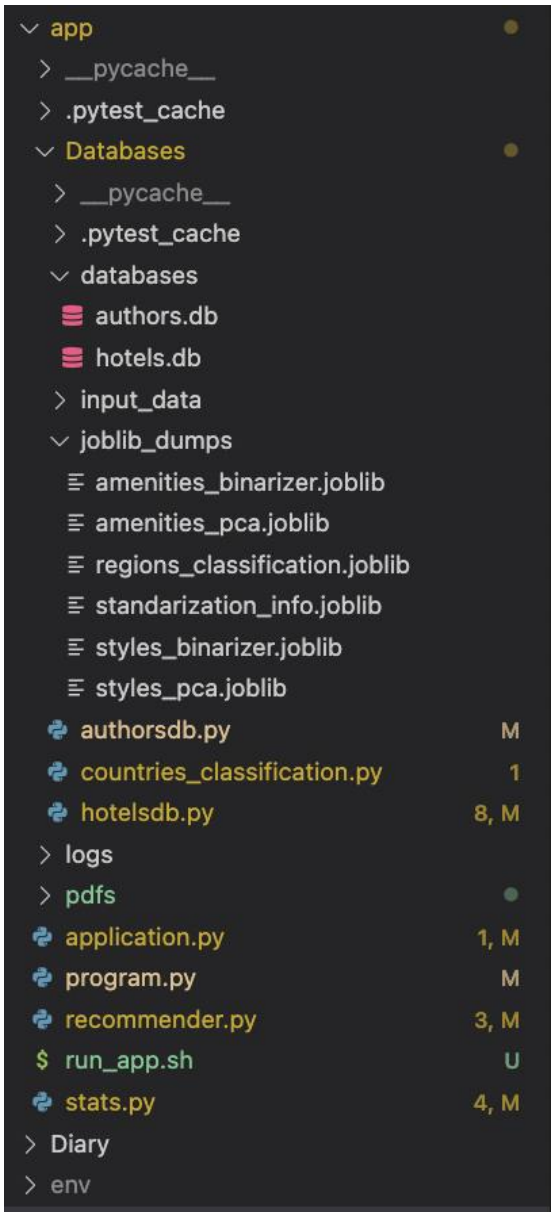
The dataset was primarily taken from the GitHub repository **[1]** and later extended through Tripadvisor API requests for additional hotel data, as well as manual scraping of prices during May 2025. Since prices vary depending on the season, be aware that the hotels recommended together with the price is an estimation. After gathering the data, I created three JSON data files: one containing information about the authors of the ratings, one with the ratings themselves, and one with the hotels, all of them can be processed and transformed into SQL databases.

The hotels are located across 17 states in the USA. The dataset includes 3 million user profiles: 2 million users identified their country, while 1 million did not. Among them, 1 million users are

from the USA, and the rest are distributed worldwide. Regions were categorized by continents, and we also experimented with the group of unknown users—analyzing how they affect the results when aggregated with other regions.

For the hotels, we collected data on amenities and environmental styles (e.g., decor, cozy, beach, mountain, luxury). These features were one-hot encoded and later simplified using Principal Component Analysis (PCA), assigning each amenity and style a specific number of components. This step reduced dimensionality while preserving the essential structure of the data.

Code architecture design



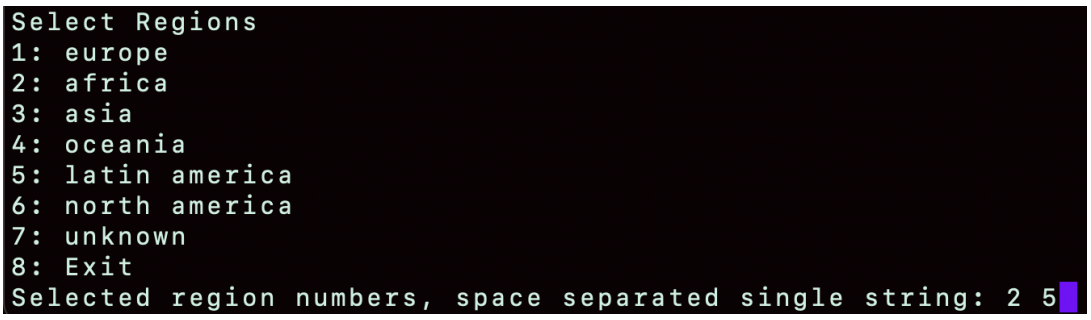
User Documentation

To run the program, enter in repository directory and follow the commands below:

1. Install python -> https://www.python.org/downloads/
2. **`python -m venv venv`**
3. Activate the environment:
   a. On Linux / macOS run: **source venv/bin/activate**
   b. On Windows (cmd): **venv\Scripts\activate**
   c. On Windows (PowerShell): **venv\Scripts\Activate.ps1**
4. **`pip install -r requirements.txt`**
5. Download the databases from the link: Google Drive Databases Public Link
6. Place the databases inside ../isp/app/Databases/databases
7. Go to ../isp/app and run **`python3 program.py`**
   a. Select the regions of the authors to test the recommender:
8. See hyperparameters usage by running **`python3 program.py --help`**
9. To use with Meta Centrum use r**un_app.pbs,** it is recommended to use a large machine.

To interact with the program, follow the instructions below:

1. Once the program has started, select the regions where the authors belong by picking numbers from the choices offered. The recommender will be trained only for those chosen regions.



2. The hotels are distributed throughout many states in the USA. Selects the number of the state where you are interested to get a recommendation. The recommender will be trained with all the hotels, but the recommendation will be focused on those particular states.

```
Select Hotel States
1: Pennsylvania
2: Indiana
3: California
4: Maryland
5: North Carolina
6: Arizona
7: Washington
8: Colorado
9: Ohio
10: Illinois
11: Florida
12: New York
13: Michigan
14: Tennessee
15: District of Columbia
16: Massachusetts
17: Texas
18: Exit
Selected state numbers, space separated single string: 12
```

3. Random options of authors are given, please pick an author. The recommendation will be personalized to that author.

```
Choose an author from the following list (generated randomly from
1: F8A8A1BCFC0656F23C884A51E10E6ED8
2: 9804596ECCA82917B0044CD31AFDB298
3: 9D4E92483C60C75A47C063EB99EE3BD5
4: 004384FD1DC0FB119B2E714F712115DC
5: 604E58B0E72570F9C12F286ADD83F9F2
6: Exit
Select author id:
```

4. The possibility to save the statistics about the trained model depends on whether a filename is inserted in this step.
5. The program returns previous reviews (at most 3) and top N recommendations together with the URL to visit its website in TripAdvisor, the price, and the state.

```
Preparing recommendations...

     Previous Hotel Reviews

State: California - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_R
t_Sunset_Cliffs-San_Diego_California.html?m=66827 - Rating: 1.0 - Price: $200.0

     Top Recommendations

State: New York - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_Rev
Hotel_By_Library_Hotel_Collection-New_York_City_New_York.html?m=66827 - Price: $260.0

State: New York - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_Rev
en_Inn_New_York_Tribeca-New_York_City_New_York.html?m=66827 - Price: $130.0

State: New York - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_Rev
l_Nyc-New_York_City_New_York.html?m=66827 - Price: $170.0

State: New York - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_Rev
el_by_Library_Hotel_Collection-New_York_City_New_York.html?m=66827 - Price: $450.0

State: New York - URL: https://www.tripadvisor.com/https://www.tripadvisor.com/Hotel_Rev
tel-New_York_City_New_York.html?m=66827 - Price: $1530.0
```

6. If the statistics were performed, in the output file located in app/pdfs/. There you will find:
    a. Distribution of authors based on their training size
    b. Distribution of how often hotels appear in the top 5 recommendations of an author.
    c. Distribution of the error depending on the training size of the authors
    d. MAE and RMSE of the hybrid predictions

Example of Recommendation

For Asia and New York state, author A1831F720E1EAE337C021CFB13560DA9 was selected (randomly).

The output of the program is the following:

Previous Hotel Reviews

State: Washington - TripAdvisor URL - Rating: 5.0 - Price: $270.0

Top Recommendations

State: New York - TripAdvisor URL - Price: $250.0

State: New York - TripAdvisor URL - Price: $2000.0

State: New York - URL: TripAdvisor URL - Price: $1560.0

State: New York - TripAdvisor URL - Price: $450.0

State: New York - TripAdvisor URL - Price: $220.0

Developer Documentation

1. Authors Database:

The **Author** table was constructed exclusively using data from the GitHub repository [1]. Rather than creating a separate table for ratings—which include their own IDs and contain many potentially useful attributes, such as text reviews for sentiment analysis—we chose to focus solely on the ratings themselves. Consequently, information from authors and their reviews was combined into a single class. This class contains details such as state, country, and region (or, more broadly, continent), as well as training_ratings, which is represented as a list of tuples in the form [(hotel_id, rating)].

For users who have rated at least two hotels, it is essential that there is at least one rating included in the training set. Once this first rating is assigned to training, the second (or last) rating is reserved for testing. This approach was determined to be the most effective given the sparsity of the dataset: approximately 1.8 million users have only 1–2 reviews distributed across 3,400 hotels in 17 states. Reserving one rating for testing is a standard technique in recommendation systems to evaluate model accuracy.

The **AuthorsDatabase** class is designed to manage author data efficiently while supporting a large-scale dataset of around 2 million users. It handles adding new authors from JSON files, maintaining a clean database, and providing queries that are directly used by the Application class.

Key features:

1. **Adding Authors**
    a. load_authors_from_json imports authors from a JSON file, avoiding duplicates by checking existing author IDs. Each author stores information such: as author_id, state, country, region, training_ratings, test_rating, accumulated_feature_vector, and accumulated_ratings. The region field is determined using a provided mapping from countries to regions.

2. **Data management**
    a. delete_author removes an author from the database and returns the list of hotel IDs affected
    b. delete_hotel_from_ratings removes a specific hotel from all relevant authors' training and test ratings.
    c. clean_database resets the database, dropping and recreating all tables.
3. **Memory efficient data retrieval**
    a. get_authors_by_regions retrieves authors in a batch-wise manner, improving performance when querying large datasets.
    b. get_all_authors_ids_by_region_and_min_training filters authors based on region and a minimum number of training ratings. Increasing the minimum number of training ratings reduces the number of eligible authors, which can impact model training.
    c. stream_authors provides an iterator over authors, allowing the program to process large datasets without loading all data into memory at once. This is particularly useful given the size of the dataset (~2 million users).
4. **Other useful queries**
    a. get_all_authors_ids returns all author IDs in the database.
    b. get_author_by_id retrieves a single author by ID.
    c. get_rated_hotels_from_author returns all hotels rated by a given author, including the test rating.
    d. authors_id_training_rev_by_regions retrieves author IDs along with their training ratings filtered by region.

The class uses **SQLAlchemy** with PickleType and MutableList to store variable-length lists like training_ratings and accumulated_feature_vector. Functions are optimized for **memory efficiency** by using batch processing and generators (yield) to handle large datasets.

2. Hotels Database:

In the previous section [Data], we noted that the Tripadvisor API was used to supplement missing information from the repository, such as prices. The API provided additional features crucial for this project, including subratings (scores for cleanliness, sleep quality, room service, price, etc.), which reveal user preferences for specific aspects. Other valuable additions were hotel styles—about 50 distinct types—and trip types, indicating whether a hotel is typically chosen for couples, families, friends, solo travelers, or business visits.

Although the price category ($ - $$$$$) was obtained, it was observed that the price for the same category had a considerable difference. After attempting to scrap the price situated just above the images for each hotel and running into troubles due to captcha verification then manual scrapping was performed and most of the hotels were kept with an estimated price. It was discovered that a considerable number of hotels were permanently closed, and so it was needed to search for their price range through old reviews.

All that data was combined into a single JSON file and processed by Class **HotelsDatabase** which handles the management of **Hotel** data**.**

Key features:

1. **Adding hotels and creating its feature vector.**

    a. **load_hotels_from_json: I**mports hotels from a JSON file, avoiding duplicates by checking hotel IDs. Since feature vectors depend on PCA components (computed across all hotels), they cannot be set at construction if the database is new. To handle this, two steps (b) and (c) follow immediately after loading.
    b. **update_means_and_stds:** Collects prices, rooms, and ratings across hotels, computes global means and standard deviations, and saves them via joblib for later z-score normalization.
    c. **calculate_pca_components:** Builds one-hot matrices for amenities and styles using MultiLabelBinarizer, then compresses them with PCA into lower-dimensional feature spaces (pca_amenities, pca_styles).

    Feature vector creation is managed by:

    d. **__set_feature_vector**: Updates PCA components if new amenities or styles appear and triggers feature vector construction.
    e. **__create_feature_vector**: Normalizes rooms, prices, stars, and ratings using z-scores. For trip categories (couple, family, business, friends), it computes proportions (trips per category / total trips), producing a distribution that sums to 1. Gathers normalized features and PCA to create the feature vector later on used for content-based predictions.

2. **Data management**

    a. **delete_hotel, delete_author_from_hotels** and **clean_database** were added for consistency.

3. **Queries:**

a. **info_for_reviews:** It is used in Application Class. It returns for each hotel the id, the feature vector and a list of authors that have rated that specific hotel. The idea is that if a hotel is deleted from the database, the training data and test data of the users that rated that hotel should be updated by removing the rating to that hotel.
b. **author_rated_to_hotels:** Returns the list of hotels that rated a specific hotel.
c. **get_hotels_ids_from_regions:** Returns hotels based on a given list of states.
d. **get_hotels_feature_vector:** Return all hotels feature vectors.

3. Hybrid Recommender

3.1 Content Based Recommender

People often have personal tendencies when evaluating items: some give consistently high ratings, while others are more critical and give lower scores. For example, one user might rate all movies 4's and 5's, while other rates the same movies 2's and 3's. Even though they enjoy the same kinds of movies, their rating profiles appear very different. To address this, we subtract each user's average rating from their individual scores. This normalization highlights what each user likes more or less compared to their personal baseline, rather than comparing raw scores. This idea was taught in Standford lectures [2].

However, due to the sparsity of the rating data, normalizing in this way sometimes led to all ratings becoming 0 for certain users. This happened when a user gave the same score to every item they rated. To handle this, Doc. RNDr. Iveta Mrázová suggested adding a small amount of noise to the ratings before normalization. Choosing random noise in the range [0, 0.2] was sufficient to avoid this issue.

In a content-based recommender system, the goal is to build feature vectors (as explained in Section 2: Hotels Database) and then compute a user profile vector. This is done by taking a weighted average of the normalized ratings combined with the feature vectors of the hotels rated. The weights reflect how much the user liked each hotel, allowing the model to capture which features matter most to them.

Because PCA is applied to the feature vectors, each user profile contains hidden information about their preferences. This means two users can end up with very similar profile vectors (small Euclidean distance), even if the hotels they rated are not the same, as long as their preferences align in the latent space. This idea is used later on for User-Based Collaborative Filtering.

Key functions:

1. **__calculate_similarity_matrix:** Receives an iterator of authors which is key to reduce memory usage, a boolean value similarity_function set to false for default usage of Cosine Similarity, otherwise Pearson Correlation is used. Uses the function below to get the user profile and uses the selected similarity function to calculate the predictions.
2. **_build_author_profile:** Calculates the author profile adding the small noise to the ratings, subtracting the author average rating and summing the weighed hotel feature vectors.

3.2 Data Augmentation using User-Based CF + Funk MF (SVD)

Paper [3] presented the idea of using SVD predictions to evaluate an item-based recommender using the columns of the latent matrix for items to calculate the nearest neighbors and later use those and the latent matrix for users to make the predictions. While this approach was not suitable for this project, the idea of using some techniques to do early predictions and refine them with the second technique was very useful.

As was commented in the previous section, the user profiles contain hidden information hence since the main interest of this project is to analyze how grouping users affect the prediction's user-based collaborative filtering seemed appropriate. The idea proposed is to use a User-Based Collaborative Filtering that given some similarity threshold, if the similarity is above, it then with some given probability the rating and the neighbor are added to the train dataset or not.

Probability was incorporated using Nature-Inspired Algorithms techniques to add ratings in a controlled way, avoiding excessive influence on the predictions of SVD. Realistically, even if people visit a hotel and like it, they might or might not leave a rating. Diversity also improves, as hidden factors in the user profile can create neighbors that will bring novelty to the user.

SVD updates a hotel's hidden latent vector based on all users' ratings, so grouping users differently can significantly impact SVD's results. By grouping users according to their region, we can investigate whether regional patterns improve prediction accuracy. Observing how the latent factors change for these groups may help enhance personalized recommendations, as the model can capture regional preferences more effectively.

Key functions:

1. **_generate_neighbor_pseudo_rating:** Uses NearestNeighbors to model the author profiles. For each author–hotel pair, it iterates through the author's neighbors. If a neighbor has rated the hotel and has a similarity above 0.7, the rating is added to a temporary list (list_) with a 0.3 probability, and the neighbor's similarity is added to a corresponding weight_ list. After exploring all neighbors, the method computes a

weighted average of the ratings in list_ using weights_. This weighted average is then added as augmented data for training the SVD model.

2. **train:** Prepares the data for the computation of content based and SVD. Uses stochastic gradient descend with MSE and L2-regularizatin formula for SVD which is taken from [4] article, see derivation of SGD.

3.3 Prediction

There are two functions **predict_selected_hotels_for_author** which receives an author and a list of hotels or **predict** for a single author-hotel pair. Both combine the results of content-based and SVD by some specific weighted, default is 0.5 for each.

4. Statistics

Class **Recommender_Statistics** takes the recommender as an input and uses the function **predict_selected_hotels_for_author** because of the matrix multiplication fast prediction than the iterative approach to predict for each user one by one ech hotel. By a factor of 0.3 the author is evaluated or not, this way we sample the population and shorten the computational time. Then prepares the following statistics:

1. **__get_author_training_size_distribution:** On the x axis there is the distribution of authors given the training data size and on the y axis is the number of authors per group, this allows us to see how imbalanced/balanced the dataset was.
2. **__get_prediction_top_hotels_distribution:** For each hotel, it counts how many times it appeared on the top 5 recommendations, then as a pie chart it is shown by the percentage of users how diverse the recommendations were. It indicates the diversity of the recommendations which is an important factor since the population might agree with some top regional popular items, but recommendations should provide new insights to users.
3. **__get_accuracy_per_authors_cluster_by_training_size:** Distribution of the error grouped by the training size of the users. If the error decreases consistently as training size increases, it suggests that the model learns to generalizable patterns from more data rather than memorizing noise.
4. **__evaluate**: calculates MSE and MAE.

5. Application

This class controls the databases, queries the necessary data, handles the recommender and the statistics. Most functionalities related to the mentioned classes controlled by Application were mentioned already, managed by Application. The addition of ratings is

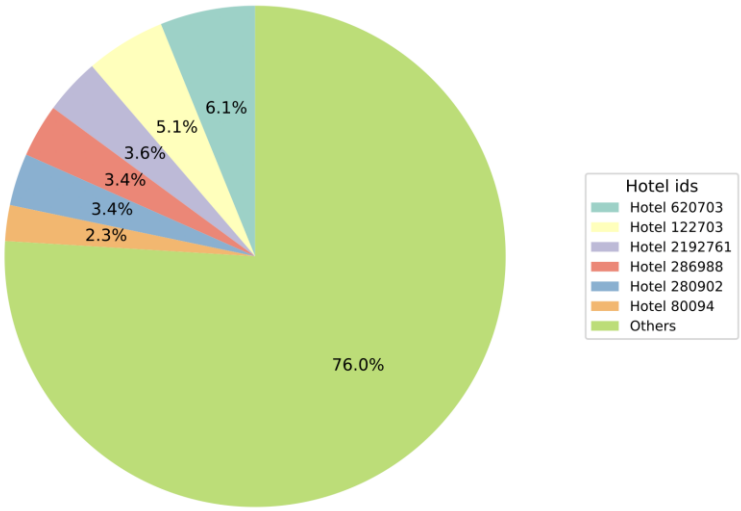
performed by two big functions **insert_reviews** and **__process_current_author.** The reviews JSON file should be ordered by author_id, hence a chunk of reviews to the same user is gathered and given to **__process_current_author.**

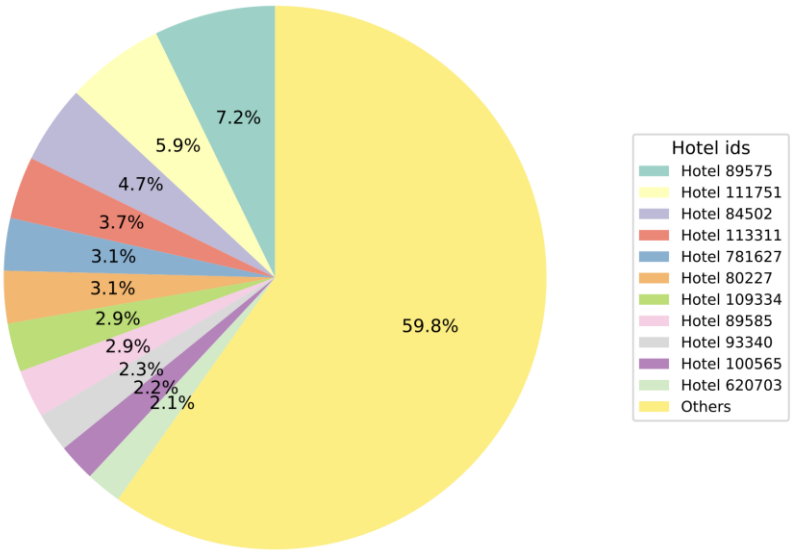
Experiments and Conclusions

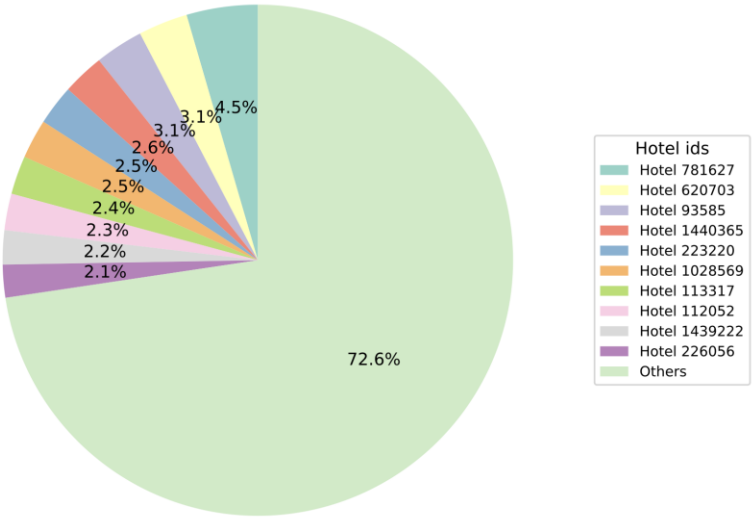
Using cosine similarity as the similarity matric, since it gave me equal results as Pearson Correlation, 0.5 weight for both content based and SVD recommenders, PCA for amenities with 40 components and for styles 20.

We can observe that in Europe the error was MAE = 0.881 RMSE = 1.055, for Africa we obtained MAE = 0.779 RMSE = 0.959 and for Asia MAE = 0.925 RMSE = 1.108. Now joining all those zones together metrics were MAE = 0.884 RMSE = 1.061. RMSE was not affected by much, but MAE improved by a small interval. For the population with unknown regions, which are 1 millions of authors, we obtained MAE = 0.923 RMSE = 1.094, not too far away in comparison with the population Euro-Asia-Africa. There is no clear insight with the current experiment that grouping people by bigger regions would lower the quality of the recommendation.

Moreover, we could also analyze the distribution of the hotels being recommended on the top 5 to the users in different regions, we find that for most of the experiments there wasn't a specific group of hotels that was recommended to majority of the population within that region, this means that there is diversity on the options user have and as expected, hotels are chosen based on the preferences of users which could have some common ground but are differ from person to person. The results below correspond to Latin America, Europe above and Asia, in descending order.


6.1%
5.1%
3.6%
3.4%
3.4%
2.3%
76.0%
Hotel ids
Hotel 620703
Hotel 122703
Hotel 2192761
Hotel 286988
Hotel 280902
Hotel 80094
Others


7.2%
5.9%
4.7%
3.7%
3.1%
3.1%
2.9%
2.9%
2.3%
2.2%
2.1%
59.8%
Hotel ids
Hotel 89575
Hotel 111751
Hotel 84502
Hotel 113311
Hotel 781627
Hotel 80227
Hotel 109334
Hotel 89585
Hotel 93340
Hotel 100565
Hotel 620703
Others


4.5%
3.1%
3.1%
2.6%
2.5%
2.5%
2.4%
2.3%
2.2%
2.1%
72.6%
Hotel ids
Hotel 781627
Hotel 620703
Hotel 93585
Hotel 1440365
Hotel 223220
Hotel 1028569
Hotel 113317
Hotel 112052
Hotel 1439222
Hotel 226056
Others

We can conclude that the distribution of the hotels' predictions is well distributed, not causing a cluster of the same hotels always recommended. As for the errors, given that the minimum error is 0 and the maximum is 5, the results are not satisfactory since the competitive models have metrics that oscillate 0.7 but also are not far from the goal aimed.

Future Work

- Implement hyperparameter tuning techniques which could help to minimize the error and achieve convergence faster.
- Implement an automatic feature which detects which hidden factors are common for users of specific regions and apply some weight to enforce preference and compare results.
- With the final chosen hybrid recommender model, return the current rank of a hotel within some region.
- Explore knowledge and social based recommendation systems, and how do they compare results with the current project.
- Iterative processes in recommender systems may perform more effectively when the input ratings or information are randomly shuffled, rather than consistently presented in the same order, this could return interesting results for Funk MF.
- Evaluate how well each hotel category is represented and recommended.
- Apply stratification by selecting an equal number of representatives from each hotel group (e.g., star rating, overall rating, or price), allowing reduction in the number of selected authors and incorporating data augmentation techniques.

Bibliography

[1] Behun, M. (2022). TripAdvisor data-scraper. GitHub. https://github.com/

[2] Ullman, J. D. (n.d.). Recommender Systems. Stanford University. Chapter 9.2.5 . Available at lecture notes.

[3] S. Gong, H. Ye and Y. Dai, "Combining Singular Value Decomposition and Item-based Recommender in Collaborative Filtering"

[4] Insight (2006) Explicit Matrix Factorization: ALS, SGD, and All That Jazz. link to the website