final_report.md

Introduction

In this report, I present an item-to-item collaborative filtering recommendation system based on centered cosine similarity (Pearson correlation). Also, baseline estimate used for improving model prediction quality. Collaborative filtering is a popular technique for making personalized recommendations by leveraging user-item interactions. The code implements a recommendation model using Nearest Neighbors with cosine similarity and incorporates a baseline estimation approach to predict user-item ratings.

Data analysis

Overview of the data

During the data overview, I used only `u.data`, `u.item`, `u.user`. For `u.data` and `u.item` data, I get the films with the highest average rating:

	itemID	movie title	rating	number of ratings
813	814	Great Day in Harlem, A (1994)	5.000	29
1598	1599	Someone Else's America (1995)	5.000	1
1200	1201	Marlene Dietrich: Shadow and Light (1996)	5.000	8
1121	1122	They Made Me a Criminal (1939)	5.000	10
1652	1653	Entertaining Angels: The Dorothy Day Story (1996)	5.000	1
1292	1293	Star Kid (1997)	5.000	6
1499	1500	Santa with Muscles (1996)	5.000	2
1188	1189	Prefontaine (1997)	5.000	8
1535	1536	Aiqing wansui (1994)	5.000	2
1466	1467	Saint of Fort Washington, The (1993)	5.000	3

During analyzing `u.user`, I calculate some statistics:

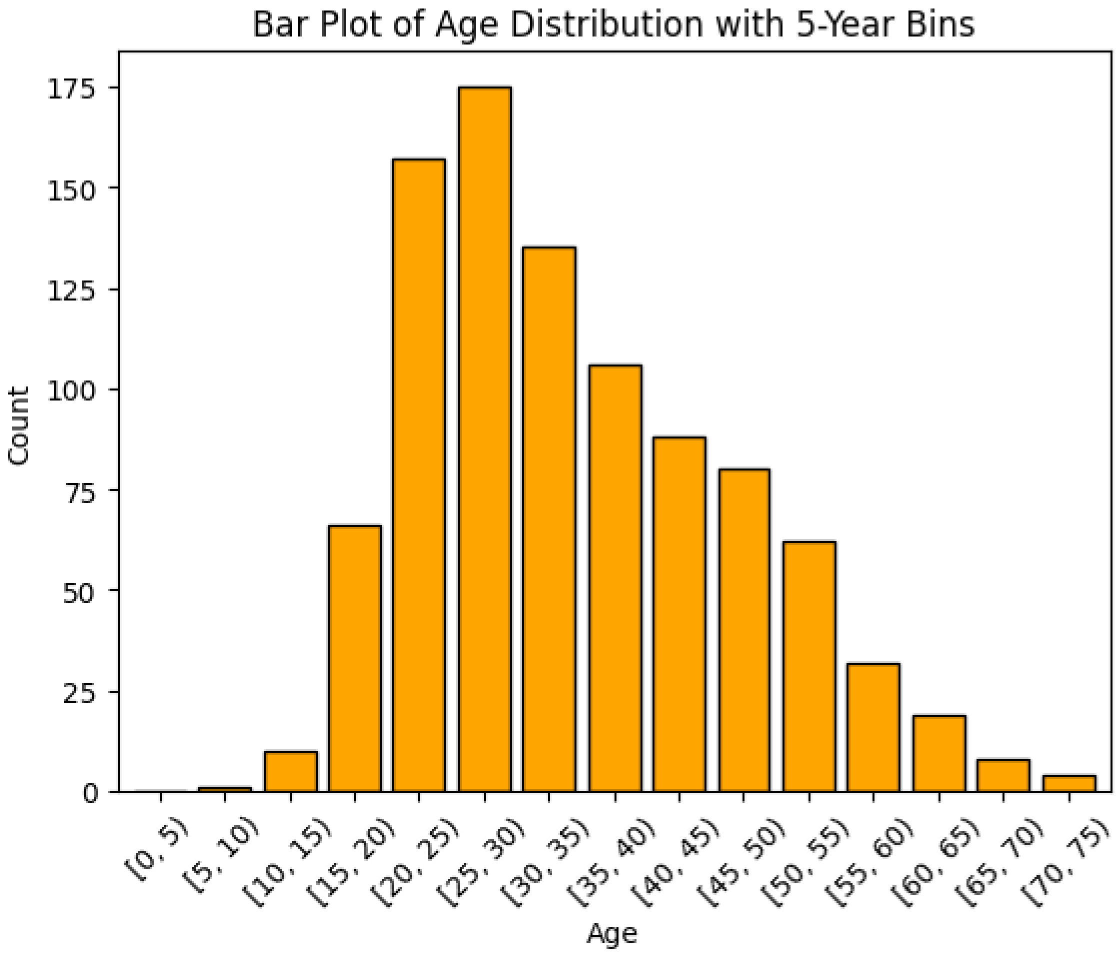
- **Occupation** distribution among the users:

```
administrator    79
engineer         67
programmer       66
librarian        51
writer           45
executive        32
scientist        31
artist           28
technician       27
marketing        26
entertainment    18
healthcare       16
retired          14
lawyer           12
salesman         12
none              9
homemaker         7
doctor            7
```

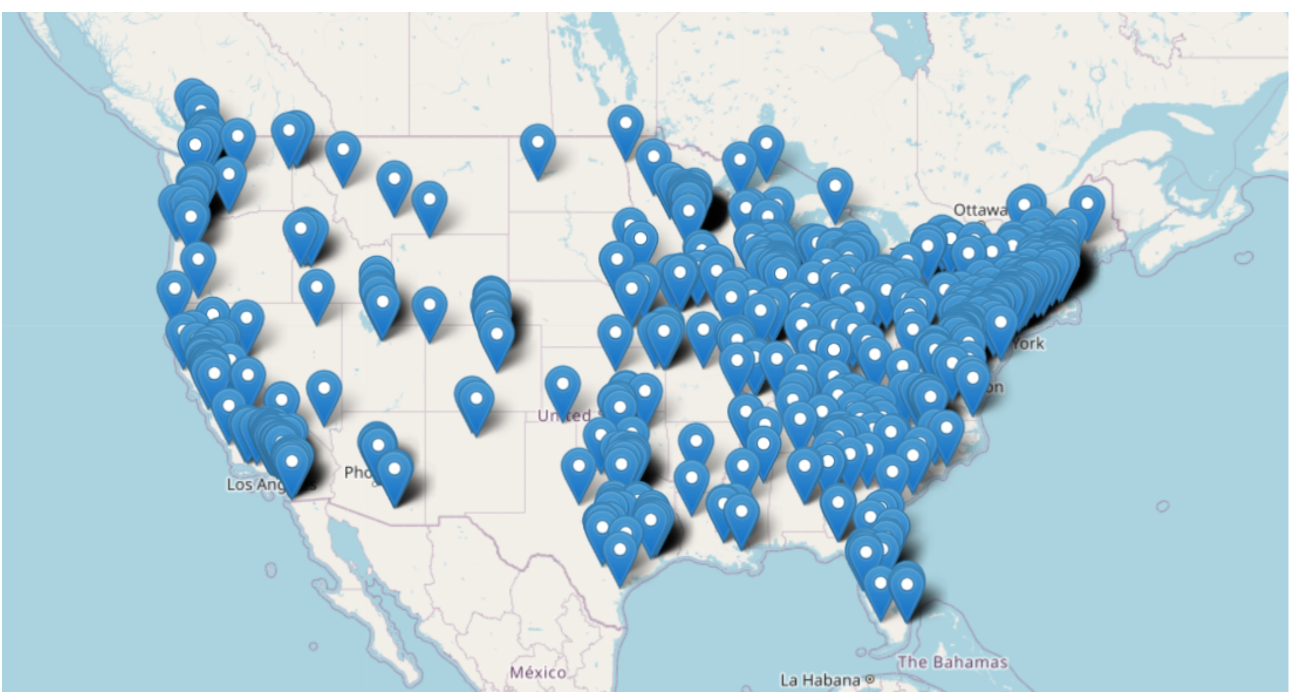
- **Gender** distribution among the users:

Gender	Num
Males	670
Females	273

- **Age** distribution among the users:



- And finally **geographic** distribution among the users:



Data preprocessing

I do all data preprocessing in `movie_cosine_similarity.ipynb`. I use only `u.data` data.

Storing item-to-user matrix is not efficient method. So, I decide store data as list that contains the lists of items that user rate and ratings.

It looks like this:

```
item_to_user = [
                 [(some_rating_of_user_0, some_item_idx_for_user_0), ...],
                 [(some_rating_of_user_1, some_item_idx_for_user_1), ...],
                 ...
               ]
```

Index of this list represent some user and list that store at this index is user ratings. These ratings just tuple pairs with user rating and item ID.

After that I split users into two groups. One group I use to do prediction.

Other group I use to test my model. For each user from test group, I pick randomly **min_num_of_items_that_user_rate** ratings and just make left ratings zero.

Note: **min_num_of_items_that_user_rate** is variable from `movie_cosine_similarity.ipynb`

Model Implementation

The collaborative filtering model is implemented in the `CF_using_cosine_similarity.py` in CFUsingCosineSimilarity class. The model utilizes a Nearest Neighbors approach with centered cosine similarity to identify similar items. Based on this similar items, model can predict ratings for user. Also, during prediction baseline estimate used.

Method **fit** of model involves preparing user-to-item ratings and item-by-user ratings lists, calculating some values that helps faster find baseline estimate values during prediction, and precalculating item neighbors and distances. But you should pass in fit method data in the following format:

```
[
  [(some_rating_of_user_0, some_item_idx_for_user_0), ...],
  [(some_rating_of_user_1, some_item_idx_for_user_1), ...],
  ...
]
```

After the model is fitted, you can use the 'predict' method, which returns the predicted ratings for each item as a numpy array. However, to use this method, you need to pass user preferences (ratings) to the predict method. The user preference is a list that should contain items that the user has already rated, formatted as follows:

```
[(rating, item1), (rating, item2), ...]
```

The prediction process combines baseline estimates with weighted average of ratings based on its centred cosine similarities. This formula is used:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$b_{xi} = \mu + b_x + b_i$$

- μ = overall mean movie rating
- b_x = rating deviation of user x
  = (*avg. rating of user* x) $- \mu$
- b_i = rating deviation of movie i

Where:

- r_{xi} is rating prediction for item 'i' by user 'x'.
- $N_{(i;x)}$ is items that similar to item 'i' and rated by user 'x'
- s_{ij} is centered cosine similarity

Model Advantages and Disadvantages

Advantages:

1. Utilizes item-to-item collaborative filtering, capturing item similarities.
2. Incorporates baseline estimates for improved prediction rating.

Disadvantages:

1. Only use information about the user's preferences but not information about user themself.

Training Process

In fact, there is no training process for this model. But there is fitting process that similar to training. During fitting, model does the following:

1. Initialize Data Structures

- The method initializes two main data structures: `user_items_list` and `item_users_list`.
- `user_items_list` is a list of lists, where each inner list contains tuples representing the ratings given by a user to different items.
- `item_users_list` is initialized as an empty list for each item. It will be populated with tuples representing the ratings given to that item by different users.

2. Calculating user-item interaction matrix

- `mtr` : A 2D array representing the user-item interaction matrix

3. Calculating some values that will be used when we will calculate baseline estimation

- `mu` : overall mean rating
- `self.mean_items_rating` : list that contains mean rating for each item

4. Build matrix user-item interaction that

- It updates the user-item interaction matrix (`mtr`) by subtracting the mean rating from each rating to center the values.

5. Fit k-Nearest Neighbors Model

- The method uses the scikit-learn `NearestNeighbors` class to fit the k-Nearest Neighbors model.
- It uses the user-item interaction matrix (`mtr`) and cosine similarity as the metric.

6. Precalculate Neighbors

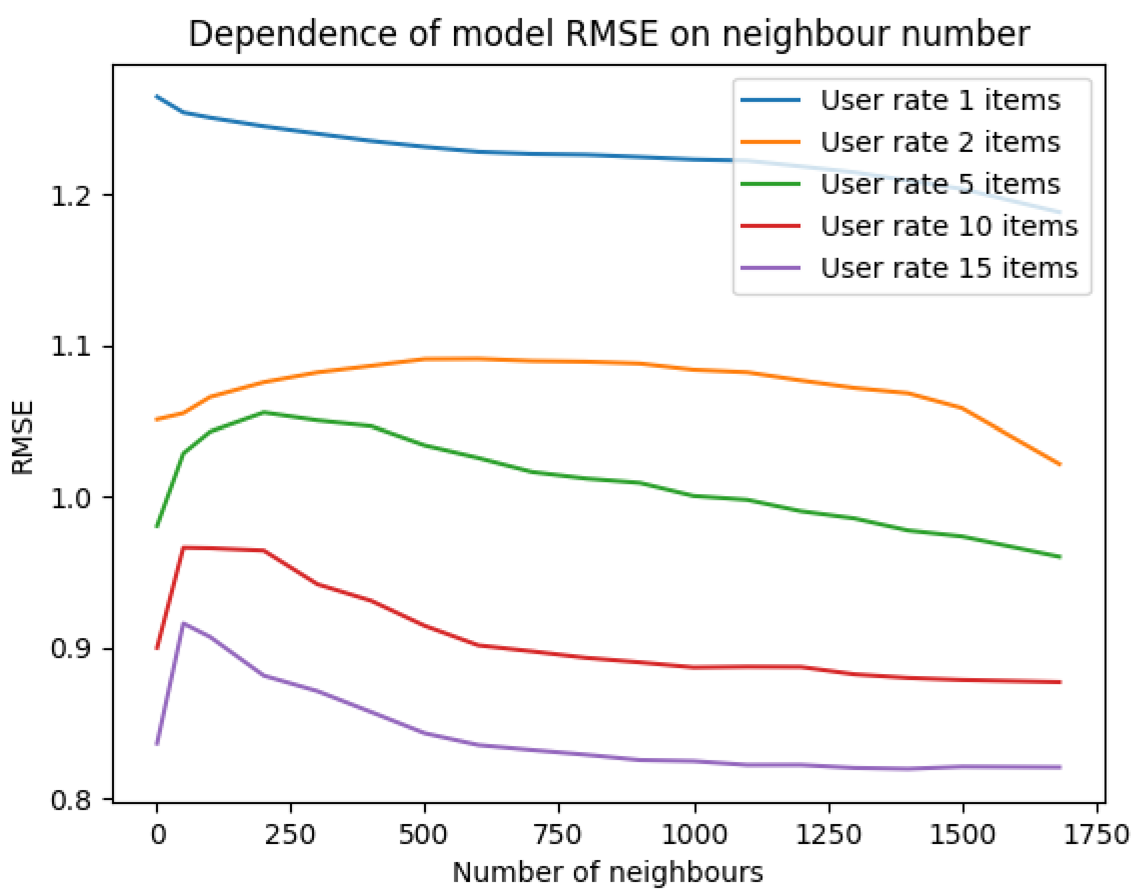
- The method calls a `precalc_neighbors` function to calculate and store the nearest neighbors for each item along with their distances.

Evaluation

The evaluation of the model is based on Root Mean Squared Error (RMSE). A function, `calc_avg_RMSE` , is defined to calculate the average RMSE for a given set of test data. The report evaluates the model's performance using different numbers of neighbors, ranging from 1 to 1682. Where 1682 is number of all different items.

Results

The evaluation the model's performance using different numbers of neighbors:



Conclusion

Based on the graph above, we can make three observations:

1. The more the model knows about user preferences, the better it predicts ratings (Obviously)
2. model works better when number of neighbours is maximum possible value.
3. strange rapid growth of the RMSE for the small number of neighbours.

From the second and third observations, we can conclude that there is no strong correlation between items. Therefore, it is important for models to gather information about as many items as possible so that the weighted average formula calculates predictions more accurately.