POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Analysis for ML

Author: **Mattia Piccinato**

Academic Year: 2023-24
Link to the repository: Click here

Contents

1 | Scope

1.1. Introduction

The referenced paper introduces a movie recommendation system called **MOVREC**. This system employs a **collaborative filtering approach**, using user-provided information to analyze and recommend movies **without personalization**.

The recommended movie list is generated using the **K-means algorithm** to cluster the movies in the dataset. It includes movies belonging to the cluster with the **highest average rating** and sorts them based on their individual ratings.

The paper addresses the common challenge individuals face since the web appearance: an overload of choices. In response, companies deploy recommendation systems to assist users by **suggesting a small cluster of movies, rather than the entire dataset**.

1.2. Aim

As the paper did not report any results, the identified scope for this project was to **implement the recommender system** in order to verify its functionality.

2 | MOVREC Main Aspects

2.1. Collaborative Filtering

Collaborative filtering systems recommend items based on similarity measures between users rather than items, providing serendipitous recommendations as they rely on user similarity rather than item similarity.

In particular, MOVREC implements a straightforward approach that aims to recommend **items preferred by the majority of users**.

2.2. Explicit Rating

Explicit ratings provided by users serve as a crucial input for predicting their movie choices.

Among the attributes of each movie, namely directors, actors, year, genre, and average rating, the latter represents the average of all ratings given by users. **The system leverages the average rating** to suggest items that are most likely to be enjoyed.

2.3. Data Filtering

Data filtering refers to the process of selectively extracting or excluding data from a dataset based on specific criteria or conditions. The goal of data filtering is to focus on relevant information or remove unwanted elements, making it easier to analyze or visualize the data.

Users are given the choice to select values for different attributes. Based on these input values, the system computes an array of suitable movies, with **at least one attribute value** that matches the input value of the user. Then, **only the top 20 movies** by rating are considered, prioritizing those that received more ratings.

2.4. K-Means Algorithm

In the K-means clustering technique, K initial centroids are chosen and assigned to the cluster with the nearest centroid. The centroid of each cluster is then updated, and the process is repeated until convergence. It aims to minimize the following objective function:

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} d(x_j, c_i)^2 \tag{2.1}$$

Here, n is the number of cases, k is the number of clusters, d is a distance measure, c_i is a cluster center, and x_j is a data point.

The system utilizes the K-Means Algorithm to form **moderately-sized clusters** of movies. This approach helps with presenting a concise list of recommended movies, **belonging to the cluster expected to better predict user choices**.

2.5. Movies Weights

To compute the average rating of a cluster, the system assigns different weights to each movie m. The weight W_m of each movie is computed as the sum of the weights given to the value of their f attributes.

$$W_m = \sum_{j=1}^{f} w_j \tag{2.2}$$

For the average rating, this is the table of the possible weights:

Rating	Weight		
	If number of votes <=1000	If number of 1000< votes <=10000	If number of votes >10000
10	10	20	30
9	9	18	27
8	8	16	24
7	7	14	21
6	6	12	18
5	5	10	15
1-4.9	1	2	3

While the weights for the other attributes' values are defined as the frequency of that value

along the entire dataset (for the year, for the actors, for the directors, and for the genre).

$$w_j = \frac{\text{count of movies with } F_j = f_j}{\text{total number of movies}} \tag{2.3}$$

Overall, the average rating for the cluster is computed, giving **more importance to movies with a higher rating**, thus choosing clusters with a higher percentage of "good movies," as well as giving priority to those movies with **common features**.

3 | Methodology

3.1. Dataset Cleaning

3.1.1. Selected Dataset

As mentioned in the paper, the dataset was obtained from **www.imdb.com** due to its extensive collection of movies and ratings from a large number of users.

3.1.2. Operations on the Dataset

To create a dataset with information on actors, directors, genre, year, and average rating, **three different datasets were merged**. Unnecessary columns were **dropped** from each, and **rows with null values were removed**.

Note that the column containing the number of ratings was retained to filter the best 20 movies before the clustering process.

3.1.3. Weights Pre-processing

The movie weights were then **pre-processed** and added as a new column in the dataset.

3.1.4. Frameworks and Technologies

This section outlines the operations performed to obtain the dataset in *get_datasets.ipynb*.

Pandas dataframes were used to perform these operations efficiently and store the data in the *dataset.csv*.

3.2. Workflow

3.2.1. Filter Function

The initial implementation involved the **filter function**.

As stated in the paper, conditions on movie attributes can be specified, and the function returns the **entire dataset dataframe** if no condition is defined. Otherwise, it returns the **dataframe containing movies that satisfy at least one** condition.

3.2.2. Recommender Function

Next, the **recommender function** was implemented.

The first task was to choose an **appropriate number of clusters**. The default number is assumed to be 4 to return a recommendation cluster containing an average of 5 movies. However, if the filtered list contains fewer than 20 movies, the number of clusters is proportionally resized.

```
# Resize the number of clusters based on the number of movies in the input list if there are less than 20 movies
# Otherwise keep the 20 movies with highest average rating, giving priority by number of ratings
if len(movies_df) < 20:
    n_clusters = math.ceil(n_clusters * (len(movies_df)/20))
```

Figure 3.1: Resizing the number of clusters.

The second task involved **selecting only the best 20 movies** based on average rating and number of ratings if the filtered list still contains more than 20 movies. After this selection, the column representing the number of ratings for each movie was dropped from the dataframe.

```
else:
    movies_df = movies_df.sort_values(by=['averageRating', 'numVotes'], ascending=False).head(20)
```

Figure 3.2: Select the top 20 movies.

At this point, the **K-Means Algorithm** was implemented. Although Euclidean distance is sensitive to categorical data, it was chosen as per the paper's recommendation. To represent categorical data, **One-Hot encoding** was used for sets of directors, authors, and genres, and the Hamming Distance was chosen to calculate the distance along each "axis".

```python
# Compute the 1D distances for the averageRating and startYear columns
for j in range(2):
    tempDistances.append(movies_df.iloc[i][col_names[j]] - centroids[c_idx][j])

# Compute the 1D one-hot tempDistance for genres, directors and authors as well
for j in range(2, num_of_columns):

    # By summing 1 for each value in movies_df[col_names[j]][i] not in centroids[c_idx][j].keys()
    tempBoth = sum([1 for x in movies_df.iloc[i][col_names[j]] if x not in centroids[c_idx][j].keys()])
    # Summing (1 - centroids[c_idx][j][x]) for each x both in movies_df[col_names[j]][i] and in centroids[c_idx][j].keys()
    tempMovie = sum([1 - centroids[c_idx][j][x] for x in movies_df.iloc[i][col_names[j]] if x in centroids[c_idx][j].keys()])
    # Summing centroids[c_idx][j][x] for each x in centroids[c_idx][j].keys() and not in movies_df[col_names[j]][i]
    tempCentroid = sum([centroids[c_idx][j][x] for x in centroids[c_idx][j].keys() if x not in movies_df.iloc[i][col_names[j]]])

    # And then appending the overall sum to the 1D distances list
    tempDistances.append(tempBoth + tempMovie + tempCentroid)

# Compute the Euclidean distance between the movie i and the centroid c_idx
distance = math.sqrt(sum([x**2 for x in tempDistances])) # [3]
```

Figure 3.3: Euclidean distance.

Finally, the **best cluster** was chosen based on the weighted average rating of the cluster.

```python
# ---- Pick best cluster ---- #

clusters_dict = {}

# Compute the weighted average movie rating for each cluster
# Each cluster contains at least one movie with non-zero weight, so we can safely compute the weighted average
for idx, cl in enumerate(clusters):
    weights_sum = 0
    clusters_dict[idx] = 0
    for i in cl:
        weights_sum += movies_df.iloc[i]['weight']
        clusters_dict[idx] += movies_df.iloc[i]['averageRating'] * movies_df.iloc[i]['weight']
    if weights_sum != 0:
        clusters_dict[idx] /= weights_sum

# Return the cluster with the highest weighted average movie rating
return movies_df.iloc[clusters[max(clusters_dict, key=clusters_dict.get)]].sort_values(by='averageRating') # max() returns the first key with the highest value
```

Figure 3.4: Selection of the best cluster.

3.2.3. Frameworks and Technologies

This section illustrates the implementation in *implementation.ipynb*.

Pandas dataframes were used to facilitate these operations.

4 | Results and Conclusions

As stated in the paper: *"Given the nature of our system, evaluating performance is challenging since there is* ***no definitive right or wrong recommendation****; it is purely a matter of opinions."*

Therefore, only a couple of functionality tests were performed. These include verifying that the recommendations are **deterministic** and exploring the system's response to different filtering choices.

Notably, **filtering by low values for the rating field is often ineffective in combination with other filter options**.