

VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF
TECHNOLOGY

—oo0oo—



LINEAR ALGEBRA

PROJECT 7: NORM, ANGLES AND YOUR MOVIE CHOICES

Semester: HK242 **Class:**

Lecturer: Dr.Đậu Thế Phiệt

Group:

Full Name	Student ID	Contribution
Nguyễn Hữu Sang	2114636	100%
Nguyễn Thụy Khánh Linh	2052576	100%
Hồ Huỳnh Minh Khoa	2352558	100%
Nguyễn Đỗ Thịnh Phát	2352886	100%

Contents

I. INTRODUCTION	3
II. THEORY	4
1. Norm	4
2. Angle Between Vectors	4
3. Pearson Correlation Coefficient	5
III.PROCEDURE	6
1. Load data	6
2. Print the titles	6
3. Select people	8
4. Find the Euclidean distance	9
5. Find the Pearson correlation coefficient	10

List of Figures

List of Tables

I. INTRODUCTION

The ability to discern and leverage user preferences has become a commercially significant endeavor, catalyzing substantial advancements in recommendation systems across prominent digital platforms such as Spotify, YouTube, and Shopee. These platforms invest heavily in sophisticated algorithms to meticulously analyze user behavior, thereby enabling the delivery of highly personalized product or content suggestions. For instance, Spotify curates custom playlists based on individual listening patterns, while YouTube utilizes machine learning to recommend videos aligned with a user's prior viewing history. This report aims to explore the underlying patterns in user preferences by applying linear algebra techniques to the extensive MovieLens dataset, which contains millions of ratings from a diverse user base. Through a comparative analysis of these ratings, this report will develop a recommendation framework designed to suggest items to users based on the preferences of those with similar tastes, employing established similarity metrics such as Euclidean distance, scalar product and Pearson correlation.

II. THEORY

1. Norm

Concept: A norm is a measure of a vector's "magnitude" that always returns a non-negative value. The zero vector has a norm of 0. Norms are widely used in data analysis and optimization.

Types of Norms:

- **Euclidean Norm (L_2):** Measures geometric distance between points in space:

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- **Manhattan Norm (L_1):** Sum of absolute values, useful for optimization:

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

- **Maximum Norm (L_∞):** The largest absolute component value:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

- **p-Norm:** Generalization of norms with parameter p:

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p} \quad (p \geq 1)$$

2. Angle Between Vectors

Purpose: The angle between two vectors indicates their similarity:

- Small angle \rightarrow similar direction.
- Large angle \rightarrow different direction.

Formulas:

- Dot Product:

$$u \cdot v = \sum_{i=1}^n u_i v_i$$

- Vector Norm:

$$\|u\| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}$$

- Angle Calculation:

$$\cos \phi = \frac{u \cdot v}{\|u\| \|v\|}$$

3. Pearson Correlation Coefficient

Interpretation: Measures linear correlation between datasets (-1 to 1):

- -1: Perfect negative correlation.
- 0: No correlation.
- 1: Perfect positive correlation.

Formula:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

III. PROCEDURE

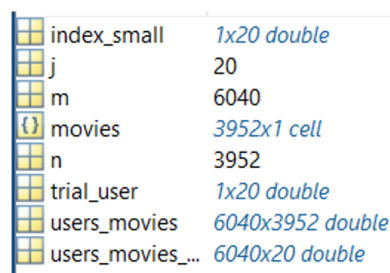
1. Load data

```

1 % Clear the workspace
2 clear;
3 % Load the user_movies.mat file
4 load('users_movies.mat', 'movies', 'users_movies', 'users_movies_sort', '
    index_small', 'trial_user')

```

First, we load the data from the file “users_movies.mat” using the “load” command. “load” helps us to load actual data stored in those variables shown below while “clear” is used to clear all variables in MATLAB workspace to ensure that no old data interferes. The matrix “users_movies” should have dimensions 6040×3952 , with integer values ranging from 0 to 5. A rating of 1 represents “strongly dislike”, while 5 represents “strongly like”. A rating of 0 indicates that the user did not rate the movie. The array “movies” contains the titles of all the movies. Moreover, the matrix “users movies sort” is a subset of “users_movies”, containing ratings for the 20 most popular movies.



index_small	1x20 double
j	20
m	6040
movies	3952x1 cell
n	3952
trial_user	1x20 double
users_movies	6040x3952 double
users_movies_...	6040x20 double

Also, the indexes of these popular movies are stored in the array “index_small”. Finally, the vector “trial_user” contains ratings of these popular movies by another user who is not part of the database. It is recommended to view all variables and their dimensions using the “Workspace” window in the MATLAB environment.

2. Print the titles

The figure below shows the 20 most popular movies:

The rating is determined based on the 20 most popular movies:

1. E.T. the Extra-Terrestrial (1982)
2. Star Wars Episode IV - A New Hope (1977)
3. Star Wars Episode V - The Empire Strikes Back (1980)
4. Star Wars Episode VI - Return of the Jedi (1983)
5. Jurassic Park (1993)
6. Saving Private Ryan (1998)
7. Terminator 2: Judgment Day (1991)
8. The Matrix (1999)
9. Back to the Future (1985)
10. The Silence of the Lambs (1991)

11. Star Wars Episode I - The Phantom Menace (1999)
12. Raiders of the Lost Ark (1981)
13. Fargo (1996)
14. The Sixth Sense (1999)
15. Braveheart (1995)
16. Shakespeare in Love (1998)
17. The Princess Bride (1987)
18. Schindler's List (1993)
19. The Shawshank Redemption (1994)
20. Groundhog Day (1993)

The code below is use to print top 20 most popular movies.

```
1  % Get the dimensions
2  [m, n] = size(users_movies);
3
4  % Print a header indicating the movies to be listed
5  fprintf('The rating is determined based on the %d most popular movies: \n
6  ', length(index_small))
7
8  % Loop through of the popular movies
9  for j = 1:length(index_small)
10     % print the movies title correponding to the current index
11     fprintf('%d. %s \n', j, movies(index_small(j)));
12 end
13 fprintf('\n');
```

The "fprintf()" function is a versatile tool used to display formatted text directly in the Command Window. When constructing the output string, special format specifiers are used to indicate how different types of data should be displayed. For instance, "%d" serves as a placeholder that allows for the printing of an integer value. To control the layout of the output, the newline character, represented as "\n", is used to move the cursor to the beginning of the next line, effectively adding a line break.

Generally, "length(index_small)" is used to determine the total number of elements within an array named "index_small". This numerical result can then be incorporated into a formatted string using "fprintf()" and the "%d" specifier to display this count.

Loops are fundamental for repetitive tasks, and the "for" loop is commonly used to execute a block of code multiple times. Within such a loop, "fprintf()" can be employed again to print details for each iteration.

For example, the format string "%d. %s \n" instructs the program to print an integer (using %d), followed by a period and a space, then a string of characters (using %s), and finally, to add a new blank line (\n). In short:

- %d: Prints the index interger.
- %s: Prints the title movies.
- \n: Adds new blank line.

This is useful for creating numbered lists, such as displaying an index number alongside a movie title. To further improve the visual separation and readability of the output, an additional `"fprintf('\n')"` can be used to print an extra blank line, creating better spacing in the Command Window.

3. Select people

The following code selects people who rated all of the 20 movies under consideration:

```

1  % Select the users to compare to
2  [m1, n1] = size(users_movies_sort);
3  rating = [];
4  for j = 1:m1
5      if prod(users_movies_sort(j, :)) ~= 0
6          rating = [rating; users_movies_sort(j, :)];
7      end;
8  end;

```

Explain:

Initially, the code determines the dimensions of a matrix named `"users_movies_sort"` by using the `"size()"` function. This function call assigns the number of rows (representing users) to the variable `m1` and the number of columns (representing movies) to `n1`. Following this, an empty matrix called `ratings` is created, which will be used to store selected user data:

- `[m1, n1] = size(users_movies_sort)`: Measure the size of the `users_movies_sort` matrix. `m1` is the number of users (rows), `n1` is the number of movies (columns).
- `ratings = []`: Create an empty matrix called `ratings`.

The core of this code segment is a for loop that iterates through each user, from the first user (index 1) up to `m1` (the total number of users). Inside this loop, a conditional check is performed using an if statement. This condition evaluates the product of all ratings for the current user `j` (accessed via `users_movies_sort(j, :)`). If this product is not equal to zero (`~= 0`), it implies that the user has provided a rating for every movie (assuming no rating is represented by a zero). When this condition is true, that user's entire row of ratings from `users_movies_sort` is appended to the `ratings` matrix.

```

    for j = 1:m1
        if prod(users_movies_sort(j, :)) ~= 0
            ratings = [ratings; users_movies_sort(j, :)];
        end;
    end;

```

The whole point of this is to sort out users that rate all the film they watched and add their ratings to a set.

Question 1: What does the command `ratings=[]` do?

This initializes an empty array named `ratings` that ratings start with no elements. This setup is commonly used to collect or build up data in subsequent operations, specially within

loops. The code then iterates over `users_movies_sort` array and adds rows to ratings for users who have rated all the movies they watched.

The below table shows a part of the result of ratings after the execution of the source code:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	5	5	5	5	5	4	5	4	4	5	5	4	4
2	4	5	5	5	4	4	5	4	4	5	4	5	5	4
3	5	5	5	4	4	4	5	4	4	4	3	5	4	4
4	5	5	4	4	5	5	5	5	5	5	4	5	5	5
5	5	5	5	5	4	5	4	4	5	5	4	5	4	4
6	4	5	5	4	4	4	4	4	5	5	4	5	5	4
7	4	5	5	5	4	4	4	5	4	5	4	4	5	4
8	3	4	3	4	4	4	3	5	4	5	4	4	5	5
9	5	5	4	4	4	4	5	5	5	3	4	5	3	5
10	3	5	4	4	3	5	5	5	3	4	3	4	5	5
11	5	5	4	4	4	5	3	5	4	5	4	5	4	5
12	3	3	1	2	1	5	1	5	2	5	1	4	5	5
13	3	2	3	4	4	3	5	5	5	3	1	5	5	4
14	4	5	5	4	4	3	4	5	3	5	3	5	4	4
15	2	4	4	4	3	4	3	5	3	3	4	4	3	4
16	4	5	5	5	5	5	4	5	5	5	3	5	5	5
17	3	5	5	4	3	3	3	5	4	4	4	5	5	4
18	5	5	4	4	4	5	5	5	4	5	3	5	5	5
19	3	3	4	3	3	4	4	4	2	1	2	4	5	3
20	5	5	5	5	3	5	4	5	5	5	4	5	5	5

4. Find the Euclidean distance

The following code calculates the Euclidean distance:

```

1  % Find the Euclidean distance
2  [m2, n2] = size(ratings);
3  for i = 1:m2
4      eucl(i) = norm(ratings(i,:)-trial_users);
5  end;

```

In this part, we would calculate the Euclidean distance to measure the similarity between a specific “`trial_user`” and every other user within the ratings dataset.

Where:

- `ratings(i, :)`: is the vector of movie ratings for i-th user.
- `trial_user`: is the vector of movie ratings for the trial user (the user whose similarities we want to find).

The first step in the comparison involves element-wise subtraction, `ratings(i,:) - trial_user`.

- `ratings(i,:)-trial_users`: computes the difference between the i-th user’s ratings and the trial user’s ratings for each movie.

Subsequently, the “`norm()`” function is applied to this difference vector specifically,

- `norm(ratings(i,:)-trial_users)`: calculates the Euclidean distance between these two vectors.

The resulting distance for each comparison is stored in “`eucl(i)`”,

- `eucl(i)`: represents the Euclidean distance between the `trial_user` and the `i`-th user in the ratings array (or matrix). The smaller the value of `eucl(i)`, the more similar the `i`-th user's ratings are to the trial user's ratings, this means that the `i`-th user has comparable tastes in movies to the `trial_user`.

Moreover, the `eucl` vector contains a series of Euclidean distances, each representing the similarity between a trial user and other users in a dataset. A portion of this vector is displayed in the table, showing various distance values calculated:

	1	2	3	4	5	6	7	8	9	10	11
1	7.9373	7.6158	7.9373	9.1652	8.0623	7.3485	8.0000	7.6158	8.2462	7.6811	7.7460

The command “[smallest_number, position] = min(eucl);” processes the `eucl` vector to find its minimum value and the index at which this minimum occurs. The minimum value is stored in the variable “smallest_number”, while its corresponding index (or position within the vector) is stored in the variable `position`. Following this computation, the code uses `disp()` commands to output these findings:

```

1 % Find the smallest Euclidean distance
2 [smallest\_number, position] = min(eucl);
3 disp(['Smallest Eucl: ', num2str(smallest\_number)]);
4 disp(['Position: ', num2str(position)]);

```

Upon running the code, the results indicate that the smallest Euclidean distance found is 5.6569, and this value corresponds to position 14 in the `eucl` vector. This means that the 14th person in the ratings data has rating patterns most closely matching the trial user, with a calculated Euclidean distance of 5.6569, which is shown in the figure below:

```

Smallest Eucl: 5.6569
Position: 14

```

5. Find the Pearson correlation coefficient

Initially, we create a zero vector to save the coefficient. This vector will eventually store the calculated Pearson correlation coefficient for each user in the ratings matrix when compared against the `trial_user`. Concurrently, the `trial_user`'s rating vector is reshaped into a column vector using `trial_user(:)`; to ensure it is correctly oriented for the subsequent matrix operations, as shown in the figure below.

```

1 % Compute the Pearson correlation coefficient
2 pearson = zeros(size(ratings, 1), 1);
3 trial_user = trial_user(:);
4
5 % Compute correlation along the ratings vectors
6 for i = 1:size(ratings, 1)

```

```
7     current_rating = ratings(i, :);
8
9     % Compute mean
10    trial_user_mean = mean(trial_user);
11    current_ratings_mean = mean(current_rating);
12
13    % Subtract with the mean
14    trial_user_centered = trial_user - trial_user_mean;
15    current_ratings_centered = current_rating - current_ratings_mean;
16
17    pearson_numerator = sum(trial_user_centered .*
18    current_ratings_centered);
19
20    pearson_denominator = sqrt(sum(trial_user_centered .^ 2) .* sum(
21    current_ratings_centered .^ 2));
22
23    if pearson_denominator == 0
24        comr = pearson_numerator / pearson_denominator;
25    else
26        comr = 0;
27    end
28
29    pearson(i) = comr;
30    end
31
32    % Find the highest Pearson correlation coefficient
33    [max_pearson, pearson_index] = max(pearson);
34    disp(['Highest Pearson correlation coefficient: ', num2str(max_pearson
35    )]);
36    disp(['Position: ', num2str(pearson_index)]);
```

References