Similarity and recommender systems

Andreas C. Kapourani

1 Introduction

In this lab session we will work with some toy data and implement a simple collaborative filtering recommender system (RS), similar to what you were taught in the lectures. Given a set of *users*, *items* and user *ratings* for those items, the goal of a RS is to solve the following 2 tasks:

1. Predict a possible rating of a user on an item that he has not rated yet.
2. Create a list of the top N recommended items for each user.

In order to accomplish this, we need to measure the *similarity* between items or users. If we are building a recommender system for movies similar user feature vectors (e.g. derived from their rating histories on IMDb) might indicate users with similar tastes. The distance between two items depends on both the representation of feature vectors and on the distance measure used. In this lab, Euclidean distance will be used.

2 Euclidean Distance

The Euclidean distance $r_2(\mathbf{u}, \mathbf{v})$ between two 2-dimensional vectors $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2)$ is given by the following expression:

$$r_2(\mathbf{u}, \mathbf{v}) = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2} = \sqrt{\sum_{i=1}^{2} (u_i - v_i)^2} \tag{1}$$

Since we will use the above equation repeatedly, let's create a general function `square_dist` that computes a $1 \times M$ row vector of square distances between $M \times N$ and $1 \times N$ data U and v, respectively. We will use the `bsxfun` function to efficiently compute the vector of square distances.

```
function sq_dist = square_dist(U, v)
% Compute 1 x M row vector of square distances for M x N and 1 x N
   data U and v, respectively.
  sq_dist = sum(bsxfun(@minus, U, v).^2, 2)';
end
```

Then, in order to obtain the Euclidean distance of the above data U and v, we take the square root ($\sqrt{\cdot}$) of the output of `square_dist` function.

Let's see some examples:

```
% Create two random row vectors
>> v = [2 3 5 7 8];
>> u = [2 5 7 8 8];
% Compute the square distance
>> sq_dist = square_dist(v, u)
sq_dist =
     9
% Compute the Euclidean distance
>> euclid_dist = sqrt(sq_dist)
euclid_dist =
     3

% Compute the Euclidean distance between a row vector 1 X N and an
   M x N matrix
>> A = [1 4 3 3 2;
        3 2 1 2 2;
        1 3 2 2 2;
        1 4 5 6 7];
% Compute the square distance between each ROW of matrix A and v
>> sq_dist = square_dist(A, v)
sq_dist =
    58    79    71     4
% Compute the Euclidean distance
>> sqrt(sq_dist)
ans =
    7.6158    8.8882    8.4261    2.0000
```

Note that the `square_dist` function computes the distance between the row vector `v` and each row of matrix `A`. Thus, the number of columns of matrix A and the vector length should be the same.

Exercises

- Implement the `square_dist` function without using the `bsxfun` function. For example, you could use the `repmat` function or even using `for` loops (which in general is not advised since MATLAB is slow in performing `for` loops). Check that your results are the same for different combinations of inputs, e.g. giving as first input a vector and second input a matrix.
- Compute the Manhattan or city-block distance (explained in lecture notes) between `A` and `v`.

3 Movie recommendation system

On the course web site:

`http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnLabSchedule.html`

you will find a file named `data.txt`. Download the file on your current folder so that MATLAB can 'see' and open the file.

The file contains movie ratings from different critics. Each row corresponds to a particular critic and the columns correspond to a movie. The value of row c and column m is the rating given by critic c to movie m.

To obtain the feature vector (i.e. ratings) per critic we can take the whole row specific to that critic; similarly, if we care about the ratings of a specific movie, we take the whole column corresponding to that movie. Let's load the data to a matrix `A` using the built-in `importdata` function.

```
% name of the file to be loaded
>> filename = 'data.txt';
% split data at specified delimiter
>> delimiterIn = '\t';
% read numeric data starting from line headerlinesIn+1
>> headerlinesIn = 0;

% Use importdata to load the data to matrix A
>> A = importdata(filename, delimiterIn, headerlinesIn);

% Critic 3 ratings for all movies
>> A(3,:)
ans =
     7     5     5     0     8     4

% Check the dimensions of A (i.e. rows are critics, columns are movies)
>> size(A)
ans =
     6     6
```

So the file contains a 6×6 matrix of ratings, that is, 6 critics have rated 6 different movies. We can imagine each critic defining a point in a 6-dimensional space. Finally, by looking at the contents of matrix `A`, we can observe that the ratings are ranging from $0 - 10$.

Figure 1 shows a 2-dimensional review space in which the 6 critics are placed in a space defined by their reviews of two movies. We make the assumption that the closer two people are in this review space, the more similar are their tastes.

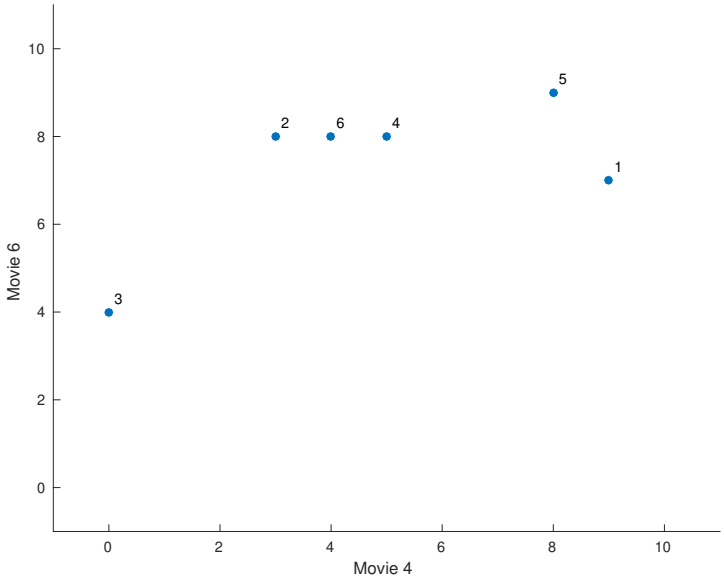

Figure 1: *Plot of critics in a 2-dimensional review space for Movie 4 and Movie 6. For the pair of movies under consideration the Euclidean distance between two points provides a measure of how different two reviewers are.*

The code for creating Figure 1 is the following:

```
% x-axis
>> x = A(:,4);
% y-axis
>> y = A(:,6);

% create a scatter-plot
>> scatter(x,y,'filled')

% set the axis limits
>> axis([-1, 11, -1, 11]);

% set the x...
>> xlabel('Movie 4');
% ... and y labels
>> ylabel('Movie 6');

% Create labels for each data point
>> a = [1:6]';
>> b = num2str(a);
>> c = cellstr(b);

% displacement so the text does not overlay the data points
>> dx = 0.1; dy = 0.3;

% show text in the scatter-plot
>> text(x+dx, y+dy, c);
```

Exercise

Create a plot similar to Figure 1, but now in a 2-dimensional review space for Critic 1 and Critic 6. Under Euclidean distance this will give you the measure of how different two movies are, based on those critics.

3.1 Compute Euclidean distance between critics

We can go ahead and compute the actual Euclidean distances between all the critics using Equation 1 and function `square_dist` in the 6-dimensional review space (i.e. the number of movies).

```
% Get matrix size
[row, col] = size(A);

% Create a row X row matrix of zeros
critic_dist = zeros(row, row);

% for each critic compute his distance with all the other critics
for i = 1:row
    critic_dist(i, :) = sqrt(square_dist(A, A(i,:)));
end
```

Check the `critic_dist` matrix and make sure you understand what the value in each cell corresponds. For example, the diagonal elements of the matrix are 0, and the matrix is symmetric (i.e. $A = A^T$).

Exercise

Compute the Euclidean distance between the movies in the 6-dimensional review space (i.e. the number of critics). **Note**: In general you can compute the Euclidean distance between movies or reviews without using `for` loop, however the code is more complicated (although it can be written in one line). In Iain Murray's web page:

`http://homepages.inf.ed.ac.uk/imurray2/code/imurray-matlab/square_dist.m`

you can find an implementation of the `square_dist` function, which can also compute all the pairwise distances when two matrices are given as input. In our previous example, if we used the updated `square_dist` function, we could compute the Euclidean distance between critics as follows:

```
% Compute Euclidean distance between critics using the square_dist
   function given in the above link
critic_dist = sqrt(square_dist(A', A'))
```

3.2 User Ratings

Consider that we have two new users, `User1` and `User2`, who reviewed a subset of movies. Based on those ratings we want to find to which critic is each user most similar. Then, based on this information we can build a simple recommender system that will propose which movie each user should see based on their individual preferences.

Let's assume, `User1` has rated `Movie4` with 2 and `Movie6` with 7, and `User2` has rated `Movie2` with 6, `Movie3` with 9 and `Movie6` with 6. We will compute the Euclidean distance of each user to all the critics and find the closest ones (i.e. the most similar ones). Note, that we can compute this distance only on the movies that the user has seen!

```
% User1 actual ratings
u1_scores = [2 7];
% Indices of movies that User1 reviewed
u1_movies = [4 6];

% User2 actual ratings
u2_scores = [6 9 6];
% Indices of movies that User2 reviewed
u2_movies = [2 3 6];

% Compute Euclidean distance between each user and critics
>> u1_critic_dist = sqrt(square_dist(A(:,u1_movies), u1_scores))
u1_critic_dist =
    7.0000    1.4142    3.6056    3.1623    6.3246    2.2361

>> u2_critic_dist = sqrt(square_dist(A(:,u2_movies), u2_scores))
u2_critic_dist =
    5.1962    4.5826    4.5826    2.2361    3.7417    2.4495
```

We observe that `User1` is most similar with Critic 2 and `User2` is most similar with Critic 4. Thus, a simple approach to fill the 'missing' user reviews for the rest of the movies, is to use x_{c*m} for $\hat{x}_{um}$, where x_{c*m} is the review of critic c for movie m, and $\hat{x}_{um}$ is the predicted review for user u for movie m.

To visualize the distances between users and critics, Figure 2 shows the plot of critics together with the `User2` reviews in the 3-dimensional review space.

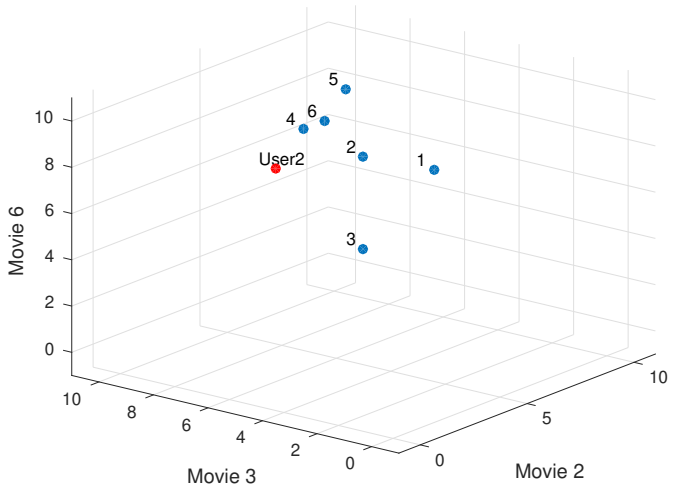

Figure 2: *Plot of critics in a 3-dimensional review space for Movie 2, Movie 3 and Movie 6. The red dot denotes the 3-dimensional review space for the same movies for User2.*

The MATLAB code for creating the above plot is the following:

```
% Critic reviews vector space
x = A(:,2);  % x-axis
y = A(:,3);  % y-axis
z = A(:,6);  % z-axis

% create a scatter3 plot with 'filled' points
scatter3(x,y,z,'filled')
% set the x, y and z axis limits
axis([-1, 11, -1, 11, -1, 11]);
% set the x, y and z labels
xlabel('Movie 2'); ylabel('Movie 3'); zlabel('Movie 6');

% Create labels for each data point
a = [1:6]'; b = num2str(a); c = cellstr(b);
% displacement so the text does not overlay the data points
dx = 0.1; dy = 0.7; dz = 0.2;
% show text in the scatter-plot
text(x+dx, y+dy, z+dz, c);

% keep the current plot
hold on

% add the User2 reviews in the plot
X = u2_scores(1);
Y = u2_scores(2);
Z = u2_scores(3);
% create a scatter3-plot for User2 reviews
scatter3(X,Y, Z, [], 'red','filled')
% show text in the scatter-plot
text(X+dx, Y+dy, Z+dz, 'User2');
```

Exercise

Create a plot of critics in a 2-dimensional review space together with the `User1` reviews for Movie 4 and Movie 6. **Hint**: Most of the code is already written for creating a 2-dimensional review space for critics in Figure 1.

4 Similarity Measure

Until now, we have used the terms Euclidean distance and similarity interchangeably, although they are not exactly the same. We can convert the Euclidean distance measure into a similarity measure using the following equation:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{1}{1 + r_2(\mathbf{u}, \mathbf{v})} \tag{2}$$

This similarity measure has some desirable properties: a *distance* of 0 corresponds to a *similarity* of 1 (the largest value it can take); a *distance* of ∞ corresponds to a *similarity* of 0 (the smallest it can take).

Since we will compute the similarity between any two objects repeatedly, let's implement this similarity measure by creating a MATLAB function named `sim_measure`:

```
function similarity = sim_measure(r2)
% Convert Euclidean distance to similarity measure
  similarity = 1 ./ (1 + r2);
end
```

Now we can compute the critics' similarities to `User2`, by converting the Euclidean distances between critics and `User2` to similarity measures as follows:

```
% Compute similarity measure
>> u2_sim = sim_measure(u2_critic_dist)
u2_sim =
    0.1614    0.1791    0.1791    0.3090    0.2109    0.2899
```

4.1 Making recommendations

One way to make a recommendation for `User2` would be to choose the most similar critic, and then propose the movie that they ranked most highly; however this approach has a couple of drawbacks (see lecture notes). An alternative approach is to take an *weighted average over critics*, that is, to weight critic scores according to the similarity between the critic and user. An overall score for each movie can be obtained by summing these weighted rankings. If u is the user, and we have C critics, then the estimated score given to movie m by u, $\hat{x}_{um}$, is obtained as follows:

$$\hat{x}_{um} = \frac{1}{\sum_{c=1}^{C} sim(\mathbf{x}_u, \mathbf{x}_c)} \sum_{c=1}^{C} \left(sim(\mathbf{x}_u, \mathbf{x}_c) \cdot x_{cm} \right) \tag{3}$$

where $\mathbf{x}_u$ is a vector of ratings for the films seen by u, and $\mathbf{x}_c$ is the vector of corresponding movie ratings from critic c. The normalisation outside the sum, means that if every critic has the same score, the (weighted) average will report that score.

We can now estimate `User2` reviews for each unseen movie using the above equation:

```
% Predict user2 reviews
>> u2_pred_rev = (1/sum(u2_sim)) * (u2_sim * A)
u2_pred_rev =
    5.7323    6.3872    6.7060    4.8003    8.4530    7.4983
```

```
% Get only the ones that user2 has not reviewed
>> u2_pred_only = u2_pred_rev(setdiff(1:6,u2_movies))
u2_pred_only =
    5.7323    4.8003    8.4530
```

Note that the above code predicts reviews for all possible movies, even though we might have an actual rating from the user in some of them. We can simply take only the ones that the user has not reviewed yet and ignore the others.

Hence, the recommender system would propose Movie 5 to `User2`. But more than just proposing a single film, it provides an estimate of the rating that `User2` would provide to each film based on `User2`'s ratings of other films, the estimated similarity of `User2` to each critic, and the ratings of the critics to films unseen by `User2`.

Exercise

Compute the critics' similarities to `User1`, by converting the Euclidean distance to similarity measure; then predict rating reviews for all unseen movies of `User1` and finally, make recommendations for `User1`.

5 Normalization

So far, our estimate of similarity has been based on the Euclidean distance between feature vectors in the original review space. However, this distance is not well *normalised*. For example, two critics may rank a set of films in the same order, but if one critic gives consistently higher scores to all movies than the other, then the Euclidean distance will be large and the estimated similarity will be small.

One way to normalise the scores given by each critic is to transform each score into a *standard score* (also called *z-score*). The standard scores are defined such that the set of scores given by each critic have the same sample mean and sample standard deviation. (**Important**: Many systems actually work better when you pre-process the data and use standarized features).

The standard score or z-score for critic's c rating for movie m is given by:

$$z_{cm} = \frac{x_{cm} - \bar{x}_c}{s_c} \tag{4}$$

where $\bar{x}_c$ is the sample mean for each critic, and s_c is the sample standard deviation for each critic.

We can compute the *standarized* critic reviews in MATLAB using the following code:

```
% Compute the sample mean for each critic
>> x_cm = mean(A, 2)'
x_cm =
    6.5000    6.0000    4.8333    6.8333    8.0000    6.8333

% Compute the sample standard deviation
% The 0 in the second argument signifies to take the sample s.d. (i.e.
   divide by N-1)
>> s_c = std(A, 0, 2)'
s_c =
    2.5100    2.0000    2.7869    1.7224    1.6733    1.4720

% Mean normalize the original review matrix
>> mean_shift = bsxfun(@minus, A', x_cm)';
```

```
% Compute the z-score
>> z_cm = bsxfun(@rdivide, mean_shift', s_c)'
z_cm =
   -1.3944    0.1992   -0.9960    0.9960    0.9960    0.1992
    0.5000   -0.5000   -0.5000   -1.5000    1.0000    1.0000
    0.7775    0.0598    0.0598   -1.7343    1.1363   -0.2990
   -1.0644   -0.4838    0.6773   -1.0644    1.2579    0.6773
   -1.7928         0         0         0    1.1952    0.5976
    0.1132    0.1132    0.7926   -1.9249    0.1132    0.7926
```

MATLAB provides a built in function `zscore` that implements for us all the previous steps, but in general writing your code helps you understand better the task you are performing.

```
% zscore function example
>> zscore(A, 0, 2);
```

We can now observe how the critic reviews vary between critics using the standardized data. Figures 3-6 show, the original review scores, mean normalized review scores and standardized review scores, respectively, for each movie. We observe that the initial high variance between critic scores, shown in Figure 3, becomes much smaller when we apply mean normalization and standardization to the data.

The code for creating Figure 3 is the following:

```
% Code for creating Figure 3 below
x = [1:6];
plot(x, A', '--*')
data_names = [{'Cr 1'},{'Cr 2'},{'Cr 3'}, {'Cr 4'}, {'Cr 5'}, {'Cr 6'}];
legend(data_names, 'Location', 'southeast')
title('Critics original review scores', 'FontSize', 14);
xlabel('Movies'); ylabel('Scores');
```

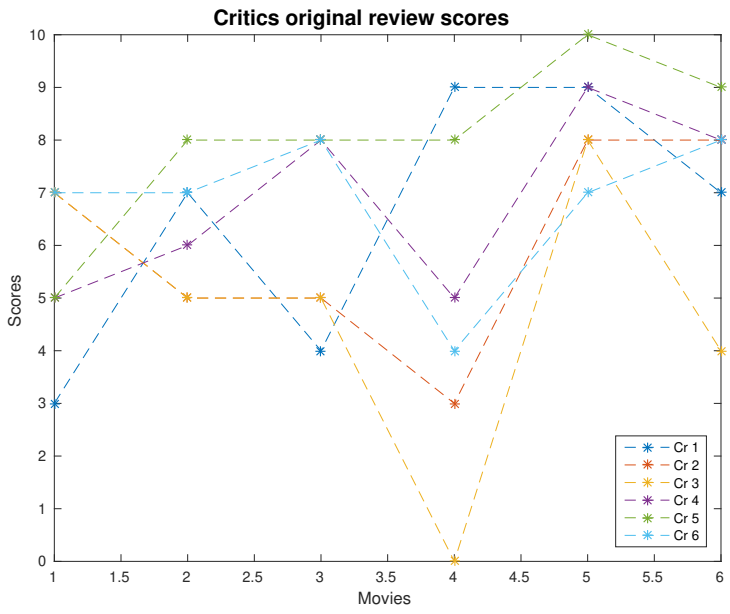

Figure 3: *Original critics review scores (y-axis) for each movie (x-axis). Each coloured line represents a different critic, and each star denotes the actual rating of that critic for the specific movie.*

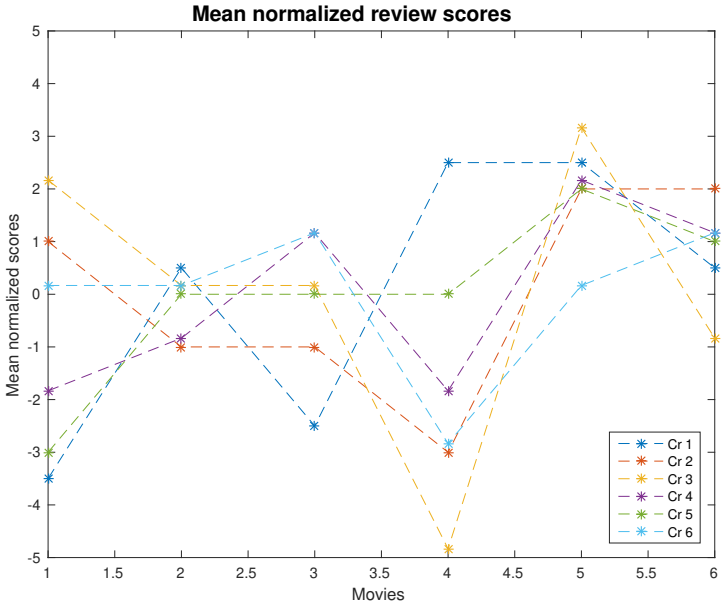

Figure 4: *Mean normalized critics review scores (y-axis) for each movie (x-axis). Each coloured line represents a different critic, and each star denotes the actual rating of that critic for the specific movie.*

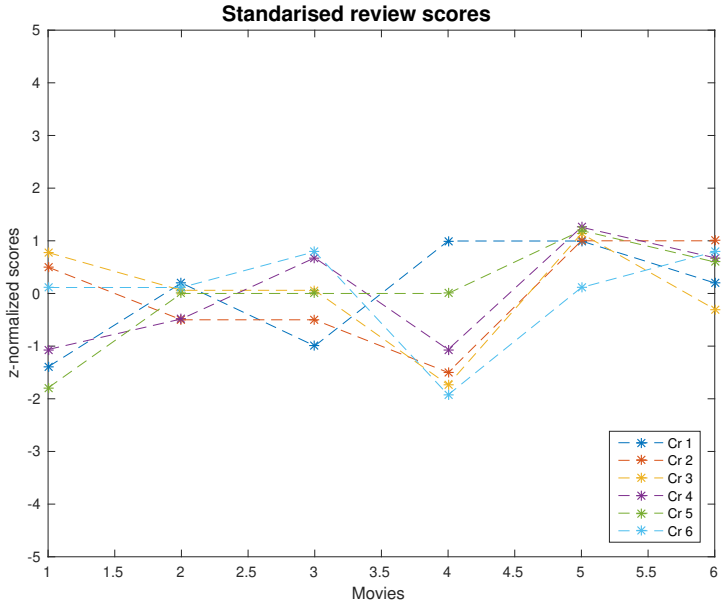

Figure 5: *Standardized critics review scores (y-axis) for each movie (x-axis). Each coloured line represents a different critic, and each star denotes the actual rating of that critic for the specific movie.*

Exercises

- Create the plots for Figures 4 and 5. Note that the y-limits are set to (-5, 5), this way we can observe the difference between the mean normalized and the z-normalized review scores.
- Compute similarity matrix between critics based on Euclidean distance.

- Compute Pearson Correlation Coefficient (PCC) similarity matrix between critics. The formula for computing the PCC between critic `c` and `d` is:

$$r_{cd} = \frac{1}{M-1} \sum_{m=1}^{M} z_{cm} z_{dm} \tag{5}$$

where z_{cm} is the z-score for critic's `c` rating for movie `m`.

6 Summary

This lab session introduced the notion of distance between feature vectors and how to implement a simple recommender system. In general for recommender systems:

- We represent the rating data from U users about M items as a $U \times M$ matrix.
- Each row of this data matrix corresponds to a user's feature vector in *review space*.
- Each column of this data matrix corresponds to an item's feature vector, and distance measures between these vectors correspond to dissimilarity between items.
- We can *predict* missing review scores based on the weighted average of similar rows.
- The similarity is based on Euclidean distance and Pearson Correlation Coefficient (see lecture notes).