Programming Assignment

In this project we will be creating a simple recommender system for movies, similar to the ones used for systems like Netflix, Hulu, Apple TV, Youtube, etc. You will implement one recommender system that will implement different recommender algorithms and you will compare the results among these algorithms.

You can use any language of your choice for this project but you need to write the code that does the respective functionality. In other words, you cannot use libraries that directly implement one of the recommendation algorithms below. You need to implement the logic that is described below and you cannot use a database for storing and joining the data.

Preliminaries

A recommender system is an information system that leverages data (information on items and user behavior) that provides suggestions for items that are most interesting to a particular user. In general, recommender systems today are pretty complex and take into account a multitude of signals for each item (e.g., movie) and user. In this project we will implement a simple version of a recommender system.

1. Data Loading (5%)

We will use a public dataset from movielens. Most specifically, you need to download the latest dataset from movie lens. You will need to download the ml-latest.zip file.

Once you download and unzip, please ensure that you can extract it and go through the README file that has useful information (together with the description web page).

After downloading the data, you need to write code to load the data. For this section, please report the number of items within each .csv file in the zip archive.

2. Recommender System (95%)

You will now create a recommender system that will be able to run several algorithms as follows:

```
recommender -d directory_of_data -n number_of_recommendations \
            -s similarity_metric -a algorithm -i input
```

Where the first parameter is the directory where you extracted the data. The second parameter is the number of recommendations that you should return. Next is a similarity metric and the algorithm to run described below. The final parameter is the input for which we request the recommendations (for example, a user id).

Given two vectors of data, your application will need to support the following similarity metrics (ensure they are given as lowercase in the command line as in the headers below):

- jaccard

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- dice

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

- cosine

$$S_C(A,B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}},$$

- pearson

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

2.a. User-user recommendation `algorithm=user` **(10%)**

In this case, you will implement the user-user similarity approach. To this end, you need to get all the user ratings from the dataset and for a given user (i.e., `input=user_id`) you will return the top `n` recommendations:

- Let r_x be the vector of user x's ratings
- Let N be the set of k users most similar to x who have rated item i (set k = 128)

Then the recommendation score for item i for user x is:

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Where s_{xy} is the similarity function given as parameter.

2.b. item-item recommendation `algorithm=item` **(10%)**

In this case, you will need to implement the item-item similarity approach. You will again need to get all the user ratings from the dataset and for a given user (i.e., `input=user_id`) you will return the top `n` recommendations:

- Let r_x be the vector of user x's ratings
- Let N be the set of k items most similar to i (set k = 128)

Then the recommendation score for item i for user x is:

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

Where s_{ij} is the similarity function given as parameter.

2.c. Tag-based recommendation `algorithm=tag` **(10%)**

In this case, you will need to create for each item a vector where position i is the number of users that have given this item this tag. For example assume that we have in the system 3 tags in total: tag1, tag2, and tag3 then for item x one example vector is this:

x = [3, 2, 4]

This vector means that 3 users have tagged item x with tag1, 2 users with tag 2 and 3 with tag 3. If we had more tags than 3 then x would have zeros in those respective places.

Next, in order to produce recommendations for a given movie (i.e., `input=movie_id` in this case), you will need to compute the n most similar movies using the similarity function that was given as a parameter.

2.d. Content-based recommendation `algorithm=title` **(30%)**

In this case, you will need to create for each item a vector where position i corresponds to a keyword. The set of all keywords in the vector is the titles of all movies.

The vectors are generated using the TF.IDF approach over all the movie titles.

For example assume that, we have 5 keywords overall in the system (i.e. the set of all keywords in all movie titles is 5): w1, …, w5

Also assume that we have computed the IDF vector as follows:

IDF = [0.2, 0.04, 0.1, 0.1, 0.3]

And that TF_x = [2, 3, 0, 0, 0]

This means that movie x has the keyword w1 2 times in its title, the keyword w2 3 times and the remaining keywords 0 times.

Then, the TF.IDF vector of x would be:

$TF.IDF_x$ = [0.4, 0.12, 0, 0, 0]

This vector can now be used as a representation of the movie and fed into the respective similarity functions specified in the parameter.

2.e. hybrid recommendation `algorithm=hybrid` **(20%)**

Create your own recommendation method by combining three or more of the algorithms above. You can use the output scores of each method and perform a linear combination (or any other combination of your choice). Please describe your approach clearly in your report.

2.f. Real system (15%)

The previous examples were implemented in a naive way in the sense that, every time you would run your application you would have to load the data and perform computations from scratch. This is both inefficient in terms of resources and also slow.

For this part of the project, you should create another application called `preprocess` that runs once and performs all the computations needed for the previous questions. Once the computations are performed they are written to disk in text files with a format of your choice (please ensure that you describe the format in detail in your report).

Next you should modify the `recommender` app to use these precomputed files in order to provide recommendations much faster.

Hint: while developing both this and the previous questions you may want to use a smaller dataset from the movie lens website e.g. the 100k dataset. However, please ensure that your code will run on the full dataset. You may also find this question challenging to run on the full dataset. If so, please explain the challenges and how you would solve them in a real recommender system.

2.g. Web interface (bonus +10%)

Implement a simple Web UI (i.e. accessible locally through a Web browser) where you can provide an input (e.g. a movie title) and you get back the top-100 similar movies. The similarity function and the algorithm to run can be selected from a drop-down list.

Deliverables

- All files for your project need to be zipped in **one single zip file.** The file name should be in the following format: Lastname_Firstname_M111_Project.zip
- Please upload the zip file to the eClass project area by the deadline. **There are absolutely no exceptions and no extensions.**
- Inside the zip file, please include a) your source code b) any results that are part of the project c) a README text file that explains how to compile (if needed) and run your code and d) a short and concise report (preferably pdf - no more than 10 pages) describing your work and your approach to solving the project. **Please do not include binaries, nor the original or any derived data in the zip file.**

Important Notes

- Please check the eClass web page: https://eclass.uoa.gr/courses/DI508/ regularly for announcements and/or clarifications to the project. Announcements will show up there and will also be sent to the mailing list.
- You are free to make any assumptions you may need to along the way as long as you document them in your report.

- The project is meant to be worked on by the student submitting and only that student. In the event that there is a submission not worked on by the student, then the student fails the class.
- You will be asked to present your project to the instructor so please ensure you have legible and well-organized code.
- Copying code from existing sources (e.g., the Web, github, etc.) is strictly prohibited.
- You are responsible to safeguard your code, so if you are using a code repository service (e.g., github) keep your repository private. Cases when code was cloned and was also submitted by someone else cannot be reliably traced and result in everyone involved (regardless of who is at fault or who copied from whom) failing the class.
- Although students are expected (and encouraged) to chat among them on potential solutions and approaches, sharing code and solutions is strictly not allowed. In the event that two or more students provide submissions that have common pieces, everyone involved (regardless of who is at fault or who copied from whom) fails the class.