Recommender Systems
Content Based methods

Notice From now until the end of this course choose any programming language that you prefer, JavaScript (nodejs), Python, C#, C++ or any other programming language is accepted. Our suggestion is Python and this exercise explanations are based on Python respectively.

MovieLens dataset : This dataset (`http://movielens.org`) describes 5-star rating and free-text tagging activity from a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018.

1 Data Preparation for content based recommender system.

Question 1: **Gather the Data**: To start with making a *content based* recommender system we first require a dataset including users, items and the user ratings on items. Download the MovieLens dataset given in the BrightSpace. For the sake of simplicity, we have prepared a small version of dataset namely `movieLens_small`. In the movieLens dataset, we need two files: `rating.csv` and `movies.csv`. There is no file containing user information in this database.

Question 2 First upload the `movies.csv` file. Second write a function namely `extractor(line)` receiving a line of `movies.csv` file and extracting movieId (in integer), movie title (in string), movie year (in integer) and movie genres (in string). For instance this function receives:

188189 Sorry to Bother You (2018) Comedy|Fantasy|Sci-Fi

and returns back:

(188189, Sorry to Bother You, 2018, Comedy|Fantasy|Sci-Fi)

Question 3, represent document based parameters Represent the movie genres as an onehot value for each movie. One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. For instance as a movie line we have a dataset containing:
[1, Toy story, 1995, Adventure|Animation|Children|Comedy|Fantasy]
genres can be presented as a vector containing 0 and 1 elements with respect to total existed genres in the dataset. If we have in total ten genres in the database such as

- Fantasy
- Children
- Action
- Horror
- Comedy
- Adventure
- Sci-Fi
- Crime
- Romance
- Animation

this movie genres should be represented as: $[1, 1, 0, 0, 1, 1, 0, 0, 0, 1]$. In the represented vector for the genre has five 1s while the rest of parameters are 0 w.r.t the total existed genres in the dataset.
This is an example of the movie dataset after a onehot representations for the genres:

	movieId	title	year	IMAX	Fantasy	Children	Action	Horror	Film-Noir	Sci-Fi	Western	Romance	Documentary	War	Mystery	Thriller	Musical	Drama	Comedy	Adventure	Animation	Crime
0	1	Toy Story	1995	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
1	2	Jumanji	1995	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	3	Grumpier Old Men	1995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	4	Waiting to Exhale	1995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
4	5	Father of the Bride Part II	1995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2738	4988	White Water Summer	1987	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2739	4989	How High	2001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
2740	4990	Jimmy Neutron: Boy Genius	2001	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
2741	4991	Joe Somebody	2001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0
2742	4992	Kate & Leopold	2001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

2743 rows × 22 columns

Hint: to simplify this exercise, first extract all available genres in the data base. The help code is given as bellow:

```
import pandas as pd
import numpy as np

#upload the database wherever you prefer. We used google
    colab in our case.
from google.colab import files
uploaded = files.upload()

#import movie.csv with description
movies = pd.read_csv('movies.csv', ',')

# check several first lines of the movies file
movies.head()

# movies preparation
all_genres = set()
for index, row in movies.iterrows():
    #print(extractor(row['title']))
    genres_list = row['genres'].split('|')
    for gen in genres_list:
        all_genres.add(gen)



all_genres = list(all_genres)
len(all_genres)
print(all_genres)
```

with the help of this script, generate the onehot codes for the movies genres and finally modify the movie dataset as Figure 1.

Note: There is no harm if some components of the vectors are boolean and others are real-valued or integer-valued. We can still compute the cosine distance between vectors, although if we do so, we should give some thought to the appropriate scaling of the nonboolean components, so that they neither dominate the calculation nor are they irrelevant.

Question 4 For binary representation, we can perform normalization by dividing the term occurrence(1/0) by the number of attributes in the movie. Hence, for Toy story: normalized attribute = 1/5.0 = 0,2. After normalizing each line of the dataframe, save the modified dataframe in a new variable. The results is similar to the following Figure:

	Adventure	Mystery	Film-Noir	Crime	Children	Drama	Documentary	Fantasy	War	Romance	Western	Musical	Thriller	Action	Horror	Comedy	IMAX	Sci-Fi	Animation
0	0.200000	0.0	0.0	0.0	0.200000	0.000000	0.0	0.200000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.200000	0.0	0.0	0.20
1	0.333333	0.0	0.0	0.0	0.333333	0.000000	0.0	0.333333	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.00
2	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.500000	0.0	0.0	0.0	0.0	0.0	0.500000	0.0	0.0	0.00
3	0.000000	0.0	0.0	0.0	0.000000	0.333333	0.0	0.000000	0.0	0.333333	0.0	0.0	0.0	0.0	0.0	0.333333	0.0	0.0	0.00
4	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	1.000000	0.0	0.0	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2738	1.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.00
2739	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	1.000000	0.0	0.0	0.00
2740	0.250000	0.0	0.0	0.0	0.250000	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.250000	0.0	0.0	0.25
2741	0.000000	0.0	0.0	0.0	0.000000	0.333333	0.0	0.000000	0.0	0.333333	0.0	0.0	0.0	0.0	0.0	0.333333	0.0	0.0	0.00
2742	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.0	0.500000	0.0	0.0	0.0	0.0	0.0	0.500000	0.0	0.0	0.00

2743 rows × 19 columns

Question 5 For preparing the utility matrix we normally use the user-product rating table. For our case, we need to upload `ratings.csv` file. Write a function namely `utility_matrix()`. This function should generate a 2 dimension matrix of the `rating.sv` file. The matrix has number of rows equal to total number of `user_id`s and number of columns equal to total number of `movie_id`s. Each element represents the given `ranking` by the `user_id` and `movie_id` i.e.:

$$\texttt{matrix}[\texttt{user\_id}, \texttt{movie\_id}] = \texttt{ranking}$$

For the simplicity, you can use the following script to progress with this exercise:

```
#import more than 10 user ratings
ratings = pd.read_csv('ratings.csv', sep =',')

def utility_matrix():

    "we add +1 here because the indexes in the csv files
   start by 1 while in python they start by 0"
```

```
    utility_matrix = np.zeros(shape=(int(max_user)+1, int(
    max_movie)+1))

    for index, row in ratings.iterrows():
        #compelet this part by yourself

    return utility_matrix

utility_matrix = utility_matrix()
```

Question 6 In order to be able to compare users with items (movies), the user should be presented with the same number of attributes. In order to first merge the properly presented movies and user files, (if you use the panda dataframe) you can write:

```

ratings['movieId']=ratings['movieId'].astype(int)
movies_new['movieId']= movies_new['movieId'].astype(int)

result = pd.merge(ratings, movies_new[movies_new_columns], on
    ='movieId')
result.head()
```

where `movies_new` is the movie data set after normalisation and `movies_new_columns` is a list of column titles to be taken into the account for merging two files. The result data frame, should be similar to:

	userId	movieId	rating	timestamp	title	year	IMAX	Fantasy	Children	Action	Horror	Film-Noir	Sci-Fi	Western	Romance	Documentary	War	Mystery	Thriller	Musical	Drama	Comedy	Adventure	Animation	Crime
0	1	1	4.0	964982703	Toy Story	1995	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0
1	5	1	4.0	847434962	Toy Story	1995	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0
2	7	1	4.5	1106635946	Toy Story	1995	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0
3	15	1	2.5	1510577970	Toy Story	1995	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0
4	17	1	4.5	1305696483	Toy Story	1995	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0

Question 7 In order to make a vector for each user, assume the list of ranked movies by user 2 are as following:

	userId	movieId	rating	year	IMAX	Fantasy	Children	Action	Horror	Film-Noir	Sci-Fi	Western	Romance	Documentary	War	Mystery	Thriller	Musical	Drama	Comedy	Adventure	Animation	Crime
2063	2	333	4.0	1995	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	1.0	0.000000	0.0	0.0
12125	2	3578	4.0	2000	0.0	0.0	0.0	0.333333	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.333333	0.0	0.333333	0.0	0.0
12660	2	1704	4.5	1997	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.500000	0.0	0.000000	0.0	0.0

in order to represent user 2 with the same number of attributes as movies, we can compute the inner product of rating column to each column for instance:

for the 'Action' we have: $4 \times 0.0 + 4 \times 0.33 + 4.5 \times 0.0 = 1.32$. consequently, the complete representation of user 2 for movie genres is as:
0.000000 0.000000 0.000000 1.333333 0.000000 0.000000 0.000000 0.000000 2.250000 0.0 0.000000 0.000000 0.000000 0.000000 3.583333 4.000000 1.333333 0.000000 0.000000
Notic: If you need save users or movie datasets as matrix in the python code. Since these matrices are sparse, we can save them in a parse matrix defined in python using `sparse_utility = scipy.sparse.coo_matrix(utility_matrix)` In order to two libraries should be imported in python project:

- `import scipy`
- `from scipy.sparse import csr_matrix`

It is better to save them in the local memory (for instance using `pickle` in order to run the time consuming codes only once.)

2 Recommending Items to Users Based on Content

With profile vectors for both users and items, we can estimate the degree to which a user would prefer an item (movie) by computing the cosine distance between the user's and movie's vectors.
Remind Assume the user profile is presented by c vector and movie profile by m vector. Then a prediction heuristic can be defined as: $u(c, m) = cos(c, m) = \frac{c \cdot m}{|c||m|}$ (the multiplication is an inner production among two vectors)
Question 8 Implement a heuristic prediction method (as above) and test it on your users items. For each given user, recommend her/his top 3 items regarding the prediction function. In order to implement the idea, we receive a user equivalent vector as c and compute its cosine similarity with total number of movies i.e. $\{cos(s, m) \forall m\}$ and from this list we select the three movies with the highest cosine similarity. Test the method for one or two users and find their most three recommended movies. $\triangle$

Classification Algorithms
A completely different approach to a recommendation system using item

profiles and utility matrices is to treat the problem as a machine learning problem. Regarding the given data as a training set, and for each user, build a classifier that predicts the rating of all items. There are a great number of different classifiers, and it is not our purpose to teach this subject here.
The general idea is that for each given user, we require to learn a classifier that classifies items into rating classes: liked by user and not liked by user. It means, we are interested in learning a classifier as:

$$C : U \times S \longrightarrow \{1, 2, 3, 4, 5\}$$

This means item $s \in S$ has been ranked by user $u \in U$ with a score between 1 and 5. Notice that the training set contains all elements of the utility matrix. And the not ranked movies by users should be predicted using a prediction function.

There are a great number of different classifiers such as Bayesian,logistic regression, support vector machine (SVM), decision trees or neural networks. Although it is not our purpose to teach this subject here, do the following question if you have some knowledge about some classical machine learning methods [1].
Question 9 Choose a classification algorithm and train it on existed $(u, s) = rate$ data set. Since the output of the classifier is 1 to 5, present the rating data between 1 and 5. After preparing the trained data, train your selected classifier method on the given data. After accomplishing the training part, test your classifier on each user. For instance to find recommendation candidates for each user, apply the classifier to each item and print the movies highly ranked by the user. $\triangle$

[1] During your first semester at this semester at ESILV, you have seen similar algorithms in machine learning and deep learning classes.