Recommender Systems: Term Project

Deadline: April 1, 2019

1 Introduction and motivation

In this project we will explore different performance evaluation metrics for recommendation systems by working on real datasets. Besides the traditional metric of accuracy of rating prediction, we are going to study:

- Diversity,
- Coverage,
- Serendipity,

in the sense to be described below. The output of a recommendation algorithm is a list of recommended items for each user. Each recommendation list has size L items recommended to each user, where L typically takes values $2, 5, 10$. In our case, we will first create the recommended lists of items for each user with a default algorithm, e.g. item-item Collaborative filtering (CF). Let $\mathcal{L}_u$ denote the set of items recommended to user u.

As we will see in class, in recommendation systems, the traditional performance objective to be optimized is the Mean Square Error that measures the squared difference between predicted ratings (obtained through the recommendation algorithm) and true ratings that users have entered. This metric measures the *accuracy* of prediction of the recommendation algorithm, but it does not take into account several other evaluation metrics, such as diversity, coverage or serendipity.

Assume that items to be recommended belong to different "owners". For example, items may be hotels of different brand chains, books of different editor companies, movies of different producers or other sponsored items for which their owners have paid to have them appear in users' recommendation lists. In this case, one of the concepts it would make sense to bring into the picture is *diversity*.

Diversity means that items are recommended to users with different profiles in general, so that they extend reach of items. For example, movies of a certain producer (which are not necessarily similar among themselves) can be recommended to people with different profiles, if that is beneficial for the system as a whole. Or, if we have two categories of books, one from Cambridge and one from Prentice-Hall, in order to treat each category in a fair manner, we would like to recommend books from each category to diverse users in order to extend their reach.

We wish to recommend items to a diverse set of users. Item-level diversity allows for more holistic dissemination of items to users. User-level diversity fails to consider item-level diversity, since assigning recommendations based on user satisfaction would still only show items to users who fall in their traditional "audience." This would result in bad item-level diversity but could still give a high user-level diversity if recommendations are diverse enough.

1.1 Problem A: Definitions

Consider a set of items $\mathcal{I}$ and a set of users $\mathcal{U}$, and assume some baseline recommendation system (e.g. item-item CF) that generates recommendation lists $\mathcal{L}_u$ for each user u. Let C be the number of categories, and each item belongs in exactly one category. Let $\rfloor$ the set of items of category c, $c = 1, \ldots, C$.

We would like to come up with new recommendation lists $\mathcal{L}'_u$ for each user u such that in the new lists, items of each category are recommended to a diverse a set of users. We do not need to recommend all the items in each category. This means that the recommendation list for each user u will change from the baseline one $\mathcal{L}_u$ (coming from the item-item CF) to

the new one $\mathcal{L}'_u$ that will fulfill the item diversity, and possibly other constraints. We will see how later.

Diversity: Let $\mathcal{I}_c$ be the set of items of category c. A metric of diversity for items of category c is as follows. Let w_{uv} the similarity between a pair of users (u, v). This can be computed through metrics such as cosine similarity or Pearson correlation coefficient, with the methodology that we have seen (or will see shortly) in the class in Unit 3, applied for CF. Furthermore, let $x_{iu} = 1$ if item i is recommended to user u, and $x_{iu} = 0$ otherwise. These variables refer to the *new recommendation lists* that we will produce. Then, the diversity of users to which items from category c are recommended is as follows:

$$\text{Div}_c = \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}: v \neq u} d_{uv} x_{iu} x_{iv} \,. \tag{1}$$

where d_{uv} is the dissimilarity between users u, v. Dissimilrity between users could be computed as $d_{uv} = 1 - w_{uv}$ with $0 \leq d_{uv} \leq 2$. According to the formula above, we compute the diversity for items of category c as follows. For each item in the set of items $\mathcal{I}_c$ of category c, we look at the set of users to which the item is recommended, and then we compute the total pairwise dissimilarity between those users; then we sum over all items in $\mathcal{I}_c$.

The average diversity (per item and per-user pair) is written as follows.

$$\overline{\text{Div}}_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \frac{\sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}: v \neq u} d_{uv} x_{iu} x_{iv}}{\Big(\sum_{u \in \mathcal{U}} x_{iu} \Big) \times \Big(\sum_{u \in \mathcal{U}} x_{iu} - 1 \Big)} \tag{2}$$

which also takes values in $[0, 2]$.

Coverage: Another aspect of a recommendation algorithm is that it should recommend items from each category to an adequate number of users, i.e. make sure that items from each category appear in the recommendation lists of enough users. Coverage can be defined in different ways, e.g. as the cardinality of the set of users to which the items of category c are recommended is called the *coverage* of category c. The average user coverage may be defined

as,

$$\text{Cov}_c = \sum_{u \in \mathcal{U}} \min\{1, \sum_{i \in \mathcal{I}_c} x_{iu}\} \,. \tag{3}$$

The coverage of a category c is the number of users to which at least one item of category c is allocated. If for a user u there are more than one items of category c recommended to this user i.e. if $\sum_{i \in \mathcal{I}_c} x_{iu} > 1$, this user counts for one in the coverage.

In this project, we choose to adopt *a simpler form of coverage*, which is counted as the number of items that items of category c appear in the recommendation lists of users. The average per-item user coverage is

$$\text{Cov}_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} x_{iu} \,. \tag{4}$$

While performing the changes in the recommendation lists from the traditional (CF based) ones to the new ones, we would like to minimize the cost of change to users. For example, if the baseline recommendation system recommends items to a user i with predicted rating $\{r_{ij}\}$, we would like the new recommender system to recommend an item with as similar rating to $\{r_{ij}\}$ as possible.

2 Problem A: Recommendation for Diversity and Coverage

Let $L = |\mathcal{L}_u| = |\mathcal{L}'_u|$ denote the number of items that are recommended to each user (i.e. L is the length of the recommendation list), where $\mathcal{L}_u$ is the set of items recommended to user u through the traditional CF algorithm, and $\mathcal{L}'_u$ is the set of items to be recommended with the new approach. Assume that each user is initially recommended some items $j \in \mathcal{L}_u$ through the baseline item-item CF algorithm. These lists will need to change in order to satisfy a constraint for adequate diversity. The cost for user u is:

$$\text{Cost}_u = \sum_{j \in \mathcal{L}_u} r_{ju} - \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}_c} \sum_{u} r_{iu} x_{iu} \,. \tag{5}$$

The objective is to do the changes in the recommendation lists of users so as to minimize the total average created cost (inconvenience) to users. This is written as follows

$$\min_{\mathbf{x}_i, i=1,\ldots,|\mathcal{U}|} \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \text{Cost}_u \tag{6}$$

where $\mathbf{x}_i = \{x_{iu}\}$ is a $0-1$ vector denoting the items recommended to user u. This problem is equivalent to

$$\max_{\mathbf{x}_i, i=1,\ldots,|\mathcal{U}|} \frac{1}{|\mathcal{U}|} \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} r_{iu} x_{iu} \,. \tag{7}$$

Namely that we would like to recommend those items that result in the maximum total predicted rating for user u. The maximum value of the above is clearly equal to $5L$ since the maximum rating of each item is 5 stars.

The following constraint needs to be fulfilled:

$$\sum_{c} \sum_{i \in \mathcal{I}_c} x_{iu} = L, \text{ for each user } u \tag{8}$$

namely L items should be recommended (substituted) to each user, since the length of the recommendation list is K.

We now think that intuitively we would like to have the items recommended to enough users that are also diverse enough on average. Therefore, the following constraints should hold,

$$\overline{\text{Div}}_c \geq D \,, \tag{9}$$

$$\text{Cov}_c \geq W \tag{10}$$

where $D \in (0, 2)$ is a predefined threshold on average per user-pair diversity and W is a threshold on the number of users to whose recommendation lists there are recommended items of any category c. Note that W could also be defined as a minimum percentage of users to which items of category c are recommended.

Therefore we need to solve problem (7) subject to constraints (8) for each user u and subject to constraints (9) and (10) for each category c.

3 Problem A: Numerical results

For Problem A, we are going to experiment with the `Movielens` `ml-latest-small.zip` (size: 1 MB) dataset from site:

`https://grouplens.org/datasets/movielens/`

Careful: there are several Movielens datasets, we need this one. Unzip, and from this dataset take the file `ratings.csv`. Read also the README.txt file in the zip for more information.

Each line of this file `ratings.csv` after the header row, represents one rating of one movie by one user, and has the following format:

`userId,movieId,rating,timestamp`

The lines within this file are ordered first by userId, then, within user, by movieId. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

The dataset contains $100,000$ ratings from 671 users and $9,125$ movies.

3.1 Suggested step-by-step process

Step 1: Download dataset.

Step 2: Data pre-processing of the dataset. We will need to filter appropriately the dataset. For example, we may need to consider only those users that have rated a certain number of movies.

Step 3: Produce the baseline recommendation for users. Use the item-item Collaborative Filtering (CF) method to derive the recommendation.

The output of item-item CF should be the user-item ratings matrix R. Then, for each user u (row), look at the ratings r_{ui} of items in this row. The baseline recommendation list $\mathcal{L}_u$ for each user u consists of the top-L rated items for each user i.e. the L items for which the rating r_{ui} is largest.

You could take e.g. $L = 5$ or $L = 10$, the size of the recommendation list.

Remark Look at https://github.com/revantkumar/Collaborative-Filtering and specifically at the file results2.csv that has implemented code in Python for generating item-item CF recommendations. You are advised to look at these implementations, understand them and modify them to produce your own (which will be different). As we said in class, there are several approaches to item-item CF for predicting the ratings.

Step 4: Computation of the similarity w_{uv} between any pair of users u and v. To do that, we can use the method outlined in Unit 3. For any pair of users u, v, see what are the movies that both users have rated in common, and compute the similarity w_{uv} as the similarity between the ratings of the two users for the common set of movies e.g. by using the similarity metrics we saw.

Step 5: For $L = 5$ and for $L = 10$, and for different values of (D, W) solve **Problem A** with a solver in Python or with MATLAB.

(a) As a first step, solve Problem A with continuous variables $x_{iu} \in [0, 1]$. Record the value of total rating. This is an upper bound on the value of total rating if variables are assumed integer. Python cvxpy or MATLAB function fmincon would work. Other choices are also possible.

(b) Solve Problem with discrete values $\{0, 1\}$ for variables x_{iu}.

For information on how to solve Non-linear Programming integer problems with Python, you can use cvxpy.

Step 6: Plot the total average rating of recommended items to all users (7) vs. the value of D for different values of W. Namely, horizontal axis will be D as the diversity of items e.g. points $D = 0$, $D = 0.1$, $D = 0.25$, $D = 0.5$, $D = 0.75$, $D = 1$, $D = 1.25$, $D = 1.5$, $D = 1.75$, $D = 2$. Vertical axis is the total average rating produced over all recommendations. In one plot, we could have the curves for $W = 1\%$ of the users, $W = 3\%$ and $W = 5\%$, and in another plot $W = 7\%$, $W = 10\%$, $W = 15\%$.

Try these plots for $L = 5$ and for $L = 10$ to see what happens.

Give the results in plots and on a table.

You can try the following solutions to the problem.

- Upper bound as generated by Nonlinear (Quadratic) Programming (continuous-valued variables $x_{iu} \in [0, 1]$).
- Exact solution as solved by an nonlinear integer programming solver. For example, check: `https://pypi.org/project/Pyomo/`. To be precise, our problem is a Integer quadratic programming problem, because of the quadratic constraint on the diversity.

4 Problem B: Serendipity in Recommender systems

A second objective, different than that of Problem A, is to do the recommendation so as to maximize the amount of *serendipity* in the system. Serendipity expresses the element of surprise of an item that is recommended to a user. We are going to assume that there are a few items to be recommended for serendipity e.g. $L_s = 2, 3$.

Again we will assume C categories of items, and let $\mathcal{I}_c$ be the set of items of category c,. Assume for now that we have somehow computed an index s_{iu} for each user u and item i that denotes the extent to which item i will surprise user u when it is recommended to him/her. Then, the goal is to maximize the overall average serendipity over all users

$$\max_{\mathbf{x}} \frac{1}{|\mathcal{U}|} \sum_{c=1,\ldots,C} \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} s_{iu} x_{iu} \tag{11}$$

subject to

$$\sum_{c} \sum_{i \in \mathcal{I}_c} x_{iu} = L_s \text{ for each user } u \tag{12}$$

and

$$\overline{S}_c = \frac{1}{|\mathcal{I}_c|} \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} s_{iu} x_{iu} \geq \theta \text{ for all categories } c \tag{13}$$

where θ is a minimum amount of average serendipity per item achieved for items of category c through recommendation to each user. Note that $x_{iu} \in \{0, 1\}$.

The maximum amount of θ is $5|\mathcal{U}|$, which is achieved if the item is recommended to all users and achieves maximum serendipity in all of them.

4.1 Suggested computation of serendipity

The serendipity index s_{iu} that an item i will surprise user u if recommended to him can be computed as follows. We select a few features that in our opinion comprise serendipity. These are,

- The extent to which the movie is *different* from other movies that the user has watched/rated until then. A movie that is not so similar to other movies that the user has seen/rated in the past has higher chances to surprise the user.
- The popularity of an item. The less popular the item is, the more likely it is to surprise the user.

In order to compute the serendipity index, we first need. a training dataset, with which we train a linear or non-linear regression model. We visit:
`https://grouplens.org/datasets/serendipity-2018/`
and download the `serendipity-sac2018.zip` dataset. Read the README.txt file and also read the paper [1] in the references below.

From the dataset `serendipity-sac2018.zip`, we need the files `movies.csv` and `answers.csv`.

The file `movies.csv` has all movie ids and different features and keywords. We are interested in the following features:

- movieId – movie id
- title – movie title
- directedBy – directors separated by commas (‘,‘)
- starring – cast separated by commas (‘,‘)

- genres – genres separated by commas (‘,‘)

From the file `answers.csv`, we are interested in the following

- userId – user id (481 users)
- movieId – movie id (1,678 movies)
- timestamp – timestamp, which indicates when the user gave the rating.
- s1 – ‘The first time I heard of this movie was when MovieLens suggested it to me.‘
- s2 – ‘MovieLens influenced my decision to watch this movie.‘
- s3 – ‘I expected to enjoy this movie before watching it for the first time.‘
- s4 – ‘This is the type of movie I would not normally discover on my own; I need a recommender system like MovieLens to find movies like this one.‘
- s5 – ‘This movie is different (e.g., in style, genre, topic) from the movies I usually watch.‘
- s6 – ‘I was (or, would have been) surprised that MovieLens picked this movie to recommend to me.“
- s7 – ‘I am glad I watched this movie.‘
- s8 – ‘Watching this movie broadened my preferences. Now I am interested in a wider selection of movies.‘

Fieds s1-s8 correspond to user ratings of the statements in our survey. Ratings are given using the scale, where 1 corresponds to ‘strongly disagree‘, 2 – ‘disagree‘, 3 – ‘neither agree nor disagree‘, 4 – ‘agree‘, 5 – ‘strongly agree‘, NA – ‘don’t remember.

We will create a model that will predict the serendipity index for any movie and for any user.

Look at each row of file `answers.csv`. For each movie i viewed by a user u in a single row:

- Find the similarity s_{ij} between this movie i and other movies j that user u has rated. The movies that u has rated some more movies that can also be found in the same file, `answers.csv`. The similarity can be found in different ways, such as (i) (an easy way): by using Jaccard similarity with the features from the file `movies.csv`, such as common words in genres, same starring actors, same directors, (ii) (a not so easy way): by using the file `ratings.csv` file (either from dataset `ml-latest-small.zip` or by the dataset `ml-latest.zip`. You will perhaps need to take the average of this similarity over the number of movies the user has rated.

- The popularity of an item can be computed through the (normalized) probability that a movie is rated by a user. This can be extracted through the dataset `ml-latest-small.zip` or by the dataset `ml-latest.zip` and. We can look at the fraction

$$p_i = \frac{\text{number of users that have rated item } i}{\text{number of users that have rated movies}} \tag{14}$$

  We can then take the $-\log_{10} p_i$ or any other metric to have the popularity.

Optionally, you could also use as a third attribute, the **predicted rating** of an item (e.g. the average rating), which is also included in file `answers.csv`.

Note that to compute the similarity above, s_{ij} between a movie i and other movies that the user has rated, we will need the timestamp as well. For a given movie i that is rated by a user, we need to compute the similarity with movies that the same user has rated, and they had the smaller timestamp.

The training dataset should look as follows:

`Movie, similarity with all past movies of a user, popularity, Serendipity index of movie,` S_i

If you use predicted rating of a movie as a third attribute, there will be three attributes above.

The serendipity index (i.e. the label) of a movie can be computed through e.g. taking the average of answers of users to questions s5, s6, s7, i.e.

$$S_i = \frac{1}{3}(s5 + s6 + s7)\,. \tag{15}$$

Other possibilities for the definition of serendipity index are possible. As you will see in the literature, there are several definitions of serendipity. With this dataset, we can build a prediction model. Possibilities for the machine-learning model to use, are:

- Linear regression model with two attributes, the similarity to past watched movies, and the movie popularity.
- Non-linear regression models, with the two attributes above. For example, one could try to fit to the data a function of the form

$$\text{serendipity} = \frac{1}{\beta \times \text{similarity} + \gamma \times \text{popularity}} \tag{16}$$

Other choices are also possible, Given this dataset, we will then predict for an unseen movie i, the serendipity index for a user u, s_{iu}. Note than an unknown, unseen movie i will have different predicted serendipity index for each user u due to the fact that the movie has different similarity to the already seen movies of that user. Try the model with a certain set of users and movies, and solve the formulated problem (11) subject to (12) and (13).

5 Problem B: Numerical results

First, it would be interesting to try to see what is the best fit to the data? Is i

We are interested in solving the problem (11) subject to constraints (12) and (13) and $x_{iu} \in \{0, 1\}$. This problem can be solved as follows.

- Case B1: Solve the problem for integer values of $x_{iu} \in \{0, 1\}$.

- Case B2: Solve the problem with continuous variables, x_{iu}, such that $0 \leq x_{iu} \leq 1$. Are the results of B1 and B2 the same or different?

You may need the Python pymprog library. You will also need to have GLPK solver installed. You can download source files from `https://sourceforge.net/projects/pymprog/`.

Plot the total serendipity vs. θ for some 4-5 appropriate different values of θ that you will have to choose. Also give results for $L_s = 2$ and $L_s = 3$. Give the results both in a plot and in a table.

We are interested to see also the effect on including the predicted rating as a third attribute to the (e.g. linear regression) model. One aspect would be to calculate the predicted ratings of items recommended to each user and see which one of the two alternatives achieves highest ratings.

Note for the entire project: You can select a given number of categories, e.g. $10 - 20$ categories and randomly assign items to each category.

6 References

[1] Denis Kotkov, Joseph A. Konstan, Qian Zhao, and Jari Veijalainen. 2018. Investigating Serendipity in Recommender Systems Based on Real User Feedback. In Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),