Recommender systems optimization for coverage, diversity and serendipity

Deadline: March 23, 2020

No Institute Given

Abstract. We introduce a framework for optimizing recommender systems in the presence of constraints on metrics such as coverage, diversity and serendipity. We consider the setting where there exist several sets (categories) of items, and each set belongs to or is produced by a different entity. These items need to be recommended to users. The first problem we address is the following. Given a set of ratings for each user and each item, generated through a conventional recommender system algorithm, our goal is to recommend items to users such that the ratings of items in the recommendation lists are maximized, subject to maintaining sufficient coverage and diversity for each set of items. That is, items need to be recommended to a diverse enough set of users (with respect to their profiles), and each set of items needs to be recommended to a certain number of users.
The second problem is to maximize the total serendipity of recommended items subject to guaranteeing a given minimum serendipity for each set of items. Serendipity can be defined in various ways. Here we define it through a combination of item popularity and its difference from items a user has experienced so far. We show that these problems may be viewed as assignment problems. We will perform extensive numerical experiments with real data that verify the findings of our approach.
With this problem, we aim to gain knowledge and experience working with real datasets and also advance the fundamentals of optimization.

Keywords: Recommender systems, optimization, diversity, coverage, serendipity.

1 Introduction

In recommendation systems (RSs), the traditional performance objective to be optimized is the Mean Square Error. This measures the squared difference between predicted ratings obtained through a specific recommendation algorithm (e.g. item-item collaborative filtering or matrix factorization), and the true ratings that users have entered. This metric is defined either over all items or over the lists of items that are recommended to each user and it captures the overall *accuracy* of prediction of the recommendation algorithm. However, it does not take into account several other performance evaluation metrics of recommender

systems, such as diversity, coverage or serendipity, all of which are known to have a significant impact on user quality of experience (QoE).

More often than not, RSs are embedded in larger online services, e.g. an online retail store or a social media site, and a number of entities besides the end-users may be interested in the results of the recommendation algorithm. These entities may be the item owners, producers, providers or advertisers, and they may have agreements with the recommendation engine about item promotion and specific user outreach through recommendation, in exchange for some payment. Thus, items may be restaurants or hotels of different brand chains, books of different editor and publishing companies, products of different firms, movies of different producer and distribution companies, or other sponsored items for which their owners have paid to have them appear in users' recommendation lists.

In the traditional recommendation problem, e.g. in a hotel recommendation system, the algorithm selects for each user the top-L (e.g. $L = 5, 10$ or 20) items with the more highly predicted ratings and recommends them to the user with no further constraint. Further, the recommendations are issued separately for each user. However, in a scenario such as the one discussed above, the challenge faced by the recommendation engine is that it should guarantee different performance metrics such as the ones mentioned above *for each* involved entity that is interested in the results of the recommendation algorithm. This makes the recommendation problem significantly more composite and challenging. However, in the case when there exist multiple entities, i.e. hotel chains, and a coverage constraint exists i.e. each hotel chain should appear in the recommendation lists of some users, then the recommendation algorithm should make the recommendations jointly for all users. Furthermore, there exist certain constraints on performance metrics. The constraint couple either the recommendations of users or the recommendations of different entities.

In this project, we explore the way recommendation algorithms should be built under different performance evaluation metrics for recommendation systems that constitute constraints for how the recommendation is performed. We consider the setting where there exist several sets of items, and each set belongs to or is produced by a different entity. These items need to be recommended to users. Besides the traditional metric of accuracy of rating prediction, we will study diversity, coverage, and serendipity. 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. Coverage says that each item or set of items needs to be recommended to at least a certain number or percentage of users. Finally, serendipity has within it the notion of surprise to the user. Serendipity in this project can be defined through a combination of item popularity and its difference from items a user has experienced so far. Or

it can be defined based only on the item's difference from items that the user has experienced so far.

1.1 Our contributions

The contribution of our work to the literature is as follows.

- We introduce a framework for optimizing recommender systems in the presence of constraints on metrics such as coverage, diversity and serendipity.
- We formulate and study the following problem: given a set of ratings for each user and each item, generated through a conventional algorithm, our goal is to recommend items to users such that the ratings of items in the recommendation lists are maximized, subject to maintaining sufficient coverage and diversity for each set of items.
- We also study a similar problem in its optimization objective subject to guaranteeing a given minimum serendipity for each set of items.
- We show that these problems may be viewed as assignment problems and provide different approaches for solving them.
- We perform experiments with real data in order to validate the benefits of our optimization-driven approach.

2 Model and 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 $\mathcal{I}_c$ the set of items of category c, $c = 1, \ldots, C$. For each user u and item i of a category, let r_{iu} denote the predicted rating with the baseline RS algorithm.

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 $2, 5, 10$ etc. 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.

The problem we consider is as follows. 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 enough 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 in the sequel.

2.1 Deviation from baseline recommendations

We would like to find a new list of recommended items $\mathcal{L}'_u$ for each user u, with $|\mathcal{L}'_u| = L$, so that items of each category are recommended to users so that they fulfill certain constraints.

Let $\mathcal{U}_i$ be the subset of users to which item $i \in \mathcal{I}_c$ is recommended according to the baseline RS algorithm, and let $|\mathcal{U}_i|$ be its cardinality. Let $\mathbf{x} = (x_{iu} : i \in \mathcal{I}, u \in \mathcal{U})$ denote the **new** $0-1$ recommendation policy that we would like to find, where for each item i and user u, the binary variable $x_{iu} = 1$ if item i is recommended to user u in the new lists of recommended items $\mathcal{L}'_u$, and $x_{iu} = 0$ otherwise.

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. The total deviation is defined as the difference of average ratings in the baseline lists of recommended items $\{\mathcal{L}_u\}$ and new ones $\{\mathcal{L}'_u\}$,

$$\text{Cost}(\mathbf{x}) = \frac{1}{L|\mathcal{U}|}\left(\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{L}_u} r_{iu} - \sum_{i\in\mathcal{I}}\sum_{u\in\mathcal{U}} r_{iu}x_{iu}\right) \tag{1}$$

2.2 Coverage

Another requirement is that the recommendation algorithm should recommend items from each category c to an adequate number of users, i.e. make sure that items from each category appear in the recommendation lists of enough users. *Coverage for a category* c can be defined in different ways, e.g. as the number of users to which items of category c are recommended, defined as

$$\text{Cov}_c(\mathbf{x}) = \sum_{u\in\mathcal{U}} \min\{1, \sum_{i\in\mathcal{I}_c} x_{iu}\}\,. \tag{2}$$

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

In this project, we can adopt a simpler form of coverage for mathematical tractability. For each item i separately, we count the number of users to whom the item is recommended. The average per-item user coverage for items of category c is

$$\text{Cov}_c(\mathbf{x}) = \frac{1}{|\mathcal{I}_c|}\sum_{i\in\mathcal{I}_c}\sum_{u\in\mathcal{U}} x_{iu}\,. \tag{3}$$

We are going to place the constraint that

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

i.e. that for each category, the total number of times an item of category c is assigned to a user should be $K_c|\mathcal{I}_c|$, where K_c takes values somewhere in the interval $[0, \ldots, |\mathcal{U}|]$.

2.3 Diversity

Let $\mathcal{I}_c$ be the set of items of category c. In the diversity metric we are interested to assign items in $\mathcal{I}_c$ in as diverse a set of users as possible in order to expand their reach to new audiences. 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, 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. Let d_{uv} be the dissimilarity between users u, v, defined as $d_{uv} = 1 - w_{uv}$, so that $0 \leq d_{uv} \leq 2$, or as another decreasing function of w_{uv}.

The per-item diversity of users to which items from category c are recommended is as follows:

$$\text{Div}_c = \frac{1}{\mathcal{I}_c} \sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}: v \neq u} d_{uv} x_{iu} x_{iv} \,. \tag{5}$$

For example consider a category with 2 items, item i1 and i2, and assume 3 users, u1, u2 and u3. Assume i1 is recommended to users u1, u2, and i2 is recommended to all three users u1, u2, u3. The total diversity is

$$\text{Div} = \frac{1}{2}(d(u1, u2) + d(u1, u2) + d(u2, u3) + d(u1, u3)) \tag{6}$$

We may normalize through dividing with

$$\frac{1}{2}\Big(\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} x_{iu}\Big)\Big(\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} x_{iu} - 1\Big). \tag{7}$$

E.g. in the example above, we would need to divide with $\frac{1}{2} \times 5 \times 4$, where 5 is the total number of times an item is assigned to a user.

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

$$\overline{\text{Div}}_c = \frac{2}{|\mathcal{I}_c|} \frac{\displaystyle\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}: v \neq u} d_{uv} x_{iu} x_{iv}}{\left(\displaystyle\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} x_{iu}\right) \times \left(\displaystyle\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} x_{iu} - 1\right)} \tag{8}$$

which is simplified to

$$\overline{\text{Div}}_c = \frac{2}{|\mathcal{I}_c|} \frac{\displaystyle\sum_{i \in \mathcal{I}_c} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{U}: v \neq u} d_{uv} x_{iu} x_{iv}}{K_c |\mathcal{I}_c| \times (K_c |\mathcal{I}_c| - 1)\Big)} \tag{9}$$

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 $\mathcal{I}_c$.

3 Problem A: Recommendation for Diversity and Coverage

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, namely to minimize the cost function in (). This is written as follows

$$\min_{\mathbf{x}} \text{Cost}(\mathbf{x}) \tag{10}$$

where $\mathbf{x} = (x_{iu} : i \in \mathcal{I}, u \in \mathcal{U})$. This problem is equivalent to

$$\max_{\mathbf{x}} \frac{1}{L|\mathcal{U}|} \sum_{i \in \mathcal{I}} \sum_{u \in \mathcal{U}} r_{iu} x_{iu} \tag{11}$$

Namely, we would like to recommend those items that result in the maximum total predicted rating for user u.

The following constraint needs to be fulfilled:

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

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

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,

$$\overline{\text{Div}}_c \geq D\,, \tag{13}$$

$$\text{Cov}_c = K_c \tag{14}$$

where D is a predefined threshold on average per user-pair diversity and K_c is the constraint in coverage discussed above. Note that K_c could also be defined as a minimum percentage of users to which items of category c are recommended.

Therefore we need to solve problem (4.1) subject to constraints (12) for each user u and subject to constraints (13) and (14) for each category c.

4 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.

4.1 Suggested step-by-step process

You could set $K_c = K$ for all categories c. Also, if the scale of the problem becomes large, consider running the numerical experiments for a subset of users and a subset of items i.e. a submatrix of the user-item ratings matrix. There should be some hundreds of users.

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, K) solve **Problem A** with a solver/library in Python or with MATLAB.

(a) As a first step, you can 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` package or MATLAB function `fmincon` function 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`. Our problem has a linear objective in terms of $\mathbf{x}$, it has a nonlinear inequality constraint (on the diversity) and a linear equality constraint (on the coverage).

Step 6: Plot the total average rating of recommended items to all users (4.1) vs. the value of D for different values of K. Namely, horizontal axis will be D as the diversity of items e.g. points $D = 0$, $D = 0.1$, $D = 0.25$, $D = 0.5$, $D = 0.75$, $D = 1$, $D = 1.25$, $D = 1.5$, $D = 1.75$, $D = 2$. Doublecheck, there may be other values of D that give meaningful plots. Vertical axis is the total average rating produced over all recommendations namely the value of the objective (). In one plot, we could have the curves for $K = 1\%$ of the users, $K = 3\%$ and $K = 5\%$, and in another plot $K = 7\%$, $K = 10\%$, $K = 15\%$. In general, try various values.

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

Give the results in plots and on a table.

Thus, to summarize try solving the problem with one following solutions to the problem.

- Upper bound on the total rating, 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.
- There is a third method, through a heuristic algorithm, and this is described in the draft paper `RECSYS-2019.pdf` paragraph 3.1.1. You may want to try that. Or to design your own heuristic.

5 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. $\mathcal{L}_u| = 2, 3$.

There exist several prevalent definitions of serendipity. Here, we select a (small) number of features that in our opinion best comprise serendipity for an item and a user:

- The extent to which the item is *different* from other items that the user has experienced until then. An item that is not so similar to other items that the user has seen 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.

Assume that we recommend a small set of items (say two items) to a user with the aim to improve serendipity. In that case, we need to guarantee that if the user views one item, the second item would still be diverse enough from the first one. So overall, for a list of (two-three) recommended items, we want that each of them separately surprises the user, but also if the user sees the one of them, then the second one will still surprise her.

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.

The main difference from Problem A is that we cannot rely any more on ratings $\{r_{iu}\}$ from the baseline RS. That is why we will assume that we have an index s_{iu} for each user u and item i that denotes the extent to which item i will surprise user u if it is recommended to her.

Let again $\mathbf{x}$ be the vector that determines which item will be recommended to which user.

The goal is to maximize total average serendipity to users,

$$\max_{\mathbf{x}} \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} s_{iu} \Bigg(\sum_{i \in \mathcal{I}} x_{iu} + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}} d_{ij} x_{iu} x_{ju} \Bigg) \tag{15}$$

where d_{ij} is the dissimilarity between items i and j. This can again be computed via Pearson similarity or cosine similarity over a set of features.

There are the following constraints

$$\sum_{c} \sum_{i \in \mathcal{I}_c} x_{iu} = L_s \text{ for each user } u \tag{16}$$

which says that L_s items will be recommended to each user u, and also that there should be a minimum average serendipity for items of category c,

$$\overline{S}_c = \frac{1}{|\mathcal{I}_c|} \sum_{u \in \mathcal{U}} \Bigg(\sum_{i \in \mathcal{I}_c} s_{iu} x_{iu} + \sum_{i \in \mathcal{I}_c} \sum_{j \in \mathcal{I}} d_{ij} x_{iu} x_{ju} \Bigg) \geq \theta \text{ for each category } c \tag{17}$$

where θ is a minimum average amount of serendipity required for items of category c. Note that $x_{iu} \in \{0, 1\}$

5.1 Suggested computation of serendipity (data-assisted)

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

$$p_i = \frac{\text{number of users that have rated item } i}{\text{number of users that have rated movies}} \tag{18}$$

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.

$$S_i = \frac{1}{3}(s5 + s6 + s7)\,. \tag{19}$$

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

$$\text{serendipity} = \frac{1}{\beta \times \text{similarity} + \gamma \times \text{popularity}} \tag{20}$$

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 (15) subject to (16) and (17).

6 Problem B: Numerical results

We are interested in solving the problem (15) subject to constraints (16) and (17) and $x_{iu} \in \{0, 1\}$. This problem can be solved as follows.

First, you can solve the problem above, assuming in the objective (15) you do not take into account item dissimilarity, i.e. it is $d_{ij} = 0$. The the problem becomes **linear**.

- Case B1: Solve the problem above 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. You can also give examples with real users i.e. fix attention to a user that has seen and rated movies x,y,z already. Then what is the movie that that is recommended to her as a result of increasing serendipity?

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.

Then you can solve the current problem with the objective (15) as it is. Then the problem has a quadratic constraint.

Note for the entire project: You can select a given number of categories, e.g. $5 - 20$ categories and randomly assign items to each category.

7 Guidelines for the project deliverables

For Report 1, you will need to read:

1. The paper by D.Kotkov et.al in the references of this document
2. The draft paper RECSYS-2019.pdf. This is similar (but not exactly the same) as the problem described in the current report. It might be useful to you.
3. Optionally, other material from the web.

For Report 2, you can study the material:

1. The paper "Recogym..."
2. The presentation "slides RL recsys"
3. The presentation "+++POLY-KALO..". It is a tutorial on Reinforcement learning
4. Articles such as https://medium.com/inside-machine-learning/recommendation-systems-using-reinforcement-learning-de6379eecfde. Google "recommender systems reinforcement learning".

All the material above is in the eclass folder "PROJECT 2020–>Project material".

The deliverables for the project are:

1. **Report 1**: A 10-15 page report (pdf) presenting the procedure you followed, what numetical packages, libraries etc you used, and the *results* on the solution of problems A, B. Remember we want the results in plots and in tables.

2. **Report 2**: A 2-3 page report on the topic "Recommender systems and reinforcement learning". Here the expectation is to describe how can reinforcement learning help address problems/scenarios in recommender systems, what are these and briefly present/discuss them. It is a research topic report.
3. A 20 minute presentation (i.e. 15 slides) with the results that each group will present on March 23. 10 out of the 15 slides will be describing Report 1, while 5 of the 15 slides will be describing Report 2.

You can send these 3 to me by March 23 in a **single** `.zip` file, under the name `XXX-YYY-ZZZ-WWW`, where XXX, YYY, ZZZ, WWW are the last names of the people of the group. Groups of 4 are allowed.

8 References

References

1. . Kotkov, J. A. Konstan, Q. Zhao, and J. Veijalainen, "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)