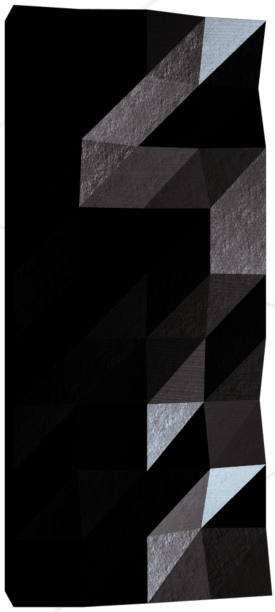

Project 02 – Myspotify

Recommender systems. Music recommendations

Summary: This project is an introduction to algorithms used for recommendations: non-personalized, content-based, collaborative filtering.

Version: 1

Contents

Chapter I

Preamble

Imagine how YouTube would look like without the recommendation section on the right of a video? How would Facebook look like without its smart feed and the people you may know? How would Amazon look like without book recommendations? What would the media look like without the section "Most popular" or "Most commented" or "Related news"?

Recommender systems are ubiquitous. Some of them became the core of a product, some of them are just great features. Anyway, they are important for the products. Why is it so?

Making a choice is hard for people. It is a complicated cognitive task. Anything that can help us with it is useful. Before the Internet was born we had already used recommendations: friends, family, colleagues. If they recommended something to us, we considered it as a piece of valuable information. Bestsellers in bookstores or blockbusters in cinemas is another technique to help us make a choice - read or watch something that other people like. Even the power of brands is based on the problem of choice - build trust and do not ruin it, and a customer will buy your goods or services just because they do not want to solve the complicated problem.

Recommender systems are useful, first of all, for users and customers, but they are a great tool for companies too: the algorithms help them increase revenues by upselling or cross-selling goods that are appealing for the users. It is a win-win situation - that is why recommendations are so valuable and important.

Chapter II

Introduction

What is under the hood of recommender systems? You are quite familiar with machine learning, but although recommendations can be created by applying some of those algorithms (by predicting a rating or by predicting the probability that a user clicks, likes or reads), there are several specific approaches to that domain.

Non-personalized recommender systems. They are useful when we do not know anything about a new user or customer. We can recommend to them something popular in different terms: bestsellers, blockbusters, most commented, most trending, most popular, top-10 in the genre, etc. But to create such recommendations we need to have data about how other users or customers made their choices. It is not very useful when you just start a new online shop or media - cold start problem.

Content-based recommender systems. To create that kind of recommendation we need to have some items descriptions: several paragraphs about books, short descriptions of goods, plots of movies, and so on. Comparing the descriptions, we may recommend a user similar items. Another option is to create collections of similar items. It does not require any information on the past user behavior, and it can be done in the case of starting something new from scratch.

Collaborative filtering. It is considered an advanced technique. In this case, we start making recommendations personally. Thus, we need data about the past behavior of that particular user. The first approach, user-to-user, is we recommend items that were bought by the users similar to that particular user. The second approach, item-to-item, is we recommend items that have similar profiles in terms of user behavior (suit and tie, for example).

Matrix factorizations are considered to be a part of collaborative filtering algorithms, but they use more advanced techniques: transforming the initial user-item matrix to a matrix of factors and making recommendations based on those latent factors (can be genres, for example).

Chapter III

Goals

The goal of this project is to give you a first approach to recommender systems. You will try all the mentioned approaches. At the same time, you will think about how to create a good product based on them. It will impact the metrics that you will use to assess your solutions.

Chapter IV

Instructions

- This project will only be evaluated by humans. You are free to organize and name your files as you desire.
- Here and further we use Python 3 as the only correct version of Python.
- The norm is not applied to this project. Nevertheless, you are asked to be clear and structured in the conception of your source code.

Chapter V

Mandatory part

a. Task

```
In this project, you will work on a music recommender system. There are tons
of different songs and tracks. Your goal is to help users to discover something
they will like and play it on repeat! As we said, in order to do that you will
try different approaches.
```

- Top-250 tracks. Non-personalized approach.
- Top-100 tracks by genre: Rock, Rap, Jazz, Electronic, Pop, Blues, Country, Reggae, New Age. Non-personalized approach.
- Collections: 50 songs about love, 50 songs about war, 50 songs about happiness, 50 songs about loneliness, 50 songs about money. Content-based approach.
- dataPeople similar to you listen: 10 recommendations for each user. Collaborative filtering approach.
- dataPeople who listen to this track usually listen: 10 recommendations for each track. Collaborative filtering.

b. Dataset

You are lucky and you do not have to collect the dataset by yourself. With the help of the data science community, you have access to parts of Million Songs Dataset (MSD):

- 1. The Echo Nest Taste Profile Subset.

  ```
  The format is the following (user_id, song_id, play_count) triplets, and
  each line looks like this (tab-delimited):
  b80344d063b5ccb3212f76538f3d9e43d87dca9e    SOAKIMP12A8C130995    1
  b80344d063b5ccb3212f76538f3d9e43d87dca9e    SOAPDEY12A81C210A9    1
  b80344d063b5ccb3212f76538f3d9e43d87dca9e    SOBBMDR12A8C13253B    2
  ```

- 2. The musiXmatch Dataset.

This file stored in a sparse format. It contains track_id, mxm_track_id, then word count for each of the top words, comma-separated. Word count is in sparse format -> ...,<word idx>:<cnt>,... <word idx> starts at 1 (not zero!)

- 3. Tagtraum genre annotations for the Million Song Dataset.

  This file contains 3 fields: track_id, majority_genre, minority_genre.

- The mapping between track ids and song ids.

  The file contains 4 fields: track_id, song_id, artist, title.

You can collect additional data if you find it useful for you.

You can find the dataset in the project page :
1.p02_train_triplets.txt.zip
2.p02_mxm_dataset_train.txt.zip
3.p02_msd_tagtraum_cd2.cls
4.p02_unique_tracks.txt

c. Implementation

In your research process, you can work in Jupyter Notebooks. After that, you need to organize your code in classes and methods. In the end, you need to create a Python script that makes the recommendations mentioned above.

Top-250 tracks

It should return a dataframe with the following fields: index number, artist name, track title, play count. The table should be sorted by the play count descendingly.

Top-100 tracks by genre

It should return on a given genre a dataframe with the following fields: index number, artist name, track title, play count. The table should be sorted by the play count descendingly. You should only use the major genre to perform the subtask.

Collections

It should return on a given keyword (love, war, happiness) a dataframe (50 tracks) with the following fields: index number, artist name, track title, play count. The table should be sorted by the play count descendingly. Try different approaches to these recommendations:

- baseline - when you look for the keyword and the number of its occurrences in a song, filter using some threshold and then sorting it by the play count,
- word2vec - when you look not only for the keyword but for several similar tokens as well using word2vec,
- classification task -you may label your data and try classification algorithms that will predict for the other part of the dataset if a track belongs to a specific class.

Maybe you find some other interesting ideas on how to make those recommendations better.

People similar to you listen

For these recommendations, you need to use the train/test split approach. In this case, the best practice is to cut a sub-matrix from the user-item matrix for the test dataset and the other parts to use for the train.

To assess your recommendations use the metric p@k (precision at k). It shows the percentage of the correct recommendations from your list. It means, that if you gave a user 10 tracks to listen and if they liked 3 of them (they really listen to them in the test dataset), then the p@k will be equal to 30%. Calculate the average p@k for your recommendations. It should be at least greater than 10%.

The script should return 10 recommendations for a given user in a dataframe: index number, artist name, track title. The table should be sorted descendingly by the "likelihood" that any given user will "like" the track.

People who listen to this track usually listen

The same things applied to these recommendations: use train/test split, use p@k. If you gave a user 10 tracks to listen and if they liked 3 of them (they really listen to them in the test dataset), then the p@k will be equal to 30%. Calculate the average p@k for your recommendations. It should be at least greater than 10%.

The script should return 10 recommendations for a given track in a dataframe: index number, artist name, track title. The table should be sorted descendingly by the "likelihood" that any given user will "like" the track.

d. Submission

You need to prepare two files for your repository: the Jupyter Notebook where you conducted your research as well as the python script. You may keep there additional files that you may find useful for your program.

Chapter VI

Bonus part

Visit any music streaming service (ex. Spotify, Apple Music, Yandex Music, etc.), find 3 elements of their recommender systems (it can be any recommendation block that you have not already implemented), and repeat them in your project.

Chapter VII

Submission and peer-correction

`Submit your work on your Git repository as usual. Only the work on your repository will be graded.`

`Here are the points that your peer-corrector will have to check:`

- all 5 recommender sub-system exist,
- all the exceptions are handled,
- dditional recommender elements added from the bonus part.