Universität Freiburg
Institut für Informatik
Dr. Fang Wei-Kleiner
Victor Anthony Arrascue Ayala

Georges-Köhler Allee, Geb. 51
D-79110 Freiburg
fwei@informatik.uni-freiburg.de
arrascue@informatik.uni-freiburg.de

Lab Course: Distributed Computing Using Spark
Summer Term 2021
Published on 29.06.2021

4. Experiment: Evaluating a Content-based Recommender System

Deadline: 24.07.2021

Prolog. The goal of this experiment is to leverage the approaches explored in previous experiments to implement two kinds of Content-based Recommender Systems (CBRS) [1] and to conduct an off-line evaluation to evaluate their performances. A CBRS typically requires the following components:

a) A representation of each item in the catalog (item profile).
b) A representation of each user, which models her taste (user profile). This has to be defined in the same dimensional space in which the item profiles are defined.
c) A component which is able to measure the similarity between a user and item profile and which reflects the degree to which an item matches the taste of a user.

The first component has been already achieved in experiment 3. Therein papers were modeled using either the TF-IDF weighting scheme (exercise 3.2, $\vec{p}_{tf-idf}$) or LDA (exercise 3.4, $\vec{p}_{LDA}$).
The second component has been achieved in exercise 3.3a and 3.4.b by aggregating the paper representations for each user with respect to her library ($\vec{u}_{tf-idf}$ and $\vec{u}_{LDA}$, respectively).

In this experiment you will implement the third component to complete the design of a CBRS. Leveraging this component the CBRS can rank for each user u all papers in the catalog C with respect to her profile and return the top-k papers with the highest scores.

Exercise 4. 1 (Accessing a Spark cluster, 0 points)
Computing recommendations even for a single user can be a computationally expensive task especially if the catalog is large. Therefore, you will have the possibility from now on to access the cluster at our department. Further instructions will be provided separately.
To verify if you have access to the cluster and you are able to run an application follow these simple steps:

- Upload the file "users_libraries.txt" to your *HDFS* home folder [4].
- Create a small python application[1] which reads that file and implements the following basic operation: count the number of distinct users and distinct papers. Save this information to HDFS.
- Launch your application using spark2-submit [2].

[1]In the cluster only Python 2 is supported, while you have been implementing your solutions in Python 3. Make sure your application works with Python 2 before you run it in the cluster. These differences will not have a big impact in your code [3].

Exercise 4. 2 (Content-based recommendations: similarity metric, 5 points)
In order to generate the recommendations a CBRS has to be able to compute the similarity between two profiles $\vec{u}$ and $\vec{p}$. We will use the cosine similarity as the similarity metric:

$$sim(\vec{u}, \vec{p}) = cos(\theta) = \frac{\vec{u} \cdot \vec{p}}{||\vec{u}||\ ||\vec{p}||} = \frac{\sum_1^n u_i \times p_i}{\sqrt{\sum_1^n (u_i)^2} \times \sqrt{\sum_1^n (p_i)^2}} \tag{1}$$

Implement this component as a function.

Exercise 4. 3 (Content-based recommendations, 10 points)
Now that all three main components are implemented it is possible to create a CBRS, which can recommend top-k papers to a user u as follow:

- First, the CBRS computes the similarity between the user profile and each item in the catalog C, excluding those which are already in her library (P_u^+): $sim(\vec{u}, \vec{p}), \forall p \in \{C \setminus P_u^+\}$.
- The CBRS ranks all papers $p \in \{C \setminus P_u^+\}$ based on their similarity score with respect to $\vec{u}$.
- The CBRS returns *top-k* items, i.e. those with the highest scores.

Reproduce these steps in order to implement two CBRSs:

a) Implement $CBRS_{tf-idf}$, a recommender system that recommends top-k papers to a user u based on $\vec{u}_{tf-idf}$.
b) Implement $CBRS_{LDA}$, a recommender system which recommends top-k papers to a user based on $\vec{u}_{LDA}$.
c) Show top-k recommendations for the user with user hash id = 1eac022a97d683eace8815545ce3153f generated with $CBRS_{tf-idf}$ and $CBRS_{LDA}$, respectively.

Implement the CBRS as functions: input to a CBRS is a user profile $\vec{u}$ and k, the number of recommendations to be delivered. Note that in both recommender systems you can reuse the same similarity component implemented in the previous task. The only difference is the input (the kind of profiles) it gets.

Exercise 4. 4 (Off-line evaluation metrics, 15 points)
A key element to measure the performance of a Recommender System (in terms of quality of recommendations) are evaluation metrics. To compute a ranking metric for a specific user you require 1) a list of the top-k recommendations for her, and 2) her test set, i.e. the items that she like, but were not used in building the Recommender System's model.

Let TS_u be the test set of user u. The papers which were recommended which are in TS_u are called *hits*. Nearly all ranking metrics are based on the number of hits. For example: given the following TS_u: {7, 12, 19, 66, 10} of user u and the following top 10 recommendations (3, 4, 5, **7**, **12**, 43, 2, **10**, **66**, 18), then the hits are {**7**, **12**, **10**, **66**}.

Let a metric score for a user u be $Metric_u@k$. This is computed taking into account (the first) k top recommendations. Implement three functions which compute $Precision_u@k$, $Recall_u@k$[2], and $MRR_u@k$. The input must be given as a DataFrame containing three columns: (user hash id, test set, top-k recommendations). The size of the list to be considered ($@k$) must be an argument of the function. If the list of top-k recommendations is larger than k, only the first k elements must be taken into account in computing the score. In the example above, when k is set to 5, the top-k list becomes (3,4,5,7,12), for $k = 3$ it becomes (3,4,5), etc. This makes it possible to generate a large list of recommendations, e.g. 50, at once, and compute all metrics for $k < 50$ without having to generate recommendations again and just by reducing the number of elements. The implemented function must then return the same dataframe with an added column which contains the score (for each user) for the given metric.

On top of that you can compute a system metric *Metric@k* by simply computing the score average of all users. The system metric formulas are listed below:

- $Precision@k = \frac{1}{|U_s|} \sum_{u \in U_s} \frac{|Hits|}{k}$
- $Recall@k = \frac{1}{|U_s|} \sum_{u \in U_s} \frac{|Hits|}{|TS_u|}$

[2]Precision and recall can be computed in the same function.

- $MRR@k = \frac{1}{|U_s|} \sum_{u \in U_s} \frac{1}{p_u}$
  where p_u is the position of the first hit in the list of top k recommended papers. In the previous example $p_u = 4$.

Exercise 4. 5 (Off-line evaluation, 20 points)
Use the sampler you implemented in experiment 3 and the functions you implemented in the previous task to compare the performance of two Recommender Systems on 20 sampled users. More concretely, compute *Precision@k*, *Recall@k*, and *MRR@k* for $k \in \{5, 10, 30\}$ and compare the performances of $CBRS_{tf-idf}$ and $CBRS_{LDA}$ based on the given metrics.

Which recommender performs the best according to the results of those metrics?
Justify your answer. Write detailed comments in your Jupyter Notebook.

References:

1. *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. G. Adomavicius, A. Tuzhilin.* `http://ieeexplore.ieee.org/document/1423975/`

2. *Submitting Spark applications:* `https://spark.apache.org/docs/latest/submitting-applications.html`

3. *What is new in Python 3:* `https://docs.python.org/3.0/whatsnew/3.0.html`

4. *HDFS list of commands:* `https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html`