GitHub Recommender System

Big Data Computing Project
2020/2021

Fabrizio Rossi 1815023
Matteo Orsini 1795119

Goal

- **GitHub** is the largest source code host
    - 40+ million users
    - 200+ million repositories
- GitHub users can **star** interesting repositories
    - Used to suggest new useful repositories
- Our project goal is:
    - Given a list of starred repositories or an already existing user
    - Predict new repositories which may be interesting

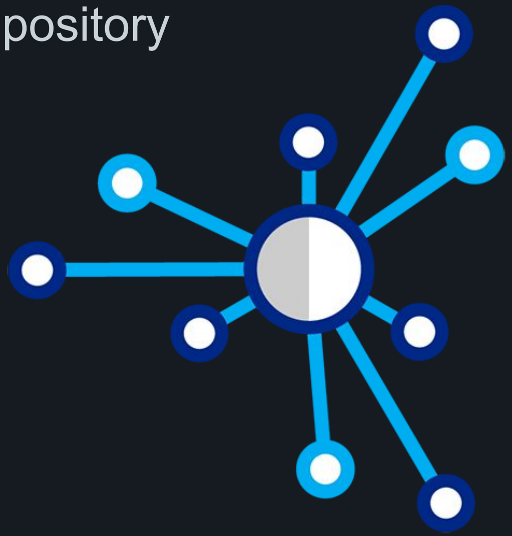
Challenges

- Find suitable **dataset**:
  - We couldn’t, so we collected it ourselves
- We don’t have ratings: **implicit feedback**
- Implementation of models with PySpark
- Deal with a large amount of data

Data collection

Dataset collection

- Custom Python scripts with *requests* and *BeautifulSoup*
- Started from users that starred the ***React.js*** repository
- Collected repositories starred by them
- Collected repositories metadata
  - creator, name
  - about (short description)
  - main programming language
  - stars, forks
  - updated
  - sponsor (a form of donations)
- Total: 10.000 users and 354.981 repositories



Dataset Pruning

Dataset pruning I

- Some models require to process DataFrames with **100+ billions of records**
    - Even if number of users/repos is restricted, number of comparisons is huge
- Unfortunately neither Colab or Databricks can provide us the necessary computational power for free
- So, we **pruned** the dataset obtaining:
    - Users: 1.000
    - Repositories: 1.000
    - Starred relationships: 104.499

Dataset pruning II

- **How** we did it:
    - Took 1.000 users with highest number of starred repos
    - Took 1.000 most starred repos among those starred by the previous users
    - Discarded all user-repo relationships not relative to the previous users/repos
- Even with these numbers, still have many records:
    - Example: Item-Based Collaborative Filtering
    - Dataframe with *`users`* x *`ns_repos`* x *`s_repos`* records
    - Total: 64.413.366

Models

Content-Based Filtering I

- Based on **features** extracted from repositories
  - Doesn't need to know ratings from other users
- Needs **feature engineering**:
  - Text processing for the *about* field
    - lowercase, punctuation, tokenization
    - stopwords
    - stemming
    - tf-idf
  - One-Hot Encoding for the *language* field
  - Time conversion for the *updated* field
  - MinMax Normalization for the fields *stars, forks, updated*
- Build **item profiles**

Content-Based Filtering II

- Build **user profiles**
  - Prepare for each user the vectors of the starred repositories
  - Compute the user profile as the average of those vectors
    - We used the Summarizer class offered by PySpark
    - *groupBy* user id and compute mean of starred repos vectors
    - It works even with sparse vectors, so we can save some RAM
- Compute **cosine similarity** between each user profile and all item profiles of the repos that the user has not rated

Content-Based Filtering III

- Take top-k most similar repos for each user
  - We used a Window function
    - Partition by user id
    - Sort by similarity
    - Filter by row number

user_id	repo_id	similarity
1	2	0.92
1	3	0.56
1	4	0.88
2	1	0.92
2	3	0.31
2	4	0.25

Content-Based Filtering III

- Take top-k most similar repos for each user
  - We used a Window function
    - Partition by user id
    - Sort by similarity
    - Filter by row number

user_id	repo_id	similarity
1	2	0.92
1	3	0.56
1	4	0.88
2	1	0.92
2	3	0.25
2	4	0.31

Content-Based Filtering III

- Take top-k most similar repos for each user
  - We used a Window function
    - Partition by user id
    - Sort by similarity
    - Filter by row number

user_id	repo_id	similarity
1	2	0.92
1	4	0.88
1	3	0.56
2	1	0.92
2	4	0.31
2	3	0.25

Content-Based Filtering III

- Take top-k most similar repos for each user
  - We used a Window function
    - Partition by user id
    - Sort by similarity
    - Filter by row number

user_id	repo_id	similarity	row_number
1	2	0.92	1
1	4	0.88	2
1	3	0.56	3
2	1	0.92	1
2	4	0.31	2
2	3	0.25	3

Content-Based Filtering III

- Take top-k most similar repos for each user
    - We used a Window function
        - Partition by user id
        - Sort by similarity
        - Filter by row number

user_id	repo_id	similarity	row_number
1	2	0.92	1
1	4	0.88	2
1	3	0.56	3
2	1	0.92	1
2	4	0.31	2
2	3	0.25	3

User-Based Collaborative Filtering I

- No need to extract features from repositories
- Based on **user rating vectors**
  - Set of repositories starred by the user
- Steps:
  - Build user rating vectors
  - For each user, compute similarity among other user rating vectors
    - This time we used the **Jaccard similarity**
  - Take top-k most similar users (*Window function*)
  - Take repos starred by those users
  - Recommend them by assigning to each one the average rating given by other users

User-Based Collaborative Filtering II

- The **average rating** in our case is just the normalized count of top-k users that starred that repo:

$$r_{v,i} \in \{0, 1\} \qquad r_{u,i} = \frac{1}{k} \sum_{v \in \mathcal{U}^k} r_{v,i}$$

Item-Based Collaborative Filtering I

- Based on **item rating vectors**
  - Users that starred the repository
- Achieves better performances than User-Based but **high RAM usage**
- Steps:
  - Build item rating vectors
  - For each user we need to compute a score for each not starred repo:
    - Compute similarity with actually starred repos (**Jaccard similarity**)
    - Take top-k most similar starred repos (*Window function*)
    - Compute a score by aggregating the ratings of the found repos
  - Recommend non starred repos with highest obtained score

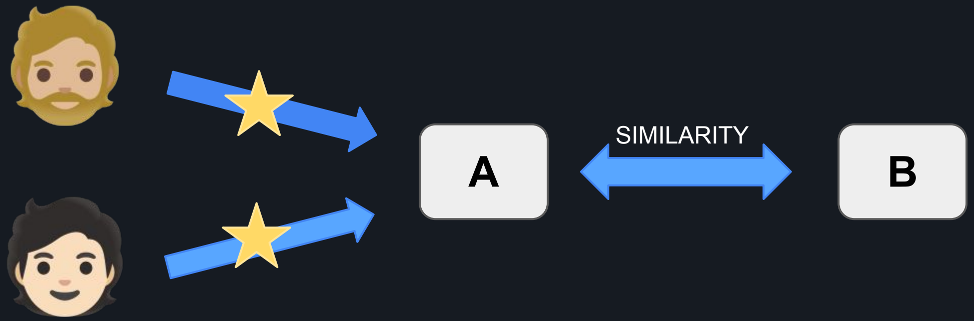
Item-Based Collaborative Filtering II

- In this case the **aggregation can't be performed** as a simple average since we have that all the ratings are 1
- We opted to use the **similarity between items** to compute the score, as follows:

$$r_{u,i} = \frac{\sum_{i' \in \mathcal{I}_u^k} sim(i, i')}{k}$$

Item-Based Collaborative Filtering III

- We have to **compute many times** the similarity between the same pairs of items
- So, we pre-computed the *Jaccard similarity* between each pair of item to make the computation faster



Matrix Factorization I

- **Ratings matrix** R seen as product of a user matrix X and an item matrix W:

$$R = X \times W^T$$

- X and W computed based on **hidden factors** extracted from observed starring relationships
- Score for a non-starred repo computed as dot product between user and repo vector

$$r_{u,i} = x_u^T \cdot w_i = \sum_{j=0}^{d} x_{u,j} w_{j,i}$$

Matrix Factorization II

- There are various methods to compute the two matrices
- **Alternating Least Squares (ALS)** method implemented and provided by PySpark
- Important parameters:
  - **rank**: rank of the latent matrices (5)
  - **implicitPrefs**: adapt implementation to implicit feedbacks (True)
  - **alpha**: feedback confidence (1.0)
- **Challenge**: how to predict repos for new user not in training?
  - Solution: non-standard approaches, not implemented in PySpark
  - So, in our case, train again the model (doesn’t take much time)

Evaluation

Evaluation

- We computed the MAP@K and the Personalization measure
- **Mean Average Precision at K** (MAP@K) is the mean of the Average Precision at K (AP@K) metric computed for each user
    - Takes into consideration precision of the system and order of the items recommended
- **Personalization** is defined as the average of the dissimilarity between users lists of recommendations
    - Tells us if the recommender system produces items which are personalized for each user, or always the same items to different users

Evaluation II

Model	MAP@1	MAP@2	MAP@3	MAP@4	MAP@5	Personalization
Content-based	0.067	0.051	0.043	0.037	0.033	0.676
User-based	0.357	0.261	0.213	0.183	0.164	**0.965**
Item-based	0.388	0.308	0.263	0.229	0.205	0.680
Matrix Factorization	**0.506**	**0.395**	**0.332**	**0.298**	**0.268**	0.864

- **Content-Based Filtering** low performances probably due to low discriminance of extracted repos features
  - about field very short and contains recurring terms ➞ similar TF-IDF vectors for different repos

Demo

Demo

- Built in **React** (frontend) + **Flask** (backend)
- Uses **ngrok** to host the server directly on Colab
- Predict repos starting from:
  - Already existing user of the system
  - A list of repos starred by a new user
- Available at https://recommend-hub.netlify.app/