UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

ALGORITHM FOR MASSIVE DATASET PROJECT REPORT

Bianchi Davide
12820A

academic year 2023-2024

Declaration

Indice

1 Introduction

The goal of this project is to implement a **similar movie pair detector** from scratch. Each movie will be represented by aggregating information from various files in the dataset in a meaningful and non-trivial way. The process involves several key steps:

- **Dataset Exploration** – Analyzing and investigating the dataset.
- **The Inverted Index Structure** – Building an efficient indexing structure for token retrieval.
- **TF-IDF Measure** – Computing term frequency-inverse document frequency for feature representation.
- **Cosine Distance** – Implement and measure the similarity between movie representations.
- **Similarity Detection on the Movie Dataset** – Identifying and linking similar movie pairs in the massive dataset.
- **Experiments and Evaluation** - Evaluate the performance of the algorithm on a large-scale dataset.

In this project, Python 3.11 version is used along with the Apache Spark API 3.5.4 version, a powerful framework designed for processing Big Data distributed across multiple clusters.

2 Dataset Preparation

2.1 Dataset Information

The dataset used in this project is the Letterboxd Movies Dataset, which contains various information about films. It is composed by the following `csv` files along with their respective attributes:

- `actors.csv`: Id, Name, Role
- `countries.csv`: Id, Country

- `crew.csv`: Id, Role, Name
- `genres.csv`: Id, Genre
- `languages.csv`: Id, Type, Language
- `movies.csv`: Id, Name, Date, Tagline, Description, Minute, Rating
- `posters.csv`: Id, Link
- `releases.csv`: Id, Country, Date, Type, Rating
- `studios.csv`: Id, Studio
- `themes.csv`: Id, Theme

Additionally, there is a directory containing the image of the posters of the films not used in this project.

2.2 Dataset Sampling

To reduce the overall running time, the dataset is subsampled. This process involves selecting only 1% of the records from the `movies.csv` file and filtering the other files based on the selected movie IDs. The data are then organized in two directories:

- Full dataset: *dataset/full_ dataset*
- Subsampled dataset: *dataset/small_ dataset*

The algorithms can be run on both the subsampled dataset and the full dataset.

2.3 Dataset Preprocessing

In this section, all `csv` files are loaded into the distributed dataset. Each record in the Resilient Distributed Dataset (RDD) is represented as a key-value pair, where the key is the movieID, and the value is a string containing the remaining attributes of the record.

For each file, the preprocessing steps are as follows:

- Load the `csv` file.
- Parse each record and ensure it follows the expected data pattern.
- Remove any quotation marks from the records.
- Store the processed data in the format: `(movieID, string containing the attributes of the record)`. If a record does not follow the specific pattern it's discarded.

After this initial preprocessing, the documents are represented using the *Bag of Words* model. This model encodes each document in terms of a set of words, or more precisely, tokens, that appear in it. The preprocessing steps for each document in the RDD are as follows:

- Retrieve the document from the RDD.
- For each record `(movieId, string)`, split the string in tokens and remove any stop words encountered.
- Store the result as a new record in the format: `(movieId, [list of tokens])`.

Once these preprocessing steps are completed, the dataset is ready for use by the algorithms described in Section 4.

3 The Inverted Index structure

An inverted index is a data structure commonly used to efficiently search and retrieve documents that contain a specific token (or word). It maps tokens (terms) to the list of documents or records that contain them, rather than mapping documents to the tokens they contain.
The idea behind an inverted index is that instead of scanning through every document to find a token, for each token is stored a list of documents in which it appears. With this structure it is easy to quickly retrieve documents that contain a specific token without having to look through every document individually. In this project this structure is implemented by the *index* function.

4 Algorithms

This section presents the key algorithms used to identify similar movies. The first is the Term Frequency-Inverse Document Frequency (TF-IDF) model, which assigns varying importance to words based on their occurrence within individual documents and across the entire dataset.
Next, the Cosine Distance algorithm is presented, which quantifies the similarity between two documents using their TF-IDF representations.

4.1 Term Frequency - Inverse Document Frequency (TF-IDF)

To improve the *Bag of Words* technique, it is important to assign different weights to the tokens contained in a document to reflect which terms are more relevant. A heuristic that helps identify these terms is the TF.IDF measure (Term Frequency times Inverse Document Frequency), obtained by multiplying two indicators defined below.

Term Frequency
The Term Frequency gives more weight to tokens that tend to appear multiple times in the same document. It is calculated as the relative frequency of the token in the document. In other words, if the document d contains 100 tokens and among them the token t appears five times, the Term Frequency of t in d is $TF(t,d) = \frac{5}{100} = 0.05$. The term frequency is implemented by the function *tf*.

Inverse Document Frequency
The Inverse Document Frequency assigns a high weight to tokens that rarely occur in a dataset. Given a token t and a set of documents U, the Inverse Document Frequency of t is equal to $IDF(t) = \frac{N}{n(t)}$, where N is the total number of documents in U and $n(t)$ indicates the number of documents in U that contain t. The inverse document frequency is implemented by the function *idfs*.

TF-IDF
The TF-IDF measure of a token t in a document d is the product of the Term Frequency and the Inverse Document Frequency: $TF.IDF(t,d) = TF(t,d) \cdot IDF(t)$. A high TF.IDF value indicates a token that frequently occurs in a document but rarely in others. The term frequency - inverse document frequency is implemented by the function *tfidf*.

4.2 Cosine Distance

The measure of similarity between documents used in this project is the cosine distance, which interprets two objects as directions in a space and calculates the cosine of the angle formed by these directions.

The documents are encoded as directions in space, and therefore as vectors. Each possible token in our corpus represents a dimension: a generic document will have as the component in the dimension corresponding to a token the corresponding value of the TF.IDF measure.

4.2.1 Cosine Similarity Formula

The similarity between two documents can be measured by computing the cosine of the angle between the two directions in the space. This can be easily done by recalling that $a \cdot b = \|a\|\|b\| \cos\theta$, where $a \cdot b$ denotes the dot product between two vectors a and b, θ is the angle between these vectors, and $\|a\|$ denotes the norm of a.
Therefore:

$$\text{sim}(a, b) = \cos\theta = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum a_i b_i}{\sqrt{\sum a_i^2}\sqrt{\sum b_i^2}}$$

It should be noted that, although this approach involves considering a large number of dimensions, the vectors can be stored in a dictionary containing only the non-zero components, taking advantage of the sparsity property. More precisely, for each token t with a non-zero TF.IDF value $i \neq 0$, the key t in such a dictionary will be associated with the value i. The cosine similarity is implemented by the following functions: *dotprod*, *norm*, *cossim*, *cosine_similarity*.

4.2.2 Fast Cosine Similarity

In the project a faster cosine similarity has been implemented. This function use precomputed norms and tf-idf weight saved in a broadcast variable. This reduces communication costs between nodes and improves the performance because each worker has local access to the data.

5 Similarity Detection on the Movie Dataset

Similarity detection refers to the process of identifying how similar two or more items (such as objects, texts, images, or data records) are to each other. In this project, the goal is to identify pairs of similar movies by leveraging the functions defined in the previous sections. The key steps in the similarity detection process are outlined below:

- **1. Dataset preparation**
  - **1.1. Load the csv Files**
    Load the `csv` files (like actors.csv, genres.csv, etc.) into the distributed system. The format of each record is the following: `('MovieId', 'Attributes')` and they are saved in a Resilient Distributed Dataset (RDD).
  - **1.2. Convert the Data Format**
    Convert each data entry from `('MovieId', 'Attributes')` format to `('MovieId', [token list])` format. This involves processing the relevant fields (e.g., title, description, genres, etc.) to create a tokenized list for each movie.
  - **1.3. Create a Combined Dataset**
    Combine the relevant RDDs (e.g., movies, genres, actors, etc.) into a single RDD. This resulting dataset will facilitate efficient processing and comparison of movie records. The data format is the same: `('MovieId', [token list])`
- **2. Compute the TF-IDF values**
  - **2.1. Compute Inverse Document Frequency (IDF) Values**
    * Calculate the Inverse Document Frequency (IDF) values for the entire Combined Dataset. The IDF is used to weigh terms based on their frequency across documents, helping to highlight important (less common) words.
    * Store the computed IDF values as a broadcast variable. Broadcast variables in Spark are sent to the workers only once and stored locally for quick access, optimizing the computation process.
    * The data format is: `{'token1': idfsToken1, 'token2':idfsToken2, ...}`
  - **2.2. Apply TF-IDF Transformation**
    * Use the previously computed IDF values to perform a Term Frequency-Inverse Document Frequency (TF-IDF) transformation. This process generates an RDD that maps each `MovieId` to a dictionary of tokens, where each token is associated with its corresponding TF-IDF measure. The TF-IDF measure helps to evaluate the importance of each token in relation to the Combined Dataset.

    - The data format is: `[('MovieId1', {'token1':idfsToken1, 'token2':idfsToken2, ...}), ..., ('MovieId2', {'token1':idfsToken1, 'token2':idfsToken2, ...})]`

- **3. Create the final structure for Movie Comparisons**
  - **3.1. Invert the TF-IDF Weights**
    - Create an inverted index structure, which maps each token to the list of movie IDs containing that token. This index allows for fast retrieval of movies that share common tokens.
    - The RDD computed at point 2.2 is inverted and saved in the new inverted RDD.
    - The value of the TF-IDF are not considered. The data format is: `[('token1', 'MovieId1'), ('token2', 'MovieId1'), ..., ('tokenN', 'MovieIdN')]`
  - **3.2. Identify Common Tokens Between Combined Dataset Records**
    - Create a new RDD that includes only tokens shared between movies. Each element in this RDD is a pair where the key is a token, and the value is a `(MovieId1, MovieId2)` pair.
    - Swap the elements in each pair, so that the key becomes the `(MovieId1, MovieId2)` pair, and the value is the common token.
  - **3.3. Avoid Record with the Same Key**
    - Filter and remove the records that have the same film ids: eg. ('1010', '1010')
    - Sort the same key with switched movieIds: eg. ('1111', '0000') = ('0000', '1111') and remove the duplicates with distinct().
    - `groupByKey` the elements in the dataset and the final RDD every record maps each (MovieId1, MovieId2) to a list of common tokens. The data format is: `(('MovieId1', 'MovieId2'), [token list])`

- **4. Compute the Cosine Similarity using precomputed TF-IDF and Norms**
  - **4.1. Compute Vector Norms**
    - Generate an additional RDD that maps each `MovieId` to the norm of its corresponding TF-IDF vector.
    - Convert this RDD into dictionaries and store them as broadcast variables allowing fast access to the norms during similarity calculations.
    - The data format is: `[('MovieId1', normMovieId1),('MovieId2', normMovieId2), ...]`

- **4.2. Compute the Fast Cosine Similarity**
  * The Fast Cosine Similarity is computed for each record (pair of movies) in the Common Tokens Dataset based on their common tokens.
  * This function is more efficient because the tf-idf weights and the norm are stored in two broadcast variables. This reduces communication costs between nodes and improves the performance because each worker has local access to the data.
  * The result is an RDD that contains the cosine similarity scores. The data format is: `(('MovieId1', 'MovieId2'), Cosine Similarity)`

6 Experiments and Evaluation

In the previous section a general method to detect similarities between movies has been presented. In this section the goal is to aggregate in different ways the informations contained in the various files in the dataset and compute their similarities.

6.1 Test1: similarity based on Movie Name, Theme, Genre

In this section, the similarity between movies is done comparing: Movie Names, Themes and Genres. The histogram in figure 1 represents the distribution of cosine similarity scores between movies, based on their name, actor names, and studio. The x-axis shows the cosine similarity values, while the y-axis (log scale) represents the frequency of pairs with a given similarity. In the histogram:

- The majority of cosine similarity values are close to zero. This indicates that most movie pairs in the dataset have very little similarity if movies' name and movies' theme are compared.
- There is a rapid drop in frequency as similarity increases, meaning that highly similar movie pairs are rare. This is expected in large datasets with wide range of attributes where only a small fraction of movies share strong similarities in names, themes, and genres.
- There are very few cases where the cosine similarity exceeds 0.5, reinforcing the idea that most movies do not have highly overlapping feature representations.

Exploring some results of the similarity analysis, it is shown that this combination of data successfully identifies similarities in movie genres but also captures some contextual similarities found in the themes.

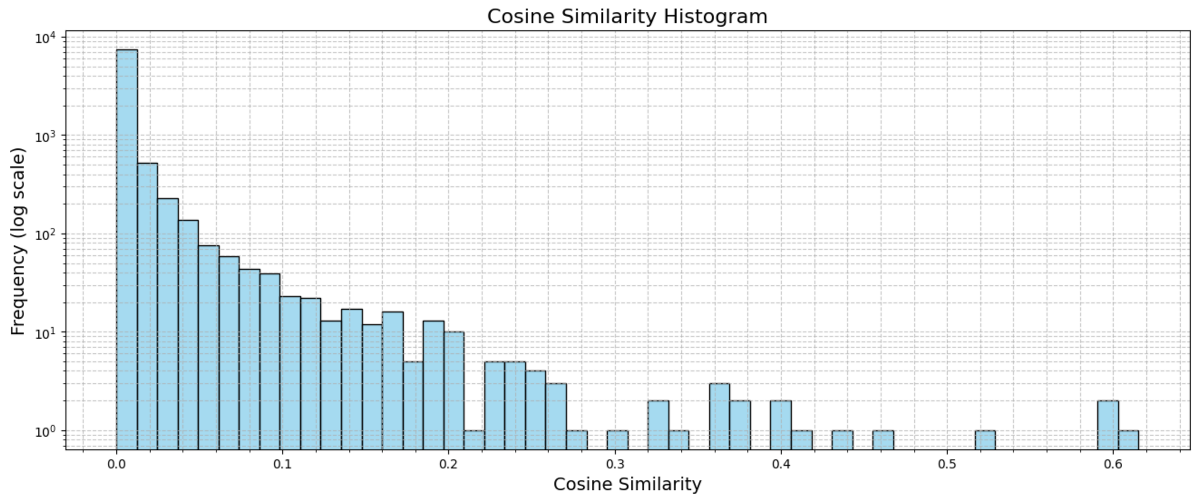

Figura 1: Test1

6.2 Test2: similarity based on Movie name, Actor Name and Studio

In this section, the similarity between movies is done comparing: Movie Name and Actor Name and the production Studio. This histogram in figure 2 represents the distribution of cosine similarity scores between movies, based on their name, actor names, and studio. The x-axis shows the cosine similarity values, while the y-axis (log scale) represents the frequency of pairs with a given similarity. In the histogram:

- The majority of movie pairs have a similarity close to zero, meaning they share few or no common features. As similarity increases, the frequency of pairs decreases, suggesting that highly similar movies (in terms of movie name, actors role and studio) are much less common.
- A few peaks at higher similarity values indicate that some movies share significant attributes, possibly sequels, remakes, or films from the same studio.

Exploring some results of the similarity analysis, it is shown that this combination of attributes identifies better similarities in the production studio, instead the information about actors and movie names are less relevant.

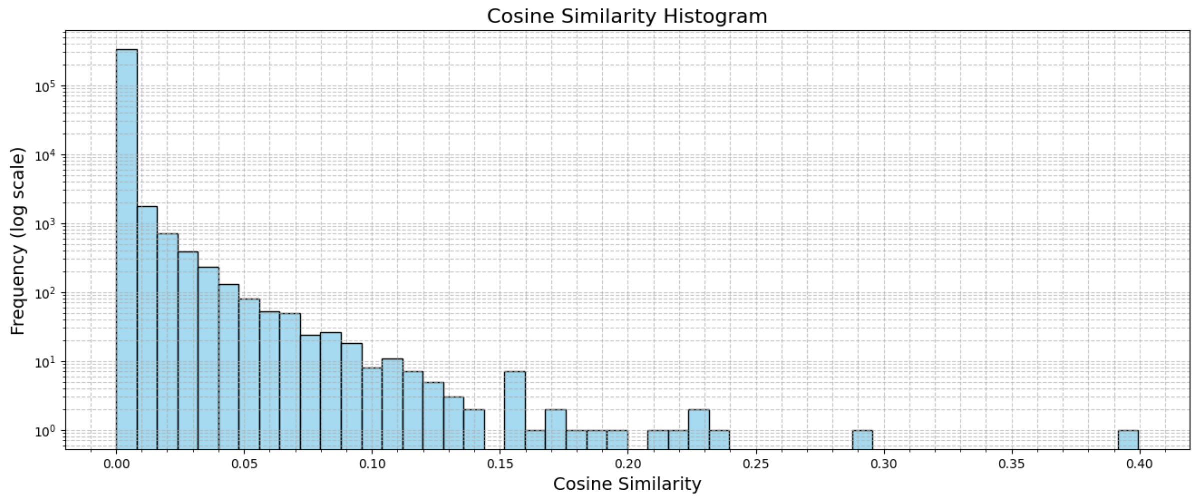

Figura 2: Test2

7 Final Consideration

As demonstrated in the previous section, the algorithm identifies some degree of similarity between movies based on specific attributes. Additional tests, not included in this report, were conducted, incorporating the movie descriptions and other textual information. One limitation of using cosine similarity with TF-IDF is that it can sometimes assign high weight to words that, despite being rare, are not particularly useful for distinguishing between movies. Further improvements could be made by incorporating more meaningful features or removing non-informative words that are rare.