**Title: Design and Implementation of a Recommendation System Using Big Data**

**Name:** NZIGAMASABO DANY-DURAN

**ID:** 204203800125

**Class:** Software Engineering 2031

**Abstract**

This paper presents a comprehensive exploration into the design and implementation of a recommendation system leveraging the power of Big Data. Focusing on both content-based and collaborative filtering approaches, the system is built using Apache Spark, a distributed computing framework. The implementation involves the Euclidean Distance metric for content-based recommendations and the Alternating Least Squares (ALS) algorithm for collaborative filtering. The paper discusses the intricacies of handling massive datasets, optimizing performance, and evaluates the system's predictive accuracy.

In the era of online shopping and digital content consumption, the overwhelming number of products and choices available pose a challenge for users to discover items tailored to their preferences. To address this issue, recommendation systems play a pivotal role in suggesting relevant products or content based on user behavior and preferences. This paper explores the design and implementation of a content-based recommendation system, focusing on its concepts, use cases, and practical applications.

**Keywords**

Recommendation Systems, Collaborative Filtering, Content-Based Recommenders, Alternating Least Squares (ALS), Matrix Factorization, Root Mean Squared Error (RMSE)

**Source code and dataset Repository**

We are using the very popular movielens dataset, which was collected by the GroupLens Research Project at the University of Minnesota. This dataset contains a list of movies that are rated by their customers. It has been simplified for this paper.

**Links**

Download or clone the repository at the link: <https://www.gitee.com/danyduran00/RecommendationSystem.git>

Link to the Gitee repository: [https://www.gitee.com/danyduran00/RecommendationSystem](https://gitee.com/danyduran00/RecommendationSystem)

Link to the full Dataset:

<https://grouplens.org/datasets/movielens/tag-genome-2021>

1. **Introduction**

When you visit a traditional bookstore to purchase books, you generally have a specific book in mind that you are interested in buying. In this scenario, you would search for that particular book among the bookshelves. Typically, the top-selling books at that moment are displayed upfront in the store, while the remaining inventory is organized on the shelves.

In a small bookstore, which may contain a few thousand books or more, the physical products available are within your immediate view as a customer. You have the freedom to choose and pick books based on your preferences during your visit. Although physical stores prioritize placing top-selling products at the forefront due to their higher likelihood of being sold, there is no personalized arrangement based on individual customer preferences.

Contrastingly, this situation changes when you explore popular online e-commerce platforms like Amazon or Walmart. These platforms boast an extensive catalog, potentially containing millions, if not billions, of products. The sheer volume of information available on an e-commerce website necessitates an elegant method for presenting the most relevant information to customers during their browsing sessions, encouraging them to make purchases.

Unlike physical stores where displaying top products was a common practice, this approach is not considered elegant for online platforms due to the limited space on the webpage. Consequently, to address this challenge, most online e-commerce stores employ sophisticated recommendation systems. These systems aim to enhance the customer experience by offering personalized suggestions based on individual preferences, ultimately influencing purchasing decisions on the website.

1. **General Terms and their Definition**
   1. **General Terms**

**Big Data:** Big Data refers to the vast and diverse set of data that surpasses the capacity and capabilities of traditional relational databases. It encompasses structured and unstructured data, including log files, videos, and various forms of information, stored and processed on distributed computing systems.

Three Vs of Big Data:

* **Volume:** Big Data involves a massive amount of data, often reaching terabytes or petabytes of storage. Traditional relational databases struggle to handle such extensive volumes efficiently.
* **Variety:** Big Data encompasses diverse types of data, ranging from text to images. It accommodates various sources, such as log files, spatial data from satellites, and information collected through sensors or mobile devices.
* **Velocity:** Refers to the speed at which data is generated, processed, and analyzed. Big Data systems operate in parallel on multiple machines, ensuring high-speed computations and insights extraction.

**Hadoop:** Hadoop is an open-source, Java-based framework that facilitates the distributed processing of large datasets across clusters of computers. It comprises three core components: Hadoop core, Hadoop Distributed File System (HDFS), and MapReduce.

Here are some features of Hadoop:

* **Failover Support:** Failover support ensures system robustness by transferring tasks from failed machines to functional ones. In Hadoop, failover support is crucial for maintaining uninterrupted data processing even when some machines experience issues.
* **Data Locality:** Data locality is a key feature in HDFS, enabling data processing programs to run closer to the data they analyze. This minimizes network transfer, resulting in faster and more efficient data processing.
  1. **Important Products of the Big Data Stack**

**Hadoop Distributed File System (HDFS**): HDFS is a distributed file system within Hadoop that provides high-performance access to data across Hadoop clusters. It is designed for immense scalability, fault tolerance, and data locality, ensuring efficient processing and storage of large datasets.

**MapReduce:** MapReduce is a programming model and framework used for parallel processing and generating large datasets in a distributed computing environment. It divides tasks into Map and Reduce phases, enabling efficient data processing on Hadoop clusters.

**Spark:** Spark is a cluster computing framework used for various purposes, including analytics, stream processing, and machine learning. It offers faster and more versatile processing compared to traditional batch processing systems like MapReduce.

**Kafka:** Kafka is a real-time stream processing engine which provides very high throughput and low latency.

* 1. **Important features of the Big Data Stack**

1. **NoSQL:** NoSQL databases, like Cassandra and HBase, are utilized in big data scenarios that require frequent reads and updates. They offer flexible data models and scalability for handling diverse types of data.
2. **Search:** Search engines, such as SOLR, are integrated with big data systems to make content easily searchable, particularly in scenarios involving large amounts of plain text data.
3. **Machine Learning Libraries:** Machine learning libraries, such as Spark ML, are integrated into big data frameworks to support advanced analytics and predictive analytics. These libraries enhance the capabilities of big data systems in handling complex machine learning tasks.
4. **Distributed Computing:** Distributed computing involves processing large volumes of data parallelly across multiple machines or nodes. It enhances speed, fault tolerance, and scalability in big data environments.
5. **Stream Processing:** Stream processing tools like Apache Spark Streaming and Apache Storm enable real-time data analysis and decision-making by connecting to sources of streaming data.
6. **Recommendation Systems and their Types**

Recommendation systems are becoming increasingly prevalent in our daily lives, and they are being used by a variety of websites and services to suggest products, videos, and other items that users might be interested in. These systems are able to make these suggestions by analyzing user data, such as their click history, ratings, and purchase history. This information is then used to create a profile of each user's preferences, which can then be used to recommend items that are similar to the items that the user has already interacted with.

Recommendation systems can be a valuable tool for users, as they can help them to discover new items that they might not have found on their own. They can also help users to save time by filtering out items that they are not interested in. However, it is important to note that recommendation systems can also be biased, as they may reflect the preferences of the majority of users rather than the preferences of individual users.

Here are some examples of how recommendation systems are being used today:

* **Amazon.com** recommends books, movies, and other products to users based on their past purchases and browsing history.
* **YouTube.com** recommends videos to users based on their past viewing history and interests.
* Fashion-related websites recommend clothing items to users based on their past purchases and browsing history.

In all cases, some form of machine learning techniques is used to predict based on historical data of user's browse history or product transactions.

* 1. **Content-Based Recommender System**

**Content-based recommendation systems** analyze the attributes or content of items to identify similarities and suggest similar items to users. For example, a movie recommendation system might consider factors like director, actors, and genre, while a news recommendation system might look for keywords in news articles.

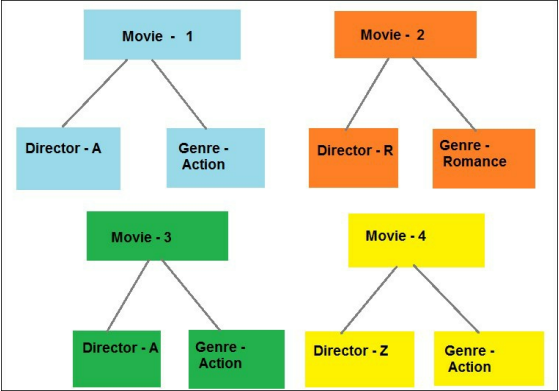


Figure 1 Diagram of a Movie Characteristics

The degree of similarity between items determines the order in which they are recommended. Mathematical techniques like cosine similarity, Euclidean distance, and Pearson coefficient are used to quantify these similarities.

* + 1. **Euclidean Distance**
* **Euclidean distance** is a mathematical formula used to calculate the distance between two points in a multidimensional space. In the context of content-based recommendation systems, Euclidean distance can be used to assess the similarity between two items based on their attributes.



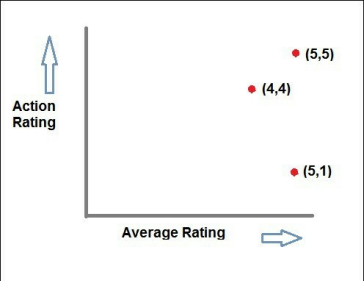


Figure 2 Two-dimensional chart

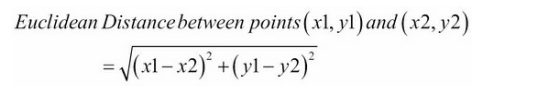


Figure 3 Euclidean formula

* + **Applying Euclidean distance in content-based recommendation systems:**
    - **Representing items as vectors:** Each item is represented as a vector of attribute values. For example, a movie could be represented by a vector containing its director, actors, genre, average rating, and action rating.
    - **Calculating pairwise distances:** The Euclidean distance is calculated between each pair of item vectors. This results in a matrix of pairwise distances, where each element represents the distance between two corresponding items.
    - **Identifying similar items:** Items with smaller Euclidean distances are considered more similar. Recommendation systems can use this information to suggest similar items to users.
  + **Advantages of Euclidean distance:**
    - **Simplicity:** Euclidean distance is a straightforward and easy-to-understand concept.
    - **Versatility:** Euclidean distance can be applied to items with a variety of attributes.
    - **Interpretability:** Euclidean distance values directly represent the similarity between items, making it easy to understand the recommendations.
    1. **Pearson Correlation**
* **Pearson correlation** is a statistical measure that quantifies the linear relationship between two data points. It is widely used in recommendation systems to assess the similarity between items based on their attributes.
  + **Key characteristics of Pearson correlation:**
    - Focus on linear relationships: Pearson correlation is best suited for identifying linear relationships between items. For non-linear relationships, other similarity measures may be more appropriate.
    - Handling user rating variations: Pearson correlation considers the tendency of users to rate items differently. It adjusts for these variations to provide a more accurate measure of similarity.
    - Calculating similarity: Pearson correlation is calculated using a mathematical formula that involves the mean and standard deviation of the attribute values for each pair of items.
  + **Advantages of Pearson correlation:**
    - **Sensitivity to rating variations:** Pearson correlation is less sensitive to variations in user ratings compared to Euclidean distance.
    - **Interpretable results:** Pearson correlation values range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.
  + **Limitations of Pearson correlation:**
    - **Assumption of linearity:** Pearson correlation assumes a linear relationship between the attributes being compared.
    - **Handling missing values:** Pearson correlation cannot handle missing values in the attribute vectors.

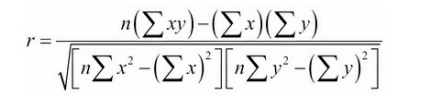


Figure 4 Pearson's correlation mathematical formula

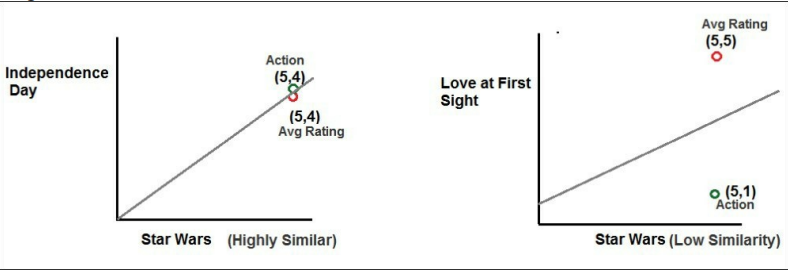


Figure 5 Implementation of Pearson's Algorithm on a Movie recommendation system

* + 1. **Cosine Similarity**
* **Cosine similarity** stands out as a powerful and widely used technique for assessing the similarity between items based on their attributes or content. Unlike Euclidean distance and Pearson correlation, which are primarily suited for analyzing numerical data, cosine similarity shines in its ability to effectively handle high-dimensional data, a common characteristic of item representations in recommendation systems.
  + **The Essence of Cosine Similarity**

At its core, cosine similarity measures the angle between two vectors in a multidimensional space. The closer the angle between the vectors, the more similar they are considered to be. This concept translates well to content-based recommendation systems, where items are represented as vectors of attributes, such as genre, director, actors, keywords, and other relevant features.

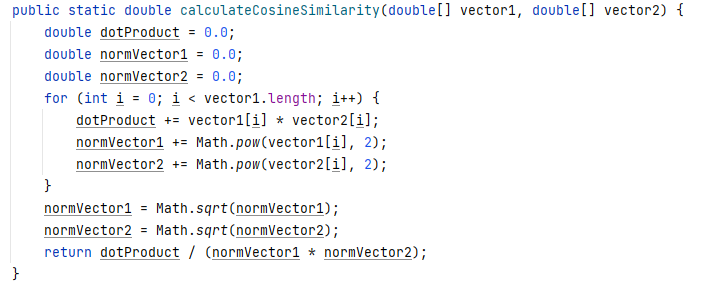
* + **Calculating Cosine Similarity**

To calculate cosine similarity, the dot product of the two vectors is divided by the product of their magnitudes. This formula ensures that the similarity score remains between 0 and 1, with 0 indicating no similarity and 1 indicating perfect similarity.

* + **Advantages of Cosine Similarity in Recommendation Systems**
    - **Handling High-Dimensional Data:** Cosine similarity is well-suited for analyzing item representations in high-dimensional spaces, where traditional distance measures may struggle.
    - **Focus on Feature Directionality:** Cosine similarity considers the direction of features, not just their magnitudes. This is particularly useful for identifying items with similar themes or concepts.
    - **Robustness to Feature Scaling:** Cosine similarity is unaffected by the scaling of features, unlike Euclidean distance, which can be sensitive to different scales.
    - **Interpretability:** Cosine similarity values provide a direct measure of similarity, making it easy to understand the recommendations.
  + **Limitations of Cosine Similarity**
    - **Assumption of Independence:** Cosine similarity assumes that features are independent of each other. However, in real-world data, features may be correlated, which could affect the similarity scores.
    - **Inability to Handle Missing Values:** Cosine similarity cannot handle missing values in the feature vectors.

**Cosine Similarity(A,B)=**

​



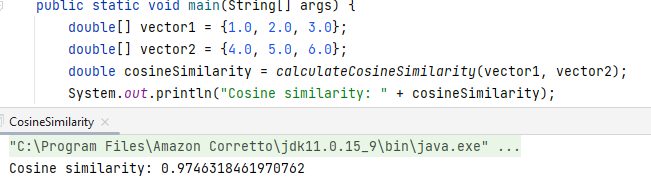


Figure 6 Simple Implementation of the Cosine Similarity

* + 1. **Case Study**

As stated at the beginning of the paper, we are using the very popular movielens dataset by the GroupLens Research project.

Here is an overview of the information we are going to use:

* **u.item:** contains the information about the movies in the dataset. The attributes in this file are:



Figure 7 Movie's Information

* **u.user:** This contains the demographic information about the users. The main attributes

from this file are:



Figure 8 Demographic Information of the users

In this project, we will create a simple content-based recommender using Apache Spark SQL and the MovieLens dataset. To determine the similarity between movies, we will use the Euclidean Distance metric. The Euclidean Distance will be calculated based on the genre properties (action, comedy, horror, etc.) of the movie items and also on the average rating for each movie.

**Setting Up SparkSession and Loading Rating Data**

* + Create Spark Configuration and SparkSession

|  |
| --- |
| SparkConf sconf = new SparkConf().setMaster("local[\*]");  SparkSession spark = SparkSession  .*builder*().config(sconf)  .appName("DataExploreMovieLens").getOrCreate(); |

* + Load Rating Data from u.data file using Spark RDD

|  |
| --- |
| JavaRDD<RatingVO> ratingsRDD = spark  .read().textFile("data/movie/u.data").javaRDD()  .map(row -> {  RatingVO rvo = RatingVO.*parseRating*(row); return rvo; }); |

* + Extract Data from Each Row and Populate RatingVO POJO
  + Create DataFrame from RDD and Register as Temporary View

|  |
| --- |
| Dataset<Row> ratings = spark.createDataFrame(ratingsRDD, RatingVO.class); ratings.createOrReplaceTempView("ratings"); |

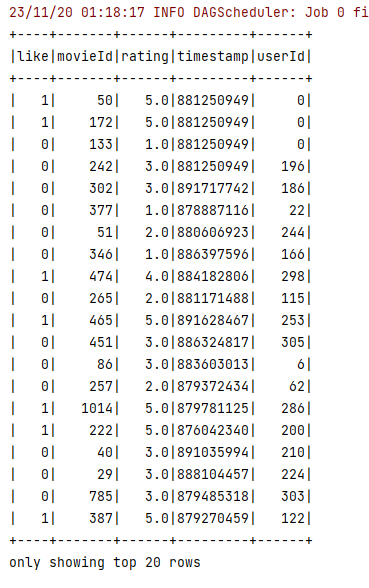


Figure 9 Output of the top 20 rows of Movie Ratings on our dataset

**Loading Movie Data:**

* + Load movie data from the 'u.item' data file.
  + Store movie data in a MovieVO POJO, including MovieId, MovieTitle, and genre information.
  + Genre information is stored as 1 or 0.
  + Use a JavaRDD to read and map the data, creating a Spark RDD called 'movieRdd.'
  + Convert 'movieRdd' into a Spark dataset using the createDataFrame function and the MovieVO POJO.

**Combining Datasets:**

* + Create a temporary view ('movies') for the movie dataset.
  + Combine the movie dataset with the 'moviesLikeCntDS' dataset using a self-join within a Spark SQL query.

**Self-Join Operation:**

* Perform a self-join within a Spark SQL query on the temporary view 'movies.'
* Create combinations of every movie with every other movie in the dataset, excluding self-combinations.
* The query produces duplicate combinations that need to be handled separately to avoid redundancy.

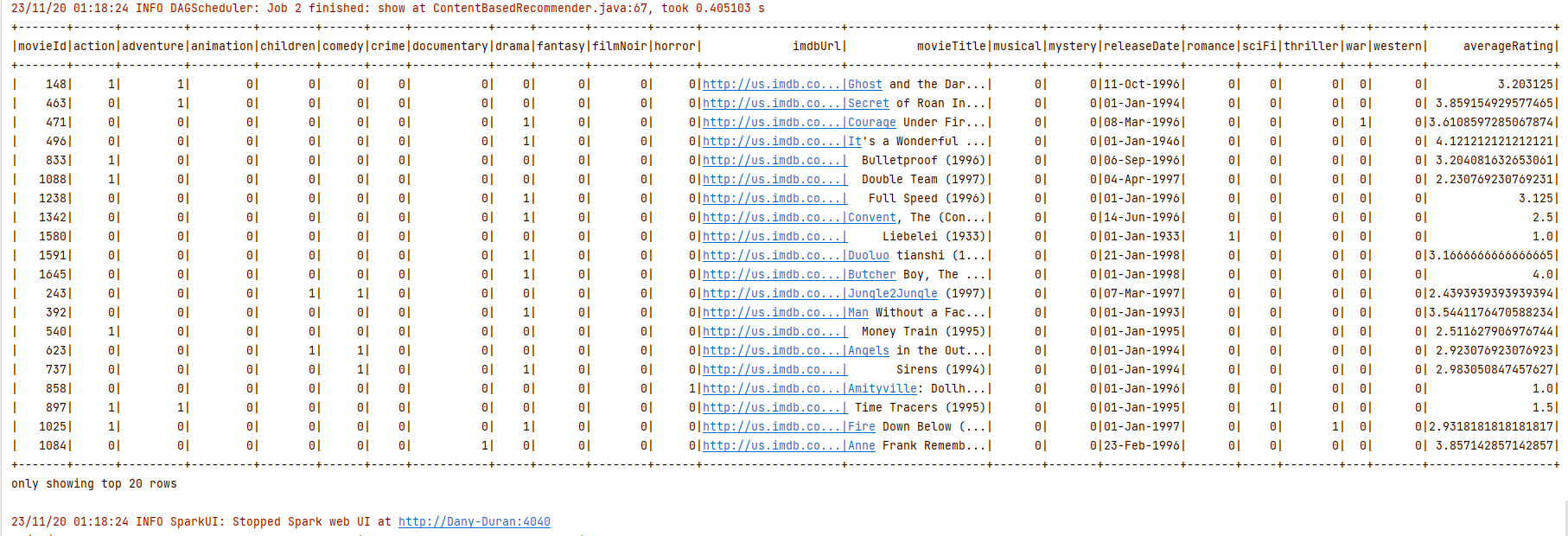


Figure 10 Output of the top 20 rows of Movie Informations from our dataset

In the realm of big data, processing large datasets can be computationally demanding and resource-intensive. To improve the performance of our content-based recommender, we can leverage Spark's distributed capabilities by utilizing the **spark-submit** command with the **executor-cores** and **num-executor** parameters. This allows us to distribute the Euclidean Distance calculations across multiple executors, significantly reducing processing time and enhancing overall efficiency.

A crucial step in our content-based recommender is calculating the Euclidean Distance between movies to determine their similarity. The Euclidean Distance measures the distance between two points in a multidimensional space, where each point represents a movie's attributes.

By converting our dataset to an RDD and employing a map function, we can efficiently calculate the Euclidean Distance for each pair of movies. Within the map function's lambda expression, we extract the relevant data from each row and compute the Euclidean Distance using the formula:

|  |
| --- |
| double euclid = Math.*sqrt*(action \* action + adventure \* adventure + animation \* animation + children \* children + comedy \* comedy + crime \* crime + documentary \* documentary + drama \* drama + fantasy \* fantasy + filmNoir \* filmNoir + horror \* horror + musical \* musical + mystery \* mystery + romance \* romance + scifi \* scifi + thriller \* thriller + war \* war + western \* western + likesCnt \* likesCnt); |

This formula considers the difference in genre, action rating, and likes between the two movies, providing a comprehensive measure of their similarity.

After calculating the Euclidean Distance for each pair of movies, we store the results in a **EuclidVO** Java POJO object. This object encapsulates the movie IDs, titles, and their corresponding Euclidean Distance.

Next, we convert our RDD of **EuclidVO** objects into a DataFrame and register it as a temporary view named **movieEuclids**. This enables us to perform SQL queries on the calculated Euclidean Distances for making recommendations.

|  |
| --- |
| Dataset<Row> results = spark.createDataFrame(euclidRdd.rdd(), EuclidVO.class); results.createOrReplaceTempView("movieEuclids"); |

With the **movieEuclids** view in place, we can make predictions by querying for the top 20 movies with the smallest Euclidean Distance to a given movie. For instance, to find the 20 movies most similar to Toy Story (movieId = 1), we would execute the following SQL query:

|  |
| --- |
| spark.sql("select \* from movieEuclids where movieId1 = 1 order by euclidDist asc").show(20); |

The **ORDER BY** clause ensures that the results are sorted in ascending order of Euclidean Distance, with the most similar movies appearing first.

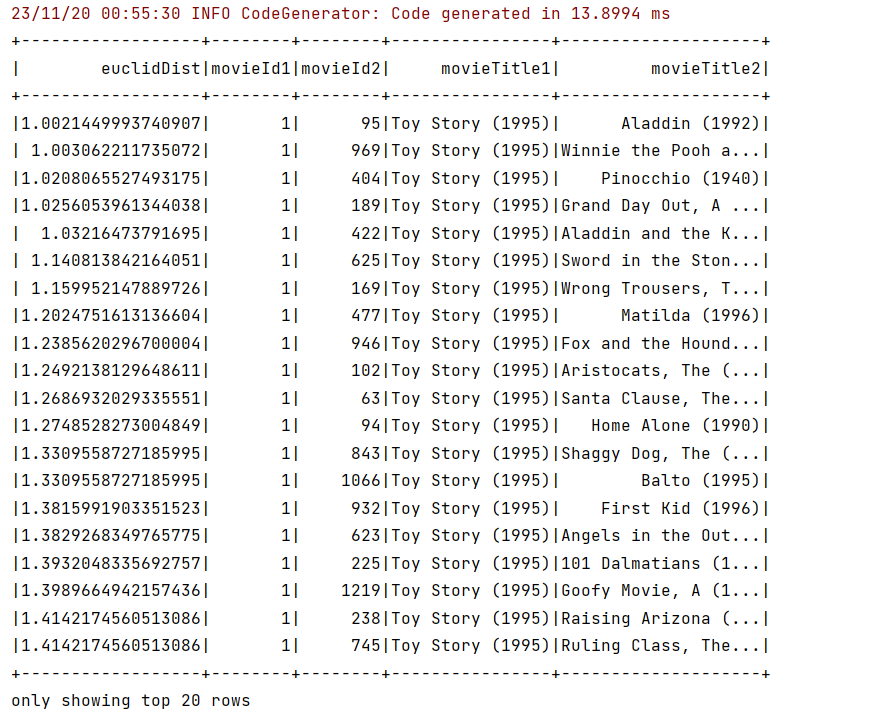


Figure 11 Output of the top 20 rows of the Euclidian Distance calculation on the dataset

* 1. **Collaborative Filtering and ALS Algorithm**

In the realm of recommendation systems, collaborative filtering stands out as a powerful technique that leverages user preferences and interactions to generate personalized recommendations. Unlike content-based recommendation systems, which focus on the attributes of items, collaborative filtering delves into the collective behavior of users, identifying patterns and similarities among them.

* + 1. **Collaborative Filtering**

At its core, collaborative filtering revolves around the idea that users with similar tastes are likely to enjoy similar items. By analyzing user interactions, such as ratings, purchases, and browsing history, collaborative filtering algorithms can identify these groups of similar users and recommend items that have been well-received by users within the same group.

1. **Types of Collaborative Filtering**

Collaborative filtering can be broadly categorized into two main approaches:

* **User-Based Collaborative Filtering**: This approach focuses on finding users with similar preferences and recommending items that have been rated highly by those similar users. The similarity between users is typically calculated using mathematical measures like Pearson correlation or cosine similarity.
* **Item-Based Collaborative Filtering:** Instead of focusing on users, item-based collaborative filtering identifies items with similar attributes or have been co-purchased or co-rated by users. This approach is particularly useful for recommending items that are not explicitly rated by the target user.

1. **Advantages of Collaborative Filtering**

Collaborative filtering offers several advantages over content-based recommendation systems:

* **Serendipity:** Collaborative filtering can recommend items that are not directly related to the items the user has previously interacted with, leading to unexpected and potentially interesting discoveries.
* **Cold Start Problem:** Unlike content-based systems that require item metadata, collaborative filtering can be effective even with new items or items with limited information.
* **Scalability:** Collaborative filtering algorithms can be efficiently implemented using distributed computing frameworks like Apache Spark, making them suitable for large-scale recommendation systems.

1. **Disadvantages of Collaborative Filtering**

Despite its strengths, collaborative filtering also has some limitations:

* **Data Sparsity:** Collaborative filtering performance can be affected by sparsity in the user-item interaction matrix, especially in cases where users have rated or interacted with only a small fraction of the available items.
* **Data Quality:** The quality of user interactions significantly impacts the accuracy of recommendations. Biased or inaccurate ratings can lead to misleading recommendations.
* **Understanding:** Collaborative filtering models can be difficult to interpret, making it challenging to understand why specific recommendations are made.

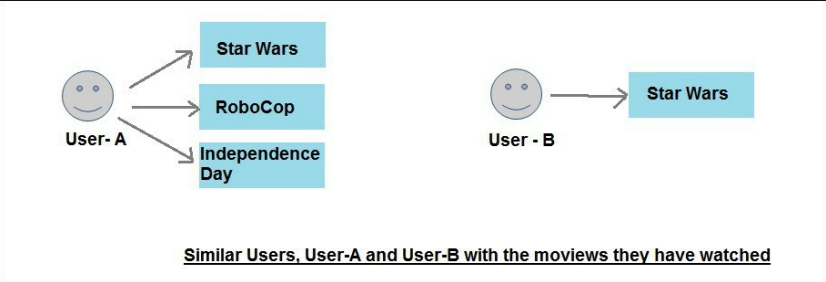


Figure 12 Collaborative Filtering approach diagram

* + 1. **Alternating least square – collaborative filtering**

Alternating Least Squares (ALS) is a collaborative filtering algorithm used for recommendation systems, and Apache Spark incorporates it as an inbuilt algorithm for collaborative filtering. The fundamental concept underlying ALS is matrix factorization, an algebraic approach to decompose a large matrix into smaller matrices, facilitating more manageable and efficient representation.

* + - 1. **Matrix Factorization**

Matrix factorization involves breaking down a large matrix into two smaller matrices that, when multiplied together, reconstruct the original matrix. In collaborative filtering, this is applied to user-item interaction matrices for recommendation systems. The goal is to predict missing entries in the matrix, representing user-item ratings.

Consider a matrix with users on the rows, items (e.g., movies) on the columns, and entries as user ratings: ​

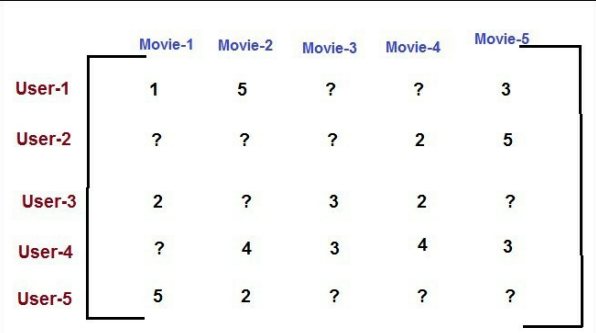


Figure 13 Example's data Structure

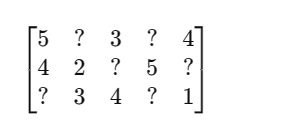


Figure 14 Example

Here, the goal is to predict the missing ratings and recommend items with high predicted ratings.

**Steps in ALS Algorithm**:

* Original Matrix:
* Obtain the user-item matrix where rows represent users, columns represent items, and entries represent ratings.
* Matrix Factorization:
  + Apply matrix factorization to break the large, sparse matrix into two smaller matrices: one for users (n×k) and another for items (m×k).
* These matrices are filled with values determined during the factorization process.

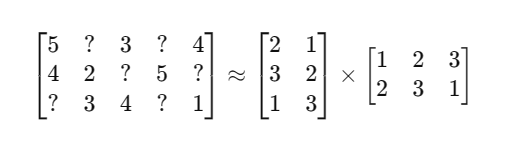


Figure 15 Factorization process

**Predicting Ratings:**

* To predict the rating for a user (u) at index i for an item (i), multiply the value at the u location in the user matrix and the value at the i location in the item matrix.
* This yields the predicted rating for the missing entries, aiding in recommendation.

**Collaborative Filtering Perspective:**

* ALS identifies user similarity based on their ratings for common items.
* For instance, if User-1 and User-4 have similar ratings for Movie-5 and Movie-2, they may share similar tastes. Recommendations can then be made based on the extra movies that one user has watched but the other hasn't.

We've explored how to use the matrix factorization approach to predict ratings for users and movies. But there's one crucial aspect we haven't delved into yet: figuring out the values for the factorized matrices.

Imagine that we've divided our large user-movie matrix into two smaller matrices: one for user factors and one for movie factors. Now, we need to determine the values in these smaller matrices.

One approach is to start by assigning high values to the user factor matrix. Since the maximum rating is 5, we can initialize the matrix with these values. For the number of factors (k), we can randomly choose a value, say 10.

Next, we keep this user factor matrix constant and focus on calculating the values in the movie factor matrix. We multiply these matrices and calculate the mean squared error (MSE), which indicates how closely the predicted ratings match the actual ratings.

Our goal is to find the values in the user factor matrix and movie factor matrix that minimize the MSE. To prevent overfitting, where the model performs well on training data but poorly on new data, we introduce regularization into the formula. Regularization adds a certain level of bias to prevent the model from becoming too sensitive to training data.

So, we alternate between keeping one matrix constant and calculating the values for the other, constantly minimizing the MSE. We continue this process until the change in MSE becomes negligible. These final values represent the optimal user factors and movie factors.

From a Spark perspective, these alternating matrix calculations can be efficiently performed in parallel across multiple nodes, making it a suitable choice for handling large datasets. Moreover, this algorithm produces accurate results even with sparse user-movie matrices.

**Reference:** Mathematical Implementation reference**[[1]](#endnote-1)**

* + - 1. **Implementation with Apache Spark**
* **Create SparkSession:** Initialize SparkSession with local master and application name "CollaborativeRecommendMovies".
* **Load Movie Data:** Read movie data from **"data/movie/u.item"** file into a **MovieVO JavaRDD**. Convert RDD to Dataset and register temporary view "movies".
* **Load Rating Data:** Read rating data from "data/movie/u.data" file into a **RatingVO JavaRDD**. Filter out rows with missing data. Convert RDD to Dataset and split into training and test datasets.



Figure 16 Attributes of the user's dataset

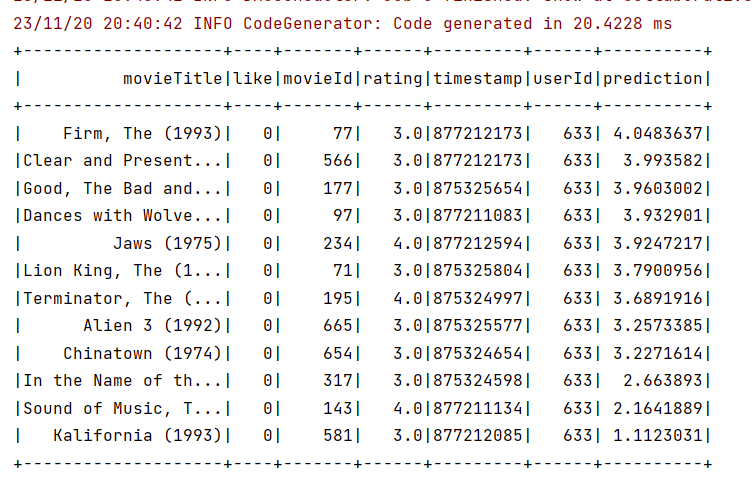


Figure 17 Movie’s Dataset

* **Build Recommendation Model**: Create an ALS model instance and set parameters: maximum iterations (10), regularization parameter (0.01), user column ("userId"), item column ("movieId"), rating column ("rating").
* **Fit Model on Training Data:** Train the ALS model on the training dataset.
* **Make Predictions on Test Data:** Transform the test dataset using the fitted model and store the predictions in a new Dataset.
* **Evaluate Model Performance:** Calculate the **root-mean-square error (RMSE**) between predicted and actual ratings using the **RegressionEvaluator** class.
* **Identify Recommended Movies:** Find movies with predicted ratings greater than 3 for a specific user using Spark SQL.

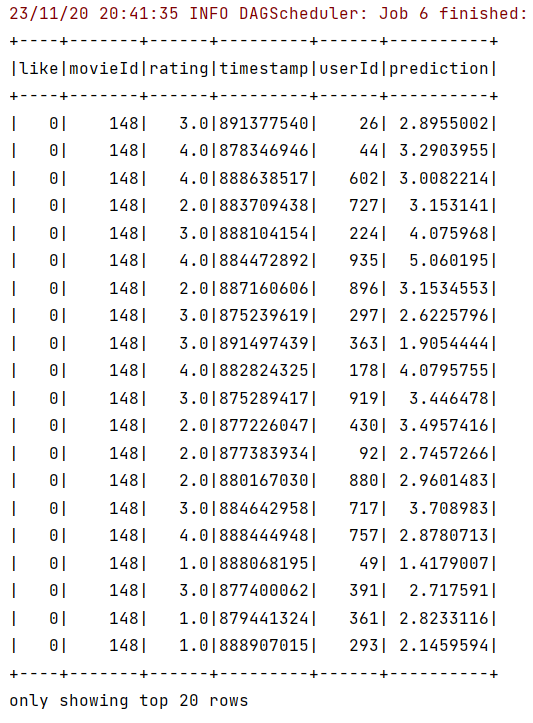


Figure 18 User's Data

The example provided for MovieLens dataset recommendations is based on explicit data where users give explicit ratings. In real life, users rarely rate content like YouTube videos after watching. To address this, implicit datasets with additional parameters like the number of views, wish list additions, DVD orders, etc., become crucial for predictions. Spark's ALS model supports implicit datasets, allowing users to specify this as a parameter.

1. **Conclusion**

Recommendation engines, both content-based and collaborative filtering, have revolutionized the way we interact with online platforms.

This paper provides a comprehensive guide to designing and implementing a recommendation system using Big Data technologies. The combination of content-based and collaborative filtering approaches, along with the utilization of Apache Spark, demonstrates the system's effectiveness in handling massive datasets and making accurate predictions. The insights provided aim to equip developers how to leverage recommendation systems to enhance user experience and drive engagement.

This paper on recommendation engines also encourages further exploration through web documentation and research papers, and emphasizes the dynamic and evolving nature of this field.

1. **References**

* (Mehta, 2017)
* [Movielens dataset collected by the GroupLens Research](https://grouplens.org/datasets/movielens/)
* [CME 323: Distributed Algorithms and Optimization, Spring 2015(PDF)](https://www.bing.com/ck/a?!&&p=22b17427c3509562JmltdHM9MTcwMDM1MjAwMCZpZ3VpZD0xYzE0OTBlOC0xYzA1LTYwNjEtMDgxMy04MjEwMWQ0ODYxOWYmaW5zaWQ9NTIzMw&ptn=3&ver=2&hsh=3&fclid=1c1490e8-1c05-6061-0813-82101d48619f&psq=alternating+least+squares+algorithm&u=a1aHR0cHM6Ly93ZWIuc3RhbmZvcmQuZWR1L35yZXphYi9jbGFzc2VzL2NtZTMyMy9TMTUvbm90ZXMvbGVjMTQucGRm&ntb=1)

1. [↑](#endnote-ref-1)