

# Movie Recommendation MLlib

Implementing a Collaborative Filtering Recommendation System Using GCP,  
Apache Spark, and Machine Learning

Niyat Habtom Seghid - 19967  
07/16/2025

Github Link:

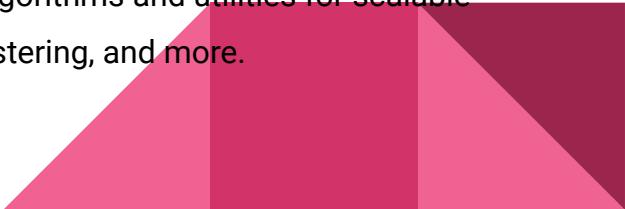
<https://github.com/niyat33/Cloud-Computing/blob/29f88f015d5f3991b53d2cc4004eef5ce21cb114/Machine%20Learning/Movie%20Recommendation%20System/ReadME.md>

# Contents:

1. Introduction
2. Design
3. Implementation
4. Test
5. Enhancement Ideas
6. Conclusion
7. Reference

---

# Introduction:

- In this project, we aim to **develop a movie recommendation system using Apache Spark's MLlib library**, specifically implementing collaborative filtering techniques. Collaborative filtering is a popular method used in recommendation systems, leveraging user-item interactions to predict user preferences.
  - The goal of this project is to build an effective movie recommendation system that can predict a user's rating for a movie they have not yet watched based on their previous ratings and the ratings given by other users with similar preferences.
  - **Apache Spark:** A powerful open-source processing engine built around speed, ease of use, and sophisticated analytics. It is designed for fast computation and supports multiple programming languages.
  - **Spark MLlib:** The machine learning library of Apache Spark, providing various algorithms and utilities for scalable machine learning, including collaborative filtering, classification, regression, clustering, and more.
- 

# Design

The project is divided into two primary steps:

## Step 1: Data Conversion

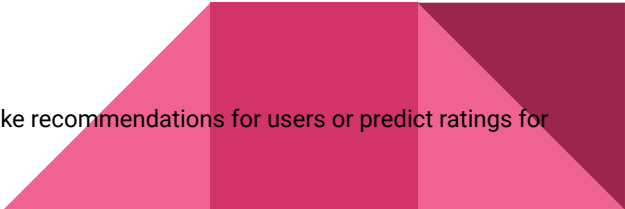
We begin by preprocessing the data from the MovieLens dataset, which initially contains four fields: UserID, MovieID, rating, and Timestamp. For our purposes, we only need the UserID, MovieID, and rating fields.

## Step 2: Implementing Collaborative Filtering using MLlib

After data preprocessing, we implement the collaborative filtering model using the Alternating Least Squares (ALS) algorithm provided by Spark's MLlib. The steps include:

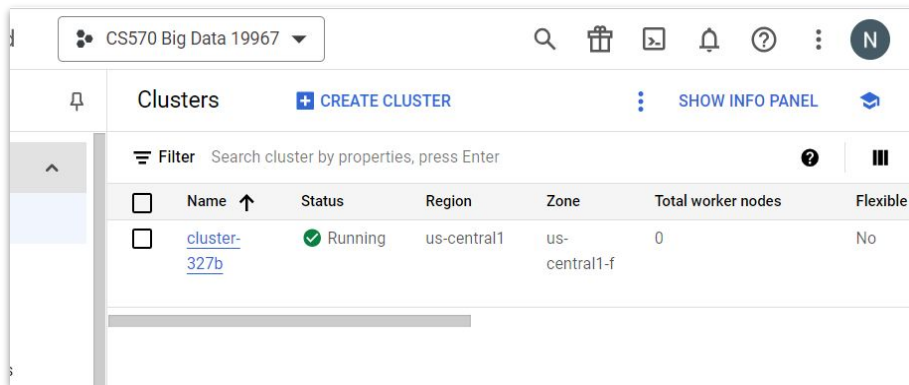
1. **Loading and Parsing Data:**
  - Load the converted data into an RDD (Resilient Distributed Dataset).
  - Parse the data to extract UserID, MovieID, and rating.
2. **Building the Recommendation Model:**
  - Train a matrix factorization model using the ALS algorithm with specified parameters such as rank and number of iterations.
3. **Model Evaluation:**
  - Evaluate the model's accuracy using Mean Squared Error (MSE) on the training data.
4. **Saving and Loading the Model:**
  - Save the trained model for future use.
  - Load the model when needed for generating recommendations.

After building and evaluating the collaborative filtering model using Spark's MLlib, you can use the trained model to make recommendations for users or predict ratings for specific user-movie pairs.



# Implementation

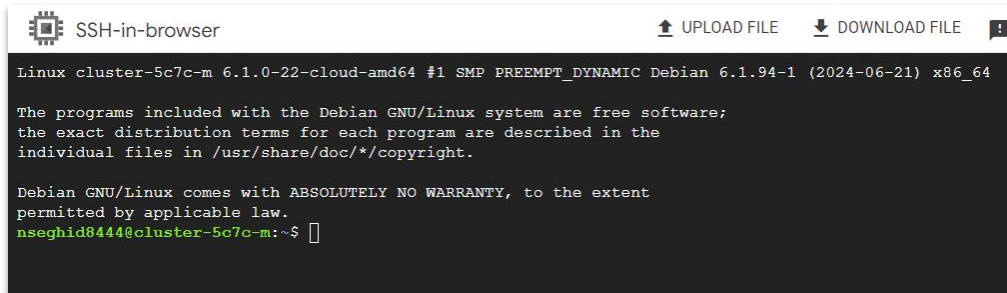
1. Set Up a Dataproc Cluster: Created and configured a Dataproc cluster to handle Spark jobs.



The screenshot shows the Google Cloud Platform Clusters page. At the top, there's a dropdown menu for 'CS570 Big Data 19967'. Below it, the 'Clusters' section has a '+ CREATE CLUSTER' button and a 'SHOW INFO PANEL' link. A filter bar allows searching by properties. The main table lists clusters with columns: Name, Status, Region, Zone, Total worker nodes, and Flexible. One cluster is listed with the name 'cluster-327b', status 'Running', region 'us-central1', zone 'us-central1-f', and 0 total worker nodes.

Name	Status	Region	Zone	Total worker nodes	Flexible
<a href="#">cluster-327b</a>	Running	us-central1	us-central1-f	0	No

2. Accessing the SSH of the Dataproc Cluster.



The screenshot shows an 'SSH-in-browser' terminal window. The title bar includes 'SSH-in-browser', 'UPLOAD FILE', and 'DOWNLOAD FILE' buttons. The terminal output shows the Linux command prompt for a cluster named 'cluster-5c7c-m' with kernel version '6.1.0-22-cloud-amd64'. It displays the Debian GNU/Linux system version '6.1.94-1' and architecture 'x86\_64'. A message states that the programs are free software with distribution terms in /usr/share/doc/\*/copyright. Another message states that Debian GNU/Linux comes with absolutely no warranty. The prompt is 'nseghid8444@cluster-5c7c-m:~\$'.

```
Linux cluster-5c7c-m 6.1.0-22-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.94-1 (2024-06-21) x86_64

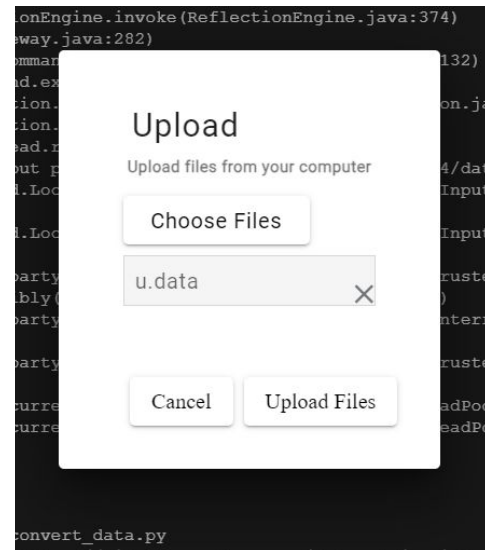
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
nseghid8444@cluster-5c7c-m:~$
```

# Implementation

3. Upload Source file u.data to the cluster for processing using any of the following two methods.

```
nseghid8444@cluster-5c7c-m:~$ wget https://files.grouplens.org/datasets/movielens/ml-100k/u.data
--2024-07-17 05:37:03-- https://files.grouplens.org/datasets/movielens/ml-100k/u.data
Resolving files.grouplens.org (files.grouplens.org)... 128.101.65.152
Connecting to files.grouplens.org (files.grouplens.org)|128.101.65.152|:443... [ ]
```



# Implementation

4. Create convert\_data.py to process the input data and convert it to the required format.

```
nseghid8444@cluster-5c7c-m:~$  
nseghid8444@cluster-5c7c-m:~$  
nseghid8444@cluster-5c7c-m:~$ vim convert_data.py
```



SSH-in-browser



UPLOAD FILE

```
input_file = 'u.data'  
output_file = 'u.data.csv'  
  
with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:  
    for line in infile:  
        userid, movieid, rating, _ = line.split()  
        outfile.write(f"{userid},{movieid},{rating}\n")  
  
print(f"Data converted and saved to {output_file}")  
~  
~
```

# Implementation

## 5. Create the Collaborative Filtering Script. recommendation.py

```
nseghid8444@cluster-5c7c-m:~$  
nseghid8444@cluster-5c7c-m:~$  
nseghid8444@cluster-5c7c-m:~$ vim recommendation.py
```

```
Collaborative Filtering Classification Example.  
  
from pyspark import SparkContext  
  
# $example on$  
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating  
# $example off$  
  
if __name__ == "__main__":  
    sc = SparkContext(appName="PythonCollaborativeFilteringExample")  
    # $example on$  
    # Load and parse the data  
    # - Each row consists of a user, a product and a rating.  
    data = sc.textFile("u.data.csv")  
  
    # Each line is  
    ratings = data.map(lambda l: l.split(',')\n                        .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2]))))  
  
    # Build the recommendation model using ALS  
    # - rank: number of features to use  
    rank = 10  
  
    # - iterations: number of iterations of ALS (recommended: 10-20)  
    numIterations = 10  
  
    # The default ALS.train() method which assumes ratings are explicit.  
    # - Train a matrix factorization model given an RDD of ratings given by  
    #   users to some products, in the form of (userID, productID, rating) pairs.  
    # - We approximate the ratings matrix as the product of two lower-rank  
    #   matrices of a given rank (number of features).  
    #   + To solve for these features, we run a given number of  
    #     iterations of ALS.  
    #   + The level of parallelism is determined automatically based  
    #     on the number of partitions in ratings.  
    model = ALS.train(ratings, rank, numIterations)  
  
    #####  
    # Evaluate the model on training data  
    # - Evaluate the recommendation model on rating training data  
    #####  
    testdata = ratings.map(lambda p: (p[0], p[1]))  
-- INSERT --
```



# Implementation

6. **Uploading file to HDFS:** The purpose of these commands is to ensure that the `u.data.csv` file is available in HDFS for further processing by your Spark job. You need this file to be in HDFS because Spark reads data from HDFS to perform distributed processing.

```
nseghid8444@cluster-5c7c-m:~$ hdfs dfs -put /home/nseghid8444/u.data.csv /user/nseghid8444/
nseghid8444@cluster-5c7c-m:~$ hdfs dfs -ls /user/nseghid8444/
Found 2 items
drwxr-xr-x   - nseghid8444  hadoop           0 2024-07-17 05:56 /user/nseghid8444/.sparkStaging
-rw-r--r--   1 nseghid8444  hadoop    979173 2024-07-17 05:58 /user/nseghid8444/u.data.csv
nseghid8444@cluster-5c7c-m:~$
```

7. Submit Spark Job: proceed to run your Spark job using the `spark-submit` command.



# Implementation

7. Submit Spark Job: proceed to run your Spark job using the `spark-submit` command.

```
nseghid8444@cluster-5c7c-m:~$ spark-submit recommendation.py
24/07/17 05:58:33 INFO SparkEnv: Registering MapOutputTracker
24/07/17 05:58:33 INFO SparkEnv: Registering BlockManagerMaster
24/07/17 05:58:33 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
24/07/17 05:58:33 INFO SparkEnv: Registering OutputCommitCoordinator
24/07/17 05:58:34 INFO DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at cluster-5c7c-m.us-east1-c.c.cs570-big-data-19967.internal./10.142.0.5:80
24/07/17 05:58:34 INFO AHSProxy: Connecting to Application History server at cluster-5c7c-m.us-east1-c.c.cs570-big-data-19967.internal./10.142.0.5:10200
24/07/17 05:58:35 INFO Configuration: resource-types.xml not found
24/07/17 05:58:35 INFO ResourceUtils: Unable to find 'resource-types.xml'.
24/07/17 05:58:36 INFO YarnClientImpl: Submitted application application_1721193856183_0004
24/07/17 05:58:37 INFO DefaultNoHARMAFailoverProxyProvider: Connecting to ResourceManager at cluster-5c7c-m.us-east1-c.c.cs570-big-data-19967.internal./10.142.0.5:80
24/07/17 05:58:38 INFO MetricsConfig: Loaded properties from hadoop-metrics2.properties
24/07/17 05:58:38 INFO MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
24/07/17 05:58:38 INFO MetricsSystemImpl: google-hadoop-file-system metrics system started
24/07/17 05:58:39 INFO GhfsGlobalStorageStatistics: Detected potential high latency for operation op_get_file_status. latencyMs=844; previousMaxLatencyMs=0; operati
proc-temp-us-east1-659426342353-9gnrwiur/8a988e1c-4aba-4706-8043-eceb65a56a6/spark-job-history
24/07/17 05:58:40 INFO GoogleCloudStorageImpl: Ignoring exception of type GoogleJsonResponseException; verified object already exists with desired state.
24/07/17 05:58:40 INFO GoogleHadoopOutputStream: hflush(): No-op due to rate limit (RateLimiter[stableRate=0.2qps]): readers will *not* yet see flushed data for gs:
426342353-9gnrwiur/8a988e1c-4aba-4706-8043-eceb65a56a6/spark-job-history/application_1721193856183_0004.inprogress [CONTEXT ratelimit_period="1 MINUTES" ]
24/07/17 05:58:42 INFO FileInputFormat: Total input files to process : 1
Mean Squared Error = 0.4839300310168524
24/07/17 05:59:25 INFO FileInputFormat: Total input files to process : 1
24/07/17 05:59:25 INFO FileInputFormat: Total input files to process : 1
24/07/17 05:59:27 WARN MatrixFactorizationModel: User factor does not have a partitioner. Prediction on individual records could be slow.
24/07/17 05:59:27 WARN MatrixFactorizationModel: User factor is not cached. Prediction could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModel: Product factor does not have a partitioner. Prediction on individual records could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModel: Product factor is not cached. Prediction could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModelWrapper: User factor does not have a partitioner. Prediction on individual records could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModelWrapper: User factor is not cached. Prediction could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModelWrapper: Product factor does not have a partitioner. Prediction on individual records could be slow.
24/07/17 05:59:28 WARN MatrixFactorizationModelWrapper: Product factor is not cached. Prediction could be slow.
nseghid8444@cluster-5c7c-m:~$
```

# Test

8. Evaluate the performance of the recommendation system.

```
24/07/17 05:58:42 INFO FileInputFormat: Total input files to process : 1  
Mean Squared Error = 0.4839300310168524  
24/07/17 05:59:25 INFO FileInputFormat: Total input files to process : 1
```

Results:

Test Results: Mean Squared Error (MSE): 0.4839



# Enhancement Ideas

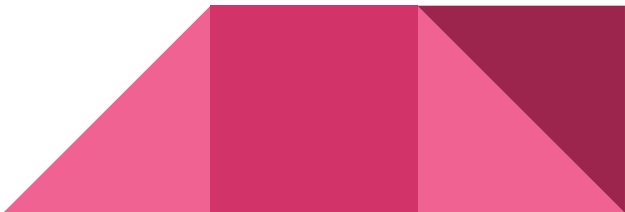
## Hybrid Recommendation System:

- Combine collaborative filtering with content-based filtering.
- Implement a metadata-based approach using movie metadata.

## Model Optimization and Tuning:

- Experiment with different hyperparameters for the ALS algorithm.
- Apply cross-validation for robustness and generalizability.

## Scalability and Performance Improvements:

- Optimize performance using Spark's capabilities (e.g., data partitioning, caching).
  - Explore advanced matrix factorization techniques or deep learning-based models.
- 

# Conclusion

This project successfully demonstrates the implementation of a movie recommendation system using collaborative filtering techniques with Apache Spark's MLlib. By preprocessing the MovieLens dataset and applying the Alternating Least Squares (ALS) algorithm, we were able to build and evaluate a recommendation model that predicts user ratings for movies.

## Key Achievements

- Data Preprocessing: Efficiently converted raw MovieLens data into the required format for collaborative filtering.
- Model Training: Trained a matrix factorization model using the ALS algorithm to capture latent factors in user-item interactions.
- Model Evaluation: Evaluated the model using Mean Squared Error (MSE) to ensure its accuracy and effectiveness.
- Recommendations and Predictions: Implemented functionalities to recommend movies for users and predict ratings for specific user-movie pairs.



# References:

[https://hc.labnet.sfbu.edu/~henry/npu/classes/mllib/collaborative\\_filtering/slide/Examples.html](https://hc.labnet.sfbu.edu/~henry/npu/classes/mllib/collaborative_filtering/slide/Examples.html)

[https://hc.labnet.sfbu.edu/~henry/npu/classes/mllib/collaborative\\_filtering/slide/exercise\\_collaborative\\_filtering.html](https://hc.labnet.sfbu.edu/~henry/npu/classes/mllib/collaborative_filtering/slide/exercise_collaborative_filtering.html)

