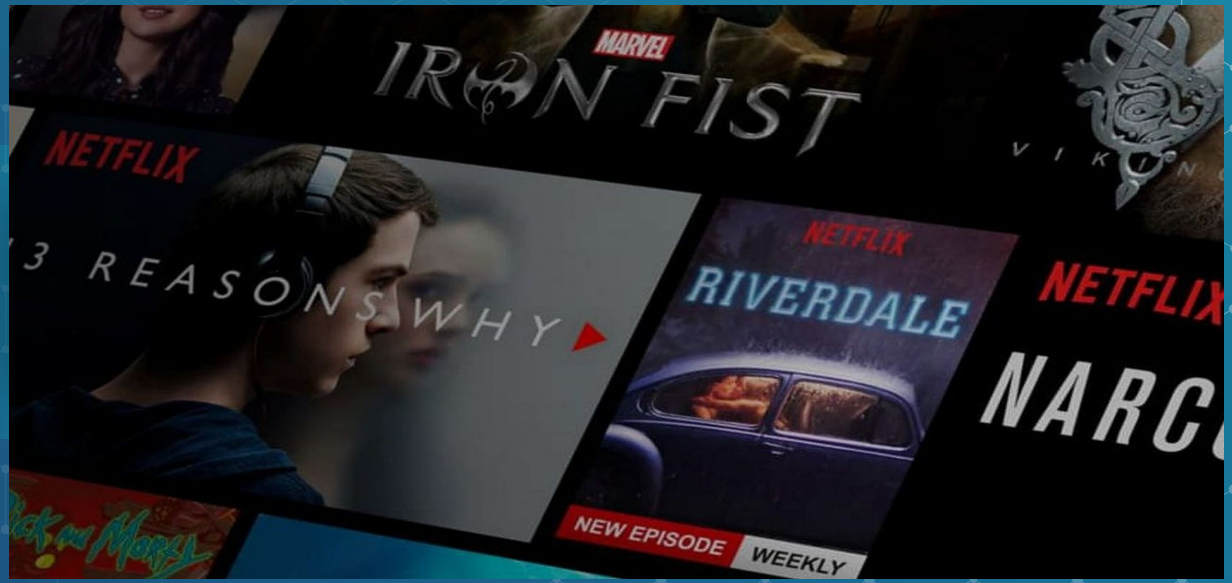
Movie Recommendation with MLlib - Collaborative Filtering



Belsabel Woldemichael

Contents

Introduction

Project Context

- **Introduction**: Explore the application of machine learning in the entertainment industry, focusing on recommending movies to users based on their preferences.
- **Significance**: Highlight the importance of personalized recommendations in improving user experience and engagement on movie platforms.

Objective of the Project

- **Model Development**: Create a machine learning model that accurately predicts user preferences and recommends movies using Collaborative Filtering techniques.
- **Technology Utilization**: Utilize Apache Spark's MLlib to efficiently process large-scale movie rating data from the MovieLens dataset.

Goals

- **Implementation and Evaluation**: Develop and assess a Collaborative Filtering model using the Alternating Least Squares (ALS) algorithm within MLlib.

Design

Description of the MovieLens Dataset

- **Origin and Composition**: The MovieLens dataset, a popular resource in recommender systems research, was compiled by the GroupLens research group at the University of Minnesota. It includes millions of movie ratings provided by users, along with user demographic details and movie metadata.
- **Data Structure**: The dataset typically comprises user IDs, movie IDs, ratings, and timestamps. Mention any specific subset or version used, such as MovieLens 100k or 1M.
- **Importance of Data**: The dataset's diversity and volume make it ideal for training robust machine learning models for recommendation systems.

Relevance of the MovieLens Dataset

- **Benchmarking Standard**: The MovieLens dataset serves as a benchmark for evaluating the effectiveness of various recommendation algorithms, enabling comparisons across different methods and studies.
- **Real-World Application**: Using a real-world dataset like MovieLens aids in understanding user behavior patterns and preferences, crucial for developing systems suitable for commercial deployment.

Design Approach Using Collaborative Filtering in MLlib

- **Algorithm Choice**: Employ Collaborative Filtering, specifically the Alternating Least Squares (ALS) algorithm.
- **Model Framework**: Discuss the integration of the ALS algorithm within Apache Spark's MLlib.
- **Scalability and Performance**: Emphasize the scalability of the Spark platform, which facilitates faster computations and model training over a distributed computing environment, essential for real-time recommendation systems.

Implementation and Test

1. Project: Movie Recommendation with MLlib - Collaborative Filtering (implementaiton)

1. Create the Shell Script 1. Create a new shell script file using a text editor, such as nano or vim.

```
belsabelwoldemichael@cloudshell:~ (cs570-big-data-429720)$ nano convert_data.sh
```

2. Copy and paste one of the following scripts into the text editor.

```
cat u.data | while read userid movieid rating timestamp
do
    echo "${userid},${movieid},${rating}"
done
```

3. Save the file and exit the text editor (Ctrl+X, then Y, then Enter for nano).

```
belsabelwoldemichael@cloudshell:~ (cs570-big-data-429720)$ chmod +x convert_data.sh
```

Prepare the Input Data Upload u.data file using the Cloud Shell file upload feature or download it using wget or curl.

```
belsabelwoldemichael@cloudshell:~ (cs570-big-data-429720)$ wget https://files.grouplens.org/datasets/movielens/ml-100k/u.data
--2024-07-17 20:22:00--  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... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1979173 (1.9M)
Saving to: 'u.data'

u.data                                              100%[================================================================================================>]

2024-07-17 20:22:01 (5.30 MB/s) - 'u.data' saved [1979173/1979173]
```

Run the Script

1. Run the script and save the output to a file:

```
belsabelwoldemichael@cloudshell:~ (cs570-big-data-429720)$ ./convert_data.sh > Edited_data.csv
```

2. Cat formatted_data.csv

```
764,596,3
537,443,3
618,628,2
487,291,3
113,975,5
943,391,2
864,685,4
750,323,3
279,64,1
646,750,3
654,370,2
617,582,4
913,690,3
660,229,2
421,498,4
495,1091,4
806,421,4
676,538,4
721,262,3
913,209,2
378,78,3
880,476,3
716,204,5
276,1090,1
13,225,2
12,203,3
```

2. Implement this version of MLlib - Collaborative Filtering Examples Python Latest code: recommendation_example.py (complete code)

Download the test.data File

1. Open Cloud Shell.
2. Use the wget command to download the filed

```
belsabelwoldemichael@cloudshell:~ (cs570-big-data-429720)$ wget https://raw.githubusercontent.com/apache/spark/master/data/mllib/als/test.data
--2024-07-17 20:26:13--  https://raw.githubusercontent.com/apache/spark/master/data/mllib/als/test.data
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 128 [text/plain]
Saving to: 'test.data'

test.data                                   100%[=========================================================================================================>]

2024-07-17 20:26:14 (4.50 MB/s) - 'test.data' saved [128/128]
```

Write the recommendation_example.py
Script 1. Create and edit the Python script

```
"""
Collaborative Filtering Classification Example.
"""
from pyspark import SparkContext, SparkConf
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

if __name__ == "__main__":
    # Use local mode
    conf = SparkConf().setMaster("local[*]").setAppName("PythonCollaborativeFilteringExample")
    sc = SparkContext(conf=conf)

    # Load and parse the data
    data = sc.textFile("file:///home/faraya85431/test.data")  # Use absolute path
    ratings = data.map(lambda l: l.split(','))\
                  .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

    # Build the recommendation model using ALS
    rank = 10
    numIterations = 10
    model = ALS.train(ratings, rank, numIterations)

    # Evaluate the model on training data
    testdata = ratings.map(lambda p: (p[0], p[1]))
    predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
    ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
    MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1])**2).mean()
    print("Mean Squared Error = " + str(MSE))

    # Save and load model
    model.save(sc, "file:///home/faraya85431/target/tmp/myCollaborativeFilter")
    sameModel = MatrixFactorizationModel.load(sc, "file:///home/faraya85431/target/tmp/myCollaborativeFilter")
```

Run the Script Execute the script using spark-submit:

```
24/07/17 21:08:02 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
24/07/17 21:08:03 INFO PythonRunner: Times: total = 781, boot = -864, init = 988, finish = 657
24/07/17 21:08:03 INFO Executor: Finished task 1.0 in stage 178.0 (TID 41). 2281 bytes result sent to driver
24/07/17 21:08:03 INFO TaskSetManager: Finished task 1.0 in stage 178.0 (TID 41) in 801 ms on cs-312169370680-default (executor driver)
24/07/17 21:08:03 INFO TaskSchedulerImpl: Removed TaskSet 178.0, whose tasks have all completed, from pool
24/07/17 21:08:03 INFO DAGScheduler: ResultStage 178 (mean at /home/belsabelwoldemichael/recommendation.py:20) finished in 1.653 s
24/07/17 21:08:03 INFO DAGScheduler: Job 13 is finished. Cancelling potential speculative or zombie tasks for this job
24/07/17 21:08:03 INFO TaskSchedulerImpl: Killing all running tasks in stage 178: Stage finished
24/07/17 21:08:03 INFO DAGScheduler: Job 13 finished: mean at /home/belsabelwoldemichael/recommendation.py:20, took 5.926181 s
Mean Squared Error = 0.4849108149757002
24/07/17 21:08:03 INFO SparkContext: Invoking stop() from shutdown hook
24/07/17 21:08:03 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/07/17 21:08:03 INFO SparkUI: Stopped Spark web UI at http://cs-312169370680-default:4040
24/07/17 21:08:03 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/07/17 21:08:03 INFO MemoryStore: MemoryStore cleared
24/07/17 21:08:03 INFO BlockManager: BlockManager stopped
24/07/17 21:08:03 INFO BlockManagerMaster: BlockManagerMaster stopped
24/07/17 21:08:03 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/07/17 21:08:03 INFO SparkContext: Successfully stopped SparkContext
24/07/17 21:08:03 INFO ShutdownHookManager: Shutdown hook called
```

The Mean Squared Error is 0.48491081…

Enhancement Ideas

Hybrid Recommendation System:

- Combine collaborative and content-based filtering for better accuracy.
- Use contextual information like time of day or user mood.

Advanced Machine Learning Techniques:

- Implement deep learning models for complex interactions.
- Utilize graph-based algorithms for better relationship insights.

Real-Time Recommendations:

- Integrate streaming data for real-time updates.
- Adjust recommendations dynamically based on user actions.

Conclusion

- Developed a robust movie recommendation system using the MovieLens dataset and Apache Spark’s MLlib.
- Employed the ALS algorithm for effective collaborative filtering, handling large-scale, sparse datasets.
- Ensured accurate and scalable recommendations to enhance user experience and engagement.

References

Movie Recommendation with Collaborative Filtering in Pyspark

Sample code

Github Link