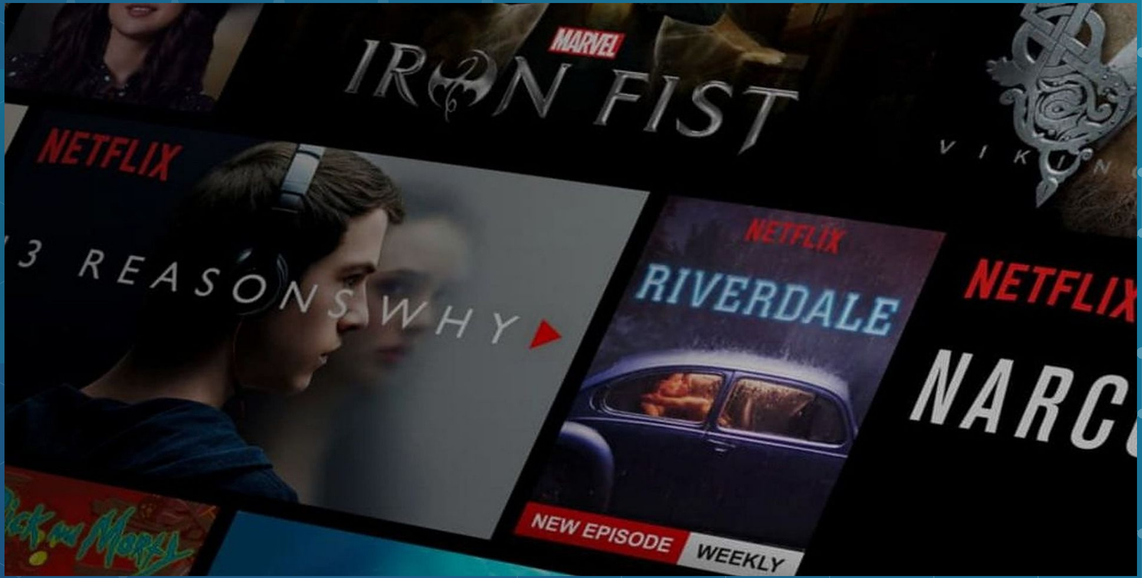
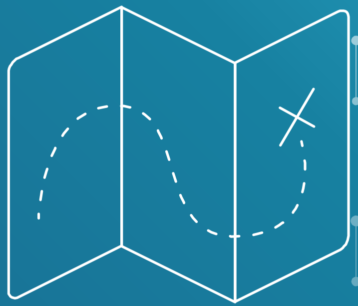
Movie Recommendation with MLlib - Collaborative Filtering



Belsabel Woldemichael

Contents

- Introduction
- Design
- Implementation
- Test
- Enhancement Ideas
- Conclusion



Introduction

Project Overview

Create a movie recommendation system using collaborative filtering with MLlib on Apache Spark.

Goals

- Build a personalized movie recommendation system tailored to user preferences.
- Showcase the application of collaborative filtering for large-scale datasets.

Technologies

- Apache Spark's MLlib for machine learning.
- Google Cloud Platform (GCP) for data storage and processing.

Design

Approach

We started by studying PySpark's ALS algorithm and its use in collaborative filtering, including a thorough review of relevant documentation and tutorials.

Problem Identification

The primary challenge was to develop an efficient and scalable recommendation system capable of handling large datasets.

Solution Investigation

We evaluated various recommendation techniques:

- **Collaborative Filtering**: Leverages user-item interactions.
- **Content-Based Filtering**: Relies on item features.
- **Hybrid Methods**: Integrates both approaches.

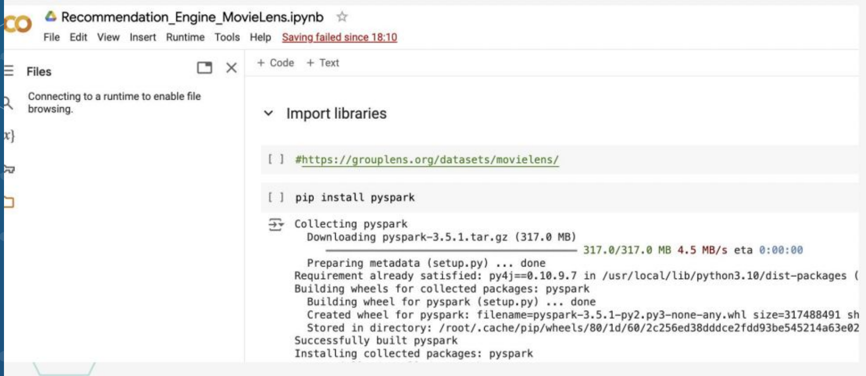
Theoretical Comparison

Collaborative Filtering, particularly with the ALS algorithm, was chosen for its scalability and excellent performance with large, sparse datasets.

Implementation and Test

Step 3.1: Download the Pyspark code (ipynb)

Step 3.2: Upload the ipynb file to your Colab



- Step 3.3: Experiment Pyspark code (ipynb) by modifying the ipynb file

Change number of folds to 3

```
# Build cross validation using CrossValidator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=3)

# Confirm cv was built
print(cv)
```

- Step 3.4: Save the modified ipynb file as py format

- Step 3.6: Save the modified ipynb file as HTML format which can be used on Step 5 of this project

Packages

```
import pandas as pd
from pyspark.sql.functions import col, explode
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
```

```
import pandas as pd
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
```

Initiate Spark Session

```
from pyspark.sql import SparkSession
sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Recommendations').getOrCreate()
```

Load Data

```
movies = spark.read.csv("/content/movies.csv",header=True)
ratings = spark.read.csv("/content/ratings.csv",header=True)
```

Display and Schema of Ratings Data

```
ratings.show()

+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
|     1|      1|   4.0|964982703|
|     1|      3|   4.0|964981247|
|     1|      6|   4.0|964982224|
|     1|     47|   5.0|964983815|
|     1|     50|   5.0|964982931|
|     1|     70|   3.0|964982400|
|     1|    101|   5.0|964980868|
|     1|    110|   4.0|964982176|
|     1|    151|   5.0|964984041|
|     1|    157|   5.0|964984100|
|     1|    163|   5.0|964983650|
|     1|    216|   5.0|964981208|
|     1|    223|   3.0|964980985|
|     1|    231|   5.0|964981179|
|     1|    235|   4.0|964980908|
|     1|    260|   5.0|964981680|
|     1|    296|   3.0|964982967|
|     1|    316|   3.0|964982310|
|     1|    333|   5.0|964981179|
|     1|    349|   4.0|964982563|
+------+-------+------+---------+
only showing top 20 rows
```

Data Preprocessing

```
from pyspark.sql import SparkSession
sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Recommendations').getOrCreate()
```

Calculate Sparsity

```
# Count the total number of ratings in the dataset
numerator = ratings.select("rating").count()

# Count the number of distinct userIds and distinct movieIds
num_users = ratings.select("userId").distinct().count()
num_movies = ratings.select("movieId").distinct().count()

# Set the denominator equal to the number of users multiplied by the number of movies
denominator = num_users * num_movies

# Divide the numerator by the denominator
sparsity = (1.0 - (numerator *1.0)/denominator)*100
print("The ratings dataframe is ", "%.2f" % sparsity + "% empty.")
```

```
The ratings dataframe is  98.30% empty.
```

Interpret Ratings

userId	count
414	2698
599	2478
474	2108
448	1864
274	1346
610	1302
68	1260
380	1218
606	1115
288	1055
249	1046
387	1027
182	977
307	975
603	943
298	939
177	904
318	879
232	862
480	836

only showing top 20 rows

movieId	count
356	329
318	317
296	307
593	279
2571	278
260	251
480	238
110	237
589	224
527	220
2959	218
1	215
1196	211
50	204
2858	204
47	203
780	202
150	201
1198	200
4993	198

only showing top 20 rows

Build ALS Model

```python
# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 1234)

# Create ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", nonnegative = True, implicitPrefs = False, coldStartStrategy="drop")

# Confirm that a model called "als" was created
type(als)
```

```
pyspark.ml.recommendation.ALS
def __init__(*, rank: int=10, maxIter: int=10, regParam: float=0.1, numUserBlocks: int=10,
numItemBlocks: int=10, implicitPrefs: bool=False, alpha: float=1.0, userCol: str='user',
itemCol: str='item', seed: Optional[int]=None, ratingCol: str='rating', nonnegative:
bool=False, checkpointInterval: int=10, intermediateStorageLevel: str='MEMORY_AND_DISK',
finalStorageLevel: str='MEMORY_AND_DISK', coldStartStrategy: str='nan', blockSize: int=4096)
```

```
>>> item_subset_recs.select("recommendations.user", "recommendations.rating").first()
Row(user=[0, 1, 2], rating=[3.910..., 3.473..., -0.899...])
>>> als_path = temp_path + "/als"
>>> als.save(als_path)
>>> als2 = ALS.load(als_path)
>>> als.getMaxIter()
```

Tune ALS Model

```
# Import the requisite items
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100, 150]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()
            #             .addGrid(als.maxIter, [5, 50, 100, 200]) \


# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))
```

```
Num models to be tested:  16
```

Cross-Validation

```
# Build cross validation using CrossValidator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=5)

# Confirm cv was built
print(cv)
```

```
CrossValidator_d27636957c4c
```

Train and Evaluate Model

```
# Print best_model
print(type(best_model))

# Complete the code below to extract the ALS model parameters
print("**Best Model**")

# # Print "Rank"
print("  Rank:", best_model._java_obj.parent().getRank())

# Print "MaxIter"
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())

# Print "RegParam"
print("  RegParam:", best_model._java_obj.parent().getRegParam())
```

```
<class 'pyspark.ml.recommendation.ALSModel'>
**Best Model**
  Rank:  150
  MaxIter:  10
  RegParam:  0.15
```

Make Predictions

```
test_predictions.show()
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|   148|    356|   4.0| 3.4951332|
|   148|   4896|   4.0| 3.4835334|
|   148|   4993|   3.0|  3.465551|
|   148|   7153|   3.0| 3.4216132|
|   148|   8368|   4.0|  3.591083|
|   148|  40629|   5.0| 3.2217665|
|   148|  50872|   3.0| 3.6663907|
|   148|  60069|   4.5|  3.695917|
|   148|  69757|   3.5| 3.3879697|
|   148|  72998|   4.0| 3.2131975|
|   148|  81847|   4.5| 3.4920812|
|   148|  98491|   5.0| 3.7356784|
|   148| 115617|   3.5| 3.5717542|
|   148| 122886|   3.5| 3.4257748|
|   463|    296|   4.0|  4.149282|
|   463|    527|   4.0| 3.7739785|
|   463|   2019|   4.0| 3.9446247|
|   471|    527|   4.5|  3.773583|
|   471|   6016|   4.0| 3.9766822|
|   471|   6333|   2.5| 3.2052839|
+------+-------+------+----------+
only showing top 20 rows
```

Generate Recommendations

```
# Generate n Recommendations for all users
nrecommendations = best_model.recommendForAllUsers(10)
nrecommendations.limit(10).show()
```

```
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|     1|[{3379, 5.7130847...|
|     2|[{131724, 4.79666...|
|     3|[{6835, 4.8578787...|
|     4|[{3851, 4.8525457...|
|     5|[{3379, 4.5449133...|
|     6|[{33649, 4.725941...|
|     7|[{33649, 4.459244...|
|     8|[{3379, 4.635308}...|
|     9|[{3379, 4.7842216...|
|    10|[{71579, 4.533425...|
+------+--------------------+
```

Merge with Movies Data for Interpretability

```
nrecommendations.join(movies, on='movieId').filter('userId = 100').show()
```

```
+-------+------+---------+--------------------+--------------------+
|movieId|userId|   rating|               title|              genres|
+-------+------+---------+--------------------+--------------------+
|  67618|   100|5.0828342|Strictly Sexual (...|Comedy|Drama|Romance|
|   3379|   100| 5.015384| On the Beach (1959)|               Drama|
|  33649|   100|5.0150394|  Saving Face (2004)|Comedy|Drama|Romance|
|  42730|   100|4.9038916|   Glory Road (2006)|               Drama|
|  74282|   100|4.8903875|Anne of Green Gab...|Children|Drama|Ro...|
| 184245|   100|4.8847737|De platte jungle ...|         Documentary|
| 179135|   100|4.8847737|Blue Planet II (2...|         Documentary|
| 138966|   100|4.8847737|Nasu: Summer in A...|           Animation|
| 117531|   100|4.8847737|    Watermark (2014)|         Documentary|
|  86237|   100|4.8847737|  Connections (1978)|         Documentary|
+-------+------+---------+--------------------+--------------------+
```

```
ratings.join(movies, on='movieId').filter('userId = 100').sort('rating', ascending=False).limit(10).show()
```

```
+-------+------+------+--------------------+--------------------+
|movieId|userId|rating|               title|              genres|
+-------+------+------+--------------------+--------------------+
|   1101|   100|   5.0|      Top Gun (1986)|      Action|Romance|
|   1958|   100|   5.0|Terms of Endearme...|        Comedy|Drama|
|   2423|   100|   5.0|Christmas Vacatio...|              Comedy|
|   4041|   100|   5.0|Officer and a Gen...|       Drama|Romance|
|   5620|   100|   5.0|Sweet Home Alabam...|      Comedy|Romance|
|    368|   100|   4.5|     Maverick (1994)|Adventure|Comedy|...|
|    934|   100|   4.5|Father of the Bri...|              Comedy|
|    539|   100|   4.5|Sleepless in Seat...|Comedy|Drama|Romance|
|     16|   100|   4.5|       Casino (1995)|         Crime|Drama|
|    553|   100|   4.5|    Tombstone (1993)|Action|Drama|Western|
+-------+------+------+--------------------+--------------------+
```

Open GCP and upload your the recommendation_Engine_MovieLens.py file

```
# -*- coding: utf-8 -*-
"""Recommendation_Engine_MovieLens.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1wNSzqsOwDDH6bXQ-I-hC4Zc1nb0SD0WP

### Import libraries
"""

#https://grouplens.org/datasets/movielens/

# pip install pyspark

# pip install spark


import pandas as pd
from pyspark.sql.functions import col, explode
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

"""### Initiate spark session"""

from pyspark.sql import SparkSession
sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Recommendations').getOrCreate()

"""# 1. Load data"""

movies = spark.read.csv("file:///home/faraya85431/movies.csv",header=True)
ratings = spark.read.csv("file:///home/faraya85431/ratings.csv",header=True)

ratings.show()

ratings.printSchema()
```

Result

```
belsabelwoldemichael@cs-312169370680-default:~$ spark submit recommendation_engine_movielens.py
```

```
**Best Model**
  Rank: 50
  MaxIter: 10
  RegParam: 0.15
RMSE: 0.8685666272031626
+------+-------+---------+
|userId|movieId|   rating|
+------+-------+---------+
|     1|   3379|5.7632384|
|     1|  33649| 5.598928|
|     1|   5490|5.5296617|
|     1| 171495| 5.416649|
|     1|   5416|5.4002886|
|     1|   3951|5.4002886|
|     1|   5328|5.4002886|
|     1| 131724| 5.363606|
|     1|   5915|5.3629937|
|     1| 177593| 5.356516|
+------+-------+---------+
```

Enhancement Ideas

Hybrid Recommendation Approach:

- **Combine Filtering Methods**: Integrate collaborative filtering with content-based filtering to improve accuracy and handle cold start problems.
- **Contextual Recommendations**: Incorporate contextual information like time of day or location to provide more relevant suggestions.

Advanced Machine Learning Models:

- **Deep Learning**: Use neural networks to capture complex user-item interactions.
- **Graph-Based Algorithms**: Leverage graph-based techniques for better relationship mapping and recommendations.

Real-Time Data Processing:

- **Streaming Data**: Implement Apache Kafka or Spark Streaming to process real-time user activity and update recommendations instantly.
- **Dynamic Personalization**: Adjust recommendations on-the-fly based on recent user interactions.

Conclusion

- Implemented a robust movie recommendation system using collaborative filtering with Apache Spark’s MLlib.
- Leveraged large-scale data processing capabilities to deliver personalized recommendations based on user preferences.
- Demonstrated the effectiveness of collaborative filtering for enhancing user engagement on movie platforms.
- Future enhancements include integrating hybrid recommendation methods and implementing real-time data processing for dynamic updates.

References

Sample code

Movie Recommendation with Collaborative Filtering in Pyspark

Github Link