Project: Movie Recommendation with MLlib

Emily Weng 20016
CS570 Big Data Project

Introduction

1. This was done on Google Colab for me since it was easier and more interactive.
2. Project is split in two parts, converting data and implementing

Step 1: Convert MovieLens' data (UserID, MovieID, rating, Timestamp)

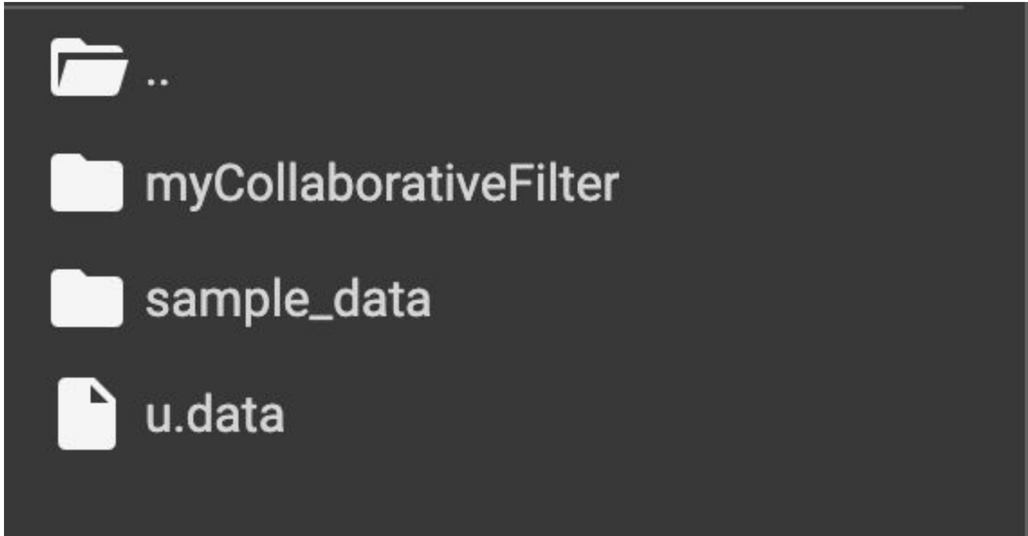
1. Install Pyspark first

```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 4.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=3978e49f4bafee688131a122994c28b96ece1988569057c0ec9890be3be68
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

Step 1: Convert MovieLens' data (UserID, MovieID, rating, Timestamp)

1. Upload u.data into Colab



Step 2: Implement this version of MLlib - Collaborative Filtering Examples

Set up the SparkContext and load your data

```python
from pyspark import SparkContext
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Initialize SparkContext
sc = SparkContext(appName="PythonCollaborativeFilteringExample")

# Load and parse the data
data = sc.textFile("/content/u.data")

# Convert data to (UserID, MovieID, rating) format
ratings = data.map(lambda l: l.strip().split('\t'))\
              .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

# Display the first few ratings to verify the data
ratings.take(5)
```

```
[Rating(user=196, product=242, rating=3.0),
 Rating(user=186, product=302, rating=3.0),
 Rating(user=22, product=377, rating=1.0),
 Rating(user=244, product=51, rating=2.0),
 Rating(user=166, product=346, rating=1.0)]
```

Build and Evaluate the Model

```
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]))

# Join input rating ((user, product), rate1) with predicted rating
# ((user, product), rate2) to create ((user, product), (rate1, rate2))
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, "/content/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, "/content/myCollaborativeFilter")
```

```
Mean Squared Error = 0.4808234959187335
```

Verify and Save the Results

```
user_id = 196
user_ratings = ratings.filter(lambda r: r[0] == user_id).collect()

print(f"Ratings given by user {user_id}:")
for r in user_ratings:
    print(f"Movie ID: {r.product}, Rating: {r.rating}")

# Generate top 10 movie recommendations for a specific user
recommendations = sameModel.recommendProducts(user_id, 10)

print(f"\nTop 10 recommendations for user {user_id}:")
for r in recommendations:
    print(f"Movie ID: {r.product}, Predicted Rating: {r.rating}")
```

Results:

```
Ratings given by user 196:
Movie ID: 242, Rating: 3.0
Movie ID: 393, Rating: 4.0
Movie ID: 381, Rating: 4.0
Movie ID: 251, Rating: 3.0
Movie ID: 655, Rating: 5.0
Movie ID: 67, Rating: 5.0
Movie ID: 306, Rating: 4.0
Movie ID: 238, Rating: 4.0
Movie ID: 663, Rating: 5.0
Movie ID: 111, Rating: 4.0
Movie ID: 580, Rating: 2.0
Movie ID: 25, Rating: 4.0
Movie ID: 286, Rating: 5.0
Movie ID: 94, Rating: 3.0
Movie ID: 692, Rating: 5.0
Movie ID: 8, Rating: 5.0
Movie ID: 428, Rating: 4.0
Movie ID: 1118, Rating: 4.0
Movie ID: 70, Rating: 3.0
Movie ID: 66, Rating: 3.0
Movie ID: 257, Rating: 2.0
Movie ID: 108, Rating: 4.0
Movie ID: 202, Rating: 3.0
Movie ID: 340, Rating: 3.0
Movie ID: 287, Rating: 3.0
Movie ID: 116, Rating: 3.0
Movie ID: 382, Rating: 4.0
```

```
Movie ID: 285, Rating: 5.0
Movie ID: 1241, Rating: 3.0
Movie ID: 1007, Rating: 4.0
Movie ID: 411, Rating: 4.0
Movie ID: 153, Rating: 5.0
Movie ID: 13, Rating: 2.0
Movie ID: 762, Rating: 3.0
Movie ID: 173, Rating: 2.0
Movie ID: 1022, Rating: 4.0
Movie ID: 845, Rating: 4.0
Movie ID: 269, Rating: 3.0
Movie ID: 110, Rating: 1.0
```

```
Top 10 recommendations for user 196:
Movie ID: 1643, Predicted Rating: 8.341500792712392
Movie ID: 6, Predicted Rating: 8.139425806016602
Movie ID: 791, Predicted Rating: 7.915776653215841
Movie ID: 1183, Predicted Rating: 7.696441443174423 5
Movie ID: 904, Predicted Rating: 7.646503783174321
Movie ID: 548, Predicted Rating: 7.587493292387760 5
Movie ID: 959, Predicted Rating: 7.499086236682457
Movie ID: 703, Predicted Rating: 7.301922219137116
Movie ID: 1100, Predicted Rating: 7.214407483018 43
Movie ID: 532, Predicted Rating: 7.19649808628942
```

Github link

https://github.com/emilywengster/sfbu/tree/4cf8faf454d53b8cd7da2b100f8157a9fc964b9c/Cloud%20Computing/Machine%20Learning/%20Movie%20Recommendation%20System