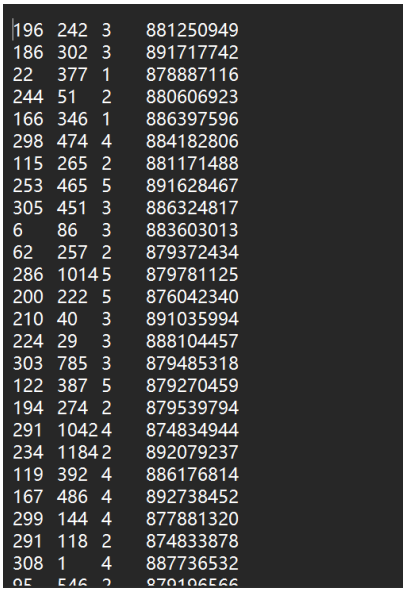
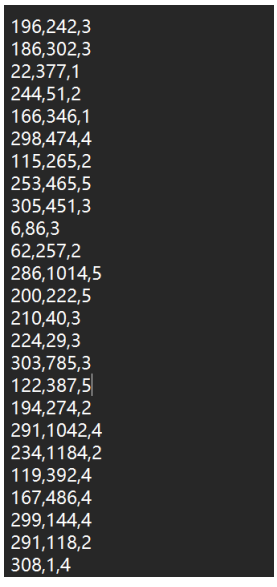
Project: Movie Recommendation with MLlib - Collaborative Filtering (implementaiton 2)

TianzeKang2000/Movie-Recommendation-System (github.com)

Step 1: Convert MoveLens' data (UserID, MovieID, rating, Timestamp) into the format of (UserID, MovieID, rating)

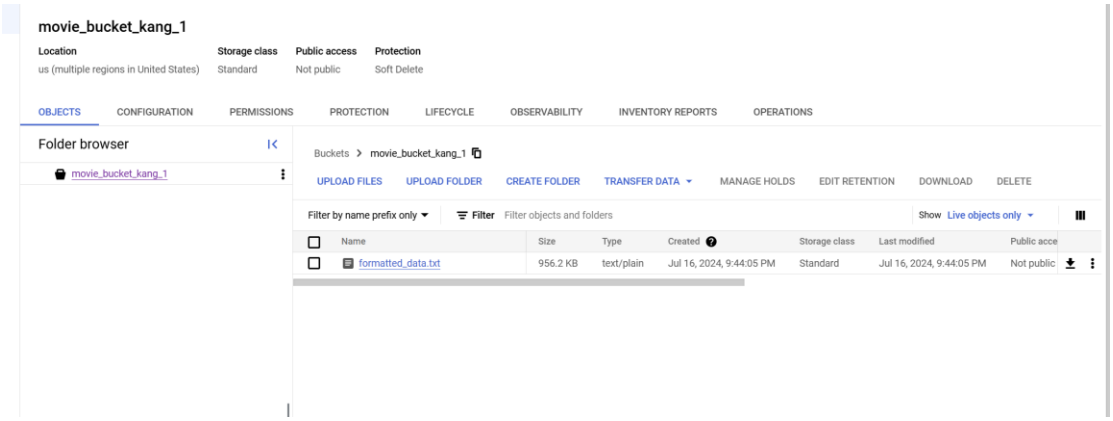
```
# Load the data file and convert it to the required format

input_file_path = '/mnt/data/New data.txt'
output_file_path = '/mnt/data/formatted_data.txt'

# Read the input file
with open(input_file_path, 'r') as file:
    data = file.readlines()

# Process the data
formatted_data = []
for line in data:
    parts = line.split()
    if len(parts) == 4:
        formatted_data.append(f"{parts[0]},{parts[1]},{parts[2]}\n")

# Write the formatted data to a new file
with open(output_file_path, 'w') as file:
    file.writelines(formatted_data)

output_file_path
```





Step 2 Implement this version of [MLlib - Collaborative Filtering Examples]

Using the GCP Console to Upload the File:
Navigate to Cloud Storage, create a new bucket.
Click the "Upload Files" button and select C:\Users\KANG\Downloads\formatted_data.txt to upload.



Create a new Python script named recommendation.py:

```
from pyspark.sql import SparkSession
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating

# Initialize Spark session
spark = SparkSession.builder.appName("MovieRecommendation").getOrCreate()
sc = spark.sparkContext

# Load and parse the data
data = sc.textFile("gs://movie_bucket_kang_1/formatted_data.txt")
ratings = data.map(lambda l: l.split(','))\
        .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

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

# Predict user ratings for movies
testdata = ratings.map(lambda p: (p[0], p[1]))
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))

# Calculate Mean Squared Error
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))
```

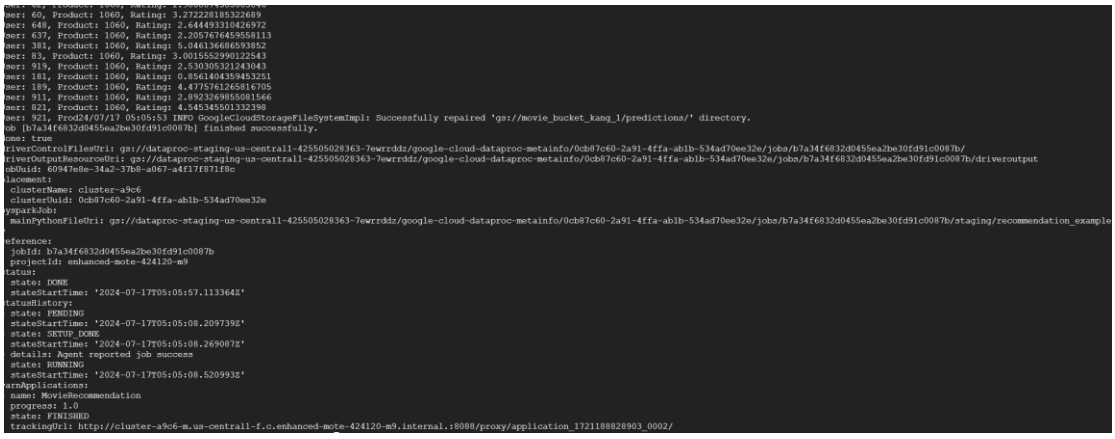
# Save the model
model.save(sc, "gs://movie_bucket_kang_1/model/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc, "gs://movie_bucket_kang_1/model/myCollaborativeFilter")

spark.stop()

Submit the PySpark Job

In Cloud Shell, submit the PySpark job to your Dataproc cluster using the following command:

gcloud dataproc jobs submit pyspark recommendation.py --cluster=cluster-a9c6 --region=us-centr



```
User: 49, Product: 320, Rating: 5.3506158193725835
User: 617, Product: 320, Rating: 4.82065962165874
User: 833, Product: 320, Rating: 3.9459219576166227
User: 13, Product: 320, Rating: 1.1587396127296636
User: 747, Product: 320, Rating: 5.490706479128973
User: 234, Product: 1330, Rating: 2.936149990875719
User: 181, Product: 1330, Rating: 1.0629440294165278
User: 943, Product: 1330, Rating: 2.9578486159627584
User: 211, Product: 1330, Rating: 3.0941604634584916
User: 14, Product: 408, Rating: 5.364819477762742
User: 514, Product: 408, Rating: 4.149183501579902
User: 226, Product: 408, Rating: 4.423223163562723
User: 160, Product: 408, Rating: 4.153093371830105
User: 420, Product: 408, Rating: 4.316801740306228
User: 58, Product: 408, Rating: 4.436220926400643
User: 338, Product: 408, Rating: 4.424361776020332
User: 216, Product: 408, Rating: 4.227886496138721
User: 292, Product: 408, Rating: 4.636627592094855
User: 92, Product: 408, Rating: 4.320357428047913
User: 344, Product: 408, Rating: 5.106607836349356
User: 664, Product: 408, Rating: 4.801354555349273
User: 84, Product: 408, Rating: 4.875224892218625
User: 592, Product: 408, Rating: 5.092428295030623
User: 536, Product: 408, Rating: 4.99543059017358
User: 402, Product: 408, Rating: 4.608248581782899
User: 334, Product: 408, Rating: 4.474194719977783
User: 312, Product: 408, Rating: 4.519703965268387
User: 158, Product: 408, Rating: 4.8338565484401625
User: 286, Product: 408, Rating: 4.053583716249065
User: 634, Product: 408, Rating: 3.7177468168353447
User: 70, Product: 408, Rating: 4.466932630415672
User: 906, Product: 408, Rating: 4.223861211925433
User: 658, Product: 408, Rating: 4.042194351769586
User: 738, Product: 408, Rating: 4.4810835522805945
User: 822, Product: 408, Rating: 4.80934150577896
User: 214, Product: 408, Rating: 4.316519452228186
User: 654, Product: 408, Rating: 4.72090460672466
User: 852, Product: 408, Rating: 4.284863745363509
User: 56, Product: 408, Rating: 3.712554365528061
User: 32, Product: 408, Rating: 3.2876072775417784
User: 684, Product: 408, Rating: 4.406452492186036
User: 806, Product: 408, Rating: 4.709603233936851
User: 526, Product: 408, Rating: 4.609398672485364
User: 412, Product: 408, Rating: 4.095676586727922
User: 622, Product: 408, Rating: 5.080670713431014
User: 838, Product: 408, Rating: 4.397468513950832
User: 458, Product: 408, Rating: 4.876656682291041
User: 868, Product: 408, Rating: 5.0349678214073545
User: 18, Product: 408, Rating: 4.735741669085218
User: 342, Product: 408, Rating: 4.127261017238123
User: 268, Product: 408, Rating: 4.34540327151419
User: 864, Product: 408, Rating: 5.092805315337992
User: 142, Product: 408, Rating: 3.9740423146469883
User: 6, Product: 408, Rating: 4.580530130400435
```

You can see the results by open predictions.