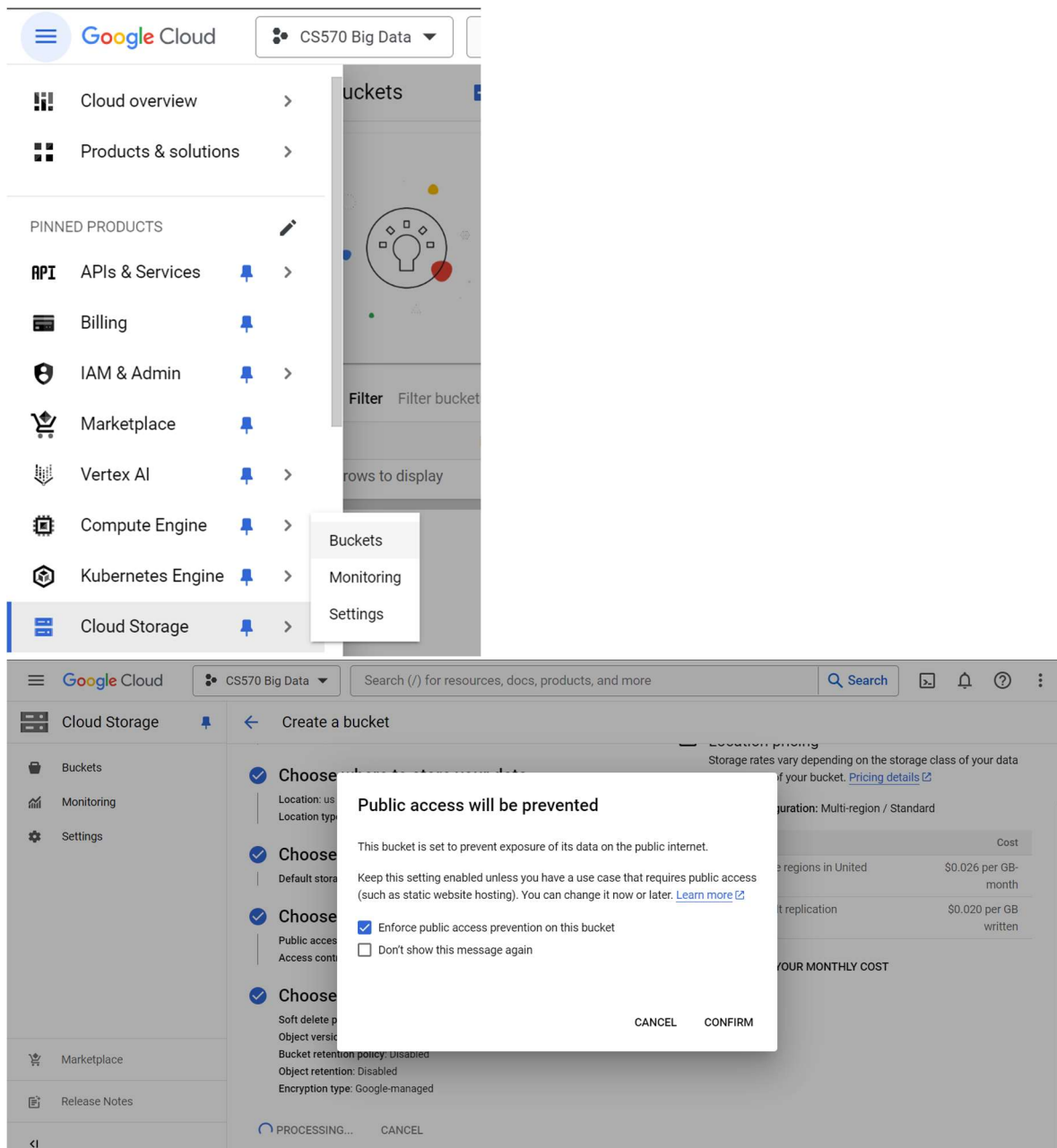


Week 8: Homework 1: Project: Movie Recommendation with MLlib - Collaborative Filtering (implementaiton 2)

Steps to complete the project:

1. Create a bucket

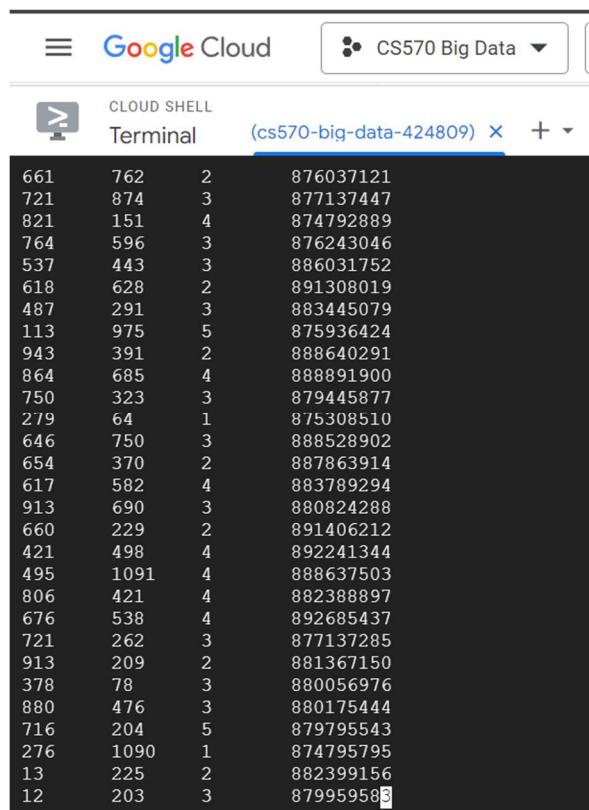


Step1: Preparing and Transform data

For this part of the project, we need to load the data we need for the recommendation system and save it in our cloud.

1. **Create a file named u.data** and load the rating data into this file. The data has an initial format of (UserID, MovieID, rating, Timestamp)

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim u.data
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```



661	762	2	876037121
721	874	3	877137447
821	151	4	874792889
764	596	3	876243046
537	443	3	886031752
618	628	2	891308019
487	291	3	883445079
113	975	5	875936424
943	391	2	888640291
864	685	4	888891900
750	323	3	879445877
279	64	1	875308510
646	750	3	888528902
654	370	2	887863914
617	582	4	883789294
913	690	3	880824288
660	229	2	891406212
421	498	4	892241344
495	1091	4	888637503
806	421	4	882388897
676	538	4	892685437
721	262	3	877137285
913	209	2	881367150
378	78	3	880056976
880	476	3	880175444
716	204	5	879795543
276	1090	1	874795795
13	225	2	882399156
12	203	3	879959585

2. **Transform the data:** next we need to convert the initial format to (UserID, MovieID, rating) this format. So, we can use a bash script to do that.

The script can be formatted as follows:

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim transform_data.sh
```

- First copy this bash script into a script file called *transform_data.sh*

```
#!/bin/bash
cat u.data | while read userid movieid rating timestamp
do
    echo "${userid},${movieid},${rating}"
done > u_transformed_data.csv
```



CLOUD SHELL

Terminal

(cs570-big-data-424809) X + ▾

```
#!/bin/bash
cat u.data | while read userid movieid rating timestamp
do
    echo "${userid},${movieid},${rating}"
done > u_transformed_data.csv
~
~
```

- Make the file executable:
chmod +x transform_data.sh
- Execute the script
./transform_data.sh

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ vim transform_data.sh
shagos90499@cloudshell:~ (cs570-big-data-424809)$ chmod +x transform_data.sh
shagos90499@cloudshell:~ (cs570-big-data-424809)$ ^C
shagos90499@cloudshell:~ (cs570-big-data-424809)$ ./transform_data.sh
shagos90499@cloudshell:~ (cs570-big-data-424809)$ cat transform_data.sh
#!/bin/bash
cat u.data | while read userid movieid rating timestamp
do
    echo "${userid},${movieid},${rating}"
done > u_transformed_data.csv
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

The shell scripts read each line of the u.data file, split the data by spaces, and reformat it to display only the UserID, MovieID, and rating fields separated by commas. This transformation removes the timestamp field and formats the data into CSV format, which is easier to work with for further processing. The transformed data will be saved in the file u_transformed_data.csv.

Step2: Uploading the data to cloud bucket

Next, we will upload the transformed data that we saved in the u_transformed_data.csv to our cloud storage bucket.

```
gsutil cp u_transformed_data.csv gs://big_data_movie_recommendation
```

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ gsutil cp u_transformed_data.csv gs://big_data_movie_recommendation
Copying file:///u transformed_data.csv [Content-Type=text/csv]...
/ [1 files][956.2 KiB/956.2 KiB]
Operation completed over 1 objects/956.2 KiB.
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

Step 3: Create and upload the PySpark script

In this step, we will create the pyspark script that will perform the collaborative filtering and upload that file to the cloud storage bucket.

1. Create a file named **recommendation_example.py**
vim recommendation_example.py
2. **Copy this script** in the file but replace the path to the text file since currently it is working from another link. Replace it with the correct path to your pyspark script in the cloud bucket.

```
"""
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("gs://big_data_movie_recommendation/u_transformed_data.csv")

    # Each line is
    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: 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]))
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, "target/tmp/myCollaborativeFilter")
sameModel = MatrixFactorizationModel.load(sc,
"target/tmp/myCollaborativeFilter")
# $example off$
```



CLOUD SHELL

Terminal

(cs570-big-data-424809) X + ▾

```
"""
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("data/mllib/als/test.data")

    # Each line is
    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: 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
```

3. Upload the script to the bucket:

```
gsutil cp recommendation_example.py
gs://big_data_movie_recommendation
```

This PySpark script implements collaborative filtering using the Alternating Least Squares (ALS) algorithm to build a movie recommendation system. It loads user-movie rating data, trains a recommendation model with the ALS algorithm, evaluates the model by calculating the Mean Squared Error (MSE) of predictions, and saves the trained model for future use.

Step 4: Submit the pyspark job to google Dataproc

To submit the pyspark script, first we need to create a Dataproc cluster and submit the script

1. Create a Dataproc cluster with this command

```
gcloud dataproc clusters create spark-cluster \
--region us-west1 \
--zone us-west1-a \
--single-node
```

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ gcloud dataproc clusters create spark-cluster \
--region us-west1 \
--zone us-west1-a \
--single-node
Waiting on operation [projects/cs570-big-data-424809/regions/us-west1/operations/c9280804-fdd4-3f83-bfde-35bb38b5cd77].
Waiting for cluster creation operation...
WARNING: No image specified. Using the default image version. It is recommended to select a specific image version in production, as the default image version may change at any time.
WARNING: Consider using Auto Zone rather than selecting a zone manually. See https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/auto-zone.
WARNING: Failed to validate permissions required for default service account: '720083396959-compute@developer.gserviceaccount.com'. Cluster creation could be successful if required permissions have been granted to the respective service accounts as mentioned in the document https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/service-accounts#dataproc_service_accounts_2. This could be due to Cloud Resource Manager API hasn't been enabled in your project 0083396959' before or it is disabled. Enable it by visiting 'https://console.developers.google.com/apis/api/cloudresourcemanager.googleapis.com/overview?project=720083396959'.
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security risk.
Waiting for cluster creation operation...done.
Created [https://dataproc.googleapis.com/v1/projects/cs570-big-data-424809/regions/us-west1/clusters/spark-cluster] Cluster placed in zone [us-west1-a].
shagos90499@cloudshell:~ (cs570-big-data-424809)$
```

2. Next, submit the pyspark job to the Dataproc cluster

```
gcloud dataproc jobs submit pyspark
gs://big_data_ml_recommendation_sys/recommendation_example.py \
--cluster spark-cluster \
--region us-west1
```

```
shagos90499@cloudshell:~ (cs570-big-data-424809)$ gcloud dataproc jobs submit pyspark gs://big_data_movie_recommendation/recommendation_example.py \
--cluster spark-cluster \
--region us-west1
Job [cf0743c466ff4eaf896b378f28ccd491] submitted.
Waiting for job output...
24/07/17 05:19:47 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
24/07/17 05:19:47 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
24/07/17 05:19:47 INFO org.apache.spark.SparkEnv: Registering BlockManagerMasterHeartbeat
24/07/17 05:19:47 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
24/07/17 05:19:47 INFO org.sparkproject.jetty.util.log: Logging initialized @4573ms to org.sparkproject.jetty.util.log.Slf4jLog
24/07/17 05:19:47 INFO org.sparkproject.jetty.server.Server: jetty-9.4.40.v20210413; built: 2021-04-13T20:42:42.668Z; git: b881a572662e1943a14ae12e7e174; jvm 1.8.0_412-b08
24/07/17 05:19:47 INFO org.sparkproject.jetty.server.Server: Started @4700ms
24/07/17 05:19:47 INFO org.sparkproject.jetty.server.AbstractConnector: Started ServerConnector@1e7102e3{HTTP/1.1, (http/1.1)}{0.0.0.0:43711}
24/07/17 05:19:48 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at spark-cluster-m/10.138.0.25:8032
24/07/17 05:19:49 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at spark-cluster-m/10.138.0.25:10200
24/07/17 05:19:50 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
24/07/17 05:19:50 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
```



```

driverOutputResourceUri: gs://dataproc-staging-us-west1-720083396959-ncaxf7jl/google-cloud-dataproc-t
69435d94d883933a2d1d56/driveroutput
jobUuid: bb828cfa-ea92-3b1a-bd8c-e9b1098ca204
placement:
  clusterName: spark-cluster
  clusterUuid: e25594a6-9c3d-4676-a348-e90edf4809ba
pysparkJob:
  mainPythonFileUri: gs://big_data_movie_recommendation/recommendation_example.py
reference:
  jobId: ba73e8893969435d94d883933a2d1d56
  projectId: cs570-big-data-424809
status:
  state: DONE
  stateStartTime: '2024-07-17T05:40:10.129484Z'
statusHistory:
- state: PENDING
  stateStartTime: '2024-07-17T05:39:04.600621Z'
- state: SETUP_DONE
  stateStartTime: '2024-07-17T05:39:04.628825Z'
- details: Agent reported job success
  state: RUNNING
  stateStartTime: '2024-07-17T05:39:04.811591Z'
yarnApplications:
- name: PythonCollaborativeFilteringExample
  progress: 1.0
  state: FINISHED
  trackingUrl: http://spark-cluster-m:8088/proxy/application_1721193394462_0004/

```

Mean Squared Error = 0.48435603686025513

24/07/17 05:40:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1

3. The output of the pyspark job is as follows:
The Mean Squared Error = 0.48435603686025513

c

```

24/07/17 05:39:18 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
Mean Squared Error = 0.48435603686025513
24/07/17 05:40:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
24/07/17 05:40:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
24/07/17 05:40:01 WARN org.apache.hadoop.util.concurrent.ExecutorHelper: Thread (Thread[GetFileInf
java.lang.InterruptedException

```