Kadirbek Sharau

Collaborative Filtering Recommendation System

Overview

This project focuses on building a collaborative filtering recommendation system using PySpark and Google Cloud Dataproc. Collaborative filtering predicts the interests of a user by collecting preferences from many users.

Problem Statement

The goal is to provide accurate movie recommendations by predicting user ratings for movies they haven't watched yet, based on historical user-movie interaction data.

Scope

The project involves:

- Data preparation
- Model training using Alternating Least Squares (ALS) in PySpark
- Evaluation of the model's performance
- Deployment on Google Cloud Dataproc

Table of Contents

Setup the Environment

1. **Create a Google Cloud Storage (GCS) Bucket**:
    - Create a bucket in GCS to store your scripts and data.

gsutil mb gs://movie_recommendation_with_mllib_collaborative_filter

2.
3. **Upload Data and Scripts to GCS**:
    - Upload the movies.csv, ratings.csv, and your PySpark script (Recommendation_Engine_MovieLens.py) to your GCS bucket.

gsutil cp movies.csv gs://movie_recommendation_with_mllib_collaborative_filter/
gsutil cp ratings.csv gs://movie_recommendation_with_mllib_collaborative_filter/
gsutil cp Recommendation_Engine_MovieLens.py
gs://movie_recommendation_with_mllib_collaborative_filter/

4.

Download Programs and Related Documentation

Clone the repository:
git clone https://github.com/ASD-Are/Big_Data

-
- Download the MovieLens dataset: MovieLens 100k Dataset

Process of Program Execution

Data Preparation

Create the u.data File:
vim u.data

1. Populate u.data with your data in the format (UserID, MovieID, rating, Timestamp).

Transform the Raw Data:
cat u.data | tr -s ' ' | cut -d' ' -f1-3 | tr ' ' ',' > u_data_transformed.csv

2. This command trims extra spaces, extracts the first three fields (UserID, MovieID, rating), and replaces spaces with commas. The transformed data is saved in u_data_transformed.csv.

Upload the Transformed Data:
gsutil cp u_data_transformed.csv gs://movie_recommendation_with_mllib_collaborative_filter/

3.

Create PySpark Script

Create a PySpark script Recommendation_Engine_MovieLens.py with the following content:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, explode
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
import argparse

# Parse command-line arguments
parser = argparse.ArgumentParser()
parser.add_argument('--input_path_movies', required=True)
parser.add_argument('--input_path_ratings', required=True)
args = parser.parse_args()

# Initialize Spark session
spark = SparkSession.builder.appName('Recommendations').getOrCreate()

# Load data from GCS
movies = spark.read.csv(args.input_path_movies, header=True)
ratings = spark.read.csv(args.input_path_ratings, header=True)

# Preprocess data
ratings = ratings \
    .withColumn('userId', col('userId').cast('integer')) \
    .withColumn('movieId', col('movieId').cast('integer')) \
    .withColumn('rating', col('rating').cast('float')) \
    .drop('timestamp')

# Split data into training and testing sets
(train, test) = ratings.randomSplit([0.8, 0.2], seed=1234)

# Build ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating",
          nonnegative=True, implicitPrefs=False, coldStartStrategy="drop")
param_grid = ParamGridBuilder() \
    .addGrid(als.rank, [10, 50, 100, 150]) \
    .addGrid(als.regParam, [.01, .05, .1, .15]) \
    .build()

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid,
                    evaluator=evaluator, numFolds=5)
```

```
# Train model
model = cv.fit(train)
best_model = model.bestModel

# Evaluate model
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(f"Root-mean-square error = {RMSE}")

# Generate recommendations
nrecommendations = best_model.recommendForAllUsers(10)
nrecommendations = nrecommendations \
    .withColumn("rec_exp", explode("recommendations")) \
    .select('userId', col("rec_exp.movieId"), col("rec_exp.rating"))
nrecommendations.show()

# Join with movie titles for better interpretability
nrecommendations.join(movies, on='movieId').filter('userId = 100').show()
ratings.join(movies, on='movieId').filter('userId = 100').sort('rating',
                                                  ascending=False).limit(10).show()

# Stop Spark session
spark.stop()
```

-

Create Dataproc Cluster

Create the cluster with the desired configuration:

```
gcloud dataproc clusters create spark-cluster \
    --region us-west1 \
    --zone us-west1-a \
    --master-machine-type n1-standard-4 \
    --worker-machine-type n1-standard-4 \
    --num-workers 2
```

1.

Submit PySpark Job

Submit the PySpark job to the Dataproc cluster:

```
gcloud dataproc jobs submit pyspark
gs://movie_recommendation_with_mllib_collaborative_filter/Recommendation_Engine_MovieLe
ns.py --cluster=spark-cluster --region=us-west1 --
```

--input_path_movies=gs://movie_recommendation_with_mllib_collaborative_filter/movies.csv
--input_path_ratings=gs://movie_recommendation_with_mllib_collaborative_filter/ratings.csv

1.

Screenshot of Execution Results

After running the job, the output will display the Root Mean Squared Error (RMSE) of the model.
Root-mean-square error = 0.48149423210378404

-

Conclusion

- Successfully built a collaborative filtering recommendation system using PySpark and Google Cloud Dataproc.
- Data preparation involved transforming and uploading the MovieLens dataset to Google Cloud Storage.
- Model training utilized the ALS algorithm on user-movie interaction data.
- The model's performance was evaluated using Root Mean Squared Error (RMSE), achieving a value of 0.48149423210378404, indicating reasonable prediction accuracy.

References

- MovieLens Dataset
- Collaborative Filtering - RDD-based API