```
#!/usr/bin/env python
"""
Copyright Google Inc. 2016
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""


import os
import sys
import pickle
import itertools
from math import sqrt
from operator import add
from os.path import join, isfile, dirname
from pyspark import SparkContext, SparkConf, SQLContext
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from pyspark.sql.types import StructType, StructField, StringType, FloatType

CLOUDSQL_INSTANCE_IP = '104.198.212.57'   # CHANGE (database server IP)
CLOUDSQL_DB_NAME = 'recommendation_spark'
CLOUDSQL_USER = 'root'
CLOUDSQL_PWD  = 'root'  # CHANGE

conf = SparkConf().setAppName("train_model")
sc = SparkContext(conf=conf)
sqlContext = SQLContext(sc)

jdbcDriver = 'com.mysql.jdbc.Driver'
jdbcUrl    = 'jdbc:mysql://%s:3306/%s?user=%s&password=%s' % (CLOUDSQL_INSTANCE_IP,
CLOUDSQL_DB_NAME, CLOUDSQL_USER, CLOUDSQL_PWD)

# checkpointing helps prevent stack overflow errors
sc.setCheckpointDir('checkpoint/')

# Read the ratings and accommodations data from Cloud SQL
dfRates = sqlContext.read.format('jdbc').options(driver=jdbcDriver, url=jdbcUrl,
dbtable='Rating', useSSL='false').load()
dfAccos = sqlContext.read.format('jdbc').options(driver=jdbcDriver, url=jdbcUrl,
dbtable='Accommodation', useSSL='false').load()
print("read ...")

# train the model
model = ALS.train(dfRates.rdd, 20, 20) # you could tune these numbers, but these are
reasonable choices
print("trained ...")

# use this model to predict what the user would rate accommodations that she has not rated
allPredictions = None
for USER_ID in range(0, 100):
  dfUserRatings = dfRates.filter(dfRates.userId == USER_ID).rdd.map(lambda r:
r.accoId).collect()
  rddPotential  = dfAccos.rdd.filter(lambda x: x[0] not in dfUserRatings)
  pairsPotential = rddPotential.map(lambda x: (USER_ID, x[0]))
  predictions = model.predictAll(pairsPotential).map(lambda p: (str(p[0]), str(p[1]),
```

```
float(p[2])))
  predictions = predictions.takeOrdered(5, key=lambda x: -x[2]) # top 5
  print("predicted for user={0}".format(USER_ID))
  if (allPredictions == None):
    allPredictions = predictions
  else:
    allPredictions.extend(predictions)

# write them
schema = StructType([StructField("userId", StringType(), True), StructField("accoId",
StringType(), True), StructField("prediction", FloatType(), True)])
dfToSave = sqlContext.createDataFrame(allPredictions, schema)
dfToSave.write.jdbc(url=jdbcUrl, table='Recommendation', mode='overwrite')
```