import java.util

import org.apache.spark.{ SparkConf, SparkContext }

import org.apache.spark.rdd.RDD

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.Row

import java.util.HashMap

import java.util.Date

import java.text.SimpleDateFormat

import org.apache.spark.mllib.classification.NaiveBayes

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.SparkContext;

import org.apache.spark.SparkConf;

import org.apache.spark.mllib.recommendation.ALS

import org.apache.spark.mllib.recommendation.MatrixFactorizationModel

import org.apache.spark.mllib.recommendation.Rating

import scala.collection.mutable.ListBuffer

import org.apache.spark.sql.SQLContext

var type\_of\_restaurant = readLine("Enter type of Food:")

var city = readLine("Enter City:")

var residingState = readLine("Enter State:")

var user\_id = readLine("Enter User ID:")

residingState = residingState.toLowerCase

city = city.toLowerCase

type\_of\_restaurant = type\_of\_restaurant.toLowerCase

var df = sqlContext.read.format("com.databricks.spark.csv").option("header","false").option("parserLib","UNIVOCITY").load("/home/hadoop/business.csv")

val rows: RDD[Row] = df.rdd

var restaurantList = scala.collection.mutable.MutableList[String]()

var ResName = new util.HashMap[String, String]()

var ResNameRat = scala.collection.mutable.Map[String, Double]()

rows.collect().foreach {

i =>

var lineStringArr = (i.toString()).split(",")

val categories = lineStringArr(19).toLowerCase

val bid = lineStringArr(14)

val name = lineStringArr(22)

val restaurantState = lineStringArr(37).toLowerCase

val restaurantCity = lineStringArr(0).toLowerCase

if (categories.contains("restaurant") && categories.contains(type\_of\_restaurant) && restaurantCity.contains(city) && restaurantState.contains(residingState)) {

restaurantList += bid

ResName.put(bid, name)

if(!lineStringArr(1).equals("stars"))

{

val rat = lineStringArr(1).toDouble

ResNameRat.put(name, rat)

}

}

}

df = sqlContext.read.format("com.databricks.spark.csv").option("header","false").option("parserLib", "UNIVOCITY").load("/home/hadoop/review.csv")

var userReviewed = false

val reviewsRows: RDD[Row] = df.rdd.filter(x => restaurantList.contains(x(3)))

var hp = new util.HashMap[String, String]()

reviewsRows.collect().foreach {

i =>

val lineStringArr = i.toString.split(",")

val uid = lineStringArr(0).trim.replace("[", "")

val review\_bid = lineStringArr(3).trim

val rating = lineStringArr(5).toFloat

val dateReviewed = lineStringArr(6)

if (uid.contains(user\_id) && restaurantList.contains(review\_bid)) {

userReviewed = true

}

if (restaurantList.contains(review\_bid)) {

val dt1 = new SimpleDateFormat("yyyy-mm-dd")

var date1 = dt1.parse(dateReviewed)

var dateInMilli = date1.getTime

val uid\_bid = uid + "|" + review\_bid

val rating\_date = rating + "|" + dateInMilli

if (hp.containsKey(uid\_bid)) {

var existingDateArray = hp.get(uid\_bid).split("\\|")

var existingDate = existingDateArray(1).toLong

if (dateInMilli > existingDate) {

hp.put(uid\_bid, rating\_date)

}

} else {

hp.put(uid\_bid, rating\_date)

}

}

}

var useridToNum = new util.HashMap[String, Int]()

var businessidToNum = new util.HashMap[String, Int]()

var usercount = 2

for (i <- hp.keySet().toArray()) {

var tempArray = i.toString().split("\\|")

var tempuid = tempArray(0)

var tempbid = tempArray(1)

if (tempuid.equals(user\_id)) {

useridToNum.put(tempuid, 1)

} else {

useridToNum.put(tempuid, usercount)

}

businessidToNum.put(tempbid, usercount)

usercount += 1

}

var ALSList = scala.collection.mutable.MutableList[String]()

for (i <- hp.keySet().toArray()) {

var tempArray = i.toString().split("\\|")

var temp = useridToNum.get(tempArray(0)) + "::" + businessidToNum.get(tempArray(1))

var valueArray = hp.get(i).toString().split("\\|")

temp = temp + "::" + valueArray(0) + "::" + valueArray(1)

ALSList += temp

}

sc.parallelize(ALSList, 1).saveAsTextFile("../RatingData9.dat")

var testHM = scala.collection.mutable.MutableList[String]()

for (i <- businessidToNum.keySet().toArray()) {

var temp = "1" + "::" + businessidToNum.get(i) + "::" + "0"

testHM += temp

}

sc.parallelize(testHM, 1).saveAsTextFile("../TestRatingData9.dat")

if (userReviewed == true) {

var data = sc.textFile("../RatingData9.dat/part-00000")

val ratings = data.map(\_.split("::") match {

case Array(userID, businessID, ratings, timestamp) =>

Rating(userID.toInt, businessID.toInt, ratings.toDouble)

})

data = sc.textFile("../TestRatingData9.dat/part-00000")

val testing = data.map(\_.split("::") match {

case Array(userID, businessID, ratings) =>

Rating(userID.toInt, businessID.toInt, ratings.toDouble)

})

val rank = 5

val numIterations = 10

val model = ALS.train(ratings, rank, numIterations, 0.01)

val usersProducts = testing.map {

case Rating(user, product, ratings) =>

(user, product)

}

val predictions =model.predict(usersProducts).map {

case Rating(user, product, rate) =>

((user, product), rate)

}

var l = predictions.sortBy(-\_.\_2)

l = sc.parallelize(l.take(5), 1)

var Top5Rest = scala.collection.mutable.MutableList[String]()

var Top5RestRat = scala.collection.mutable.Map[String,String]()

var ratMax = java.lang.Double.parseDouble("0")

for (i <- l.collect().toArray) {

var temp = i.\_1.\_2.toString()

var tempRat=i.\_2

if(ratMax<i.\_2)

{

ratMax = tempRat

}

Top5Rest += temp

Top5RestRat.put(temp, tempRat.toString())

}

println("Top 5 Restaurants:")

for (j <- businessidToNum.keySet().toArray()) {

val strValue = businessidToNum.get(j).toString()

if (Top5Rest.contains(strValue))

{

var currentRat = java.lang.Double.parseDouble(Top5RestRat.get(strValue).toString().replace("Some(", "").replace(")", ""))

currentRat = currentRat \* 5/(ratMax+1)

println(ResName.get(j.toString())+" "+currentRat)

}

}

}

else

{

var ratMapSeq=ResNameRat.toSeq.sortBy(-\_.\_2).take(5)

println("Top 5 Restaurants based on the search is ->")

ratMapSeq.foreach

{

x=>println(x.\_1+" "+x.\_2)

}

}