# [SRC] 10.1. Recommend Similar Users

import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer, StopWordsRemover}

import org.apache.spark.ml.Pipeline

val tokenizer = new Tokenizer()

.setInputCol("text")

.setOutputCol("words")

val stopWordsRemover = new StopWordsRemover()

.setInputCol(tokenizer.getOutputCol)

.setOutputCol("filtered")

val tf = new HashingTF()

.setInputCol(tokenizer.getOutputCol)

.setOutputCol("rawFeatures")

.setNumFeatures(20)

val idf = new IDF()

.setInputCol(tf.getOutputCol)

.setOutputCol("features")

val pipeline = new Pipeline()

.setStages(Array(tokenizer, stopWordsRemover, tf, idf))

val tweets = spark.read.parquet("/dataset/twitter/reference")

val model = pipeline.fit(tweets.select("text"))

val tfidf = model.transform(tweets.select("text"))

tfidf.select("features").take(5).foreach(println)

tfidf.cache

import org.apache.spark.mllib.linalg.{Vectors => MlLibVectors}

import org.apache.spark.mllib.linalg.distributed.{MatrixEntry, RowMatrix, CoordinateMatrix}

import org.apache.spark.ml.linalg.Vector

val vectors = tfidf.rdd.map(r => MlLibVectors.dense(r.getAs[Vector]("features").toDense.toArray))

val mat = new RowMatrix(vectors)

println(s"RowMatrix size: ${mat.numRows}x${mat.numCols}")

val rmTfidf = vectors.zipWithIndex().map({case (v, r) => {

val buf = scala.collection.mutable.ListBuffer.empty[MatrixEntry]

v.foreachActive((c, v) => buf += new MatrixEntry(r, c, v))

buf.toList

}}).flatMap(identity)

val cmat = new CoordinateMatrix(rmTfidf)

println(s"CoordinateMatrix size: ${cmat.numRows}x${cmat.numCols}")

val transposed = cmat.transpose()

println(s"Transposed CoordinateMatrix size: ${transposed.numRows}x${transposed.numCols}")

val indexedRowMatrix = transposed.toRowMatrix()

val brutforce = indexedRowMatrix.columnSimilarities()

println(brutforce.entries.map(me => ((me.i, me.j), me.value)).first)

val approx = indexedRowMatrix.columnSimilarities(0.1)

println(approx.entries.map(me => ((me.i, me.j), me.value)).first)

# [SRC] 10.2. Analyze Mentions with GraphX

import org.apache.spark.graphx.Edge

import org.apache.spark.graphx.Graph

val tweets = spark.read.parquet("/dataset/twitter/reference")

import org.apache.spark.sql.functions.explode

val mentions = tweets

.select(

tweets("id"),

tweets("user.screenName").as("screenName"),

explode(tweets("userMentionEntities.screenName")).as("mentionedScreenName")

)

mentions.show

val screenNames = mentions.rdd.flatMap({ mention =>

Seq(

mention.getAs[String]("screenName"),

mention.getAs[String]("mentionedScreenName")

)})

.distinct

.toDF("screenName")

val vertices = screenNames.rdd.map { mention =>

val screenName = mention.getAs[String]("screenName")

(screenName.hashCode.toLong, screenName)

}

val edges = mentions.rdd.map ({ mention =>

val id = mention.getAs[Long]("id")

val screenName = mention.getAs[String]("screenName")

val mentionedScreenName = mention.getAs[String]("mentionedScreenName")

Edge(screenName.hashCode.toLong, mentionedScreenName.hashCode.toLong, id)

})

val graph = Graph(vertices, edges)

graph.cache

graph.vertices.take(5).foreach(println)

graph.edges.take(5).foreach(println)

graph.triplets.take(5).foreach(println)

val ranks = graph.pageRank(0.0001).vertices

val ranksByScreenName = screenNames

.rdd

.map(r => {

val screenName = r.getAs[String]("screenName")

(screenName.hashCode().toLong, screenName)

})

.join(ranks)

.map {

case (id, (screenName, rank)) => (screenName, rank)

}

println(ranksByScreenName.take(10).mkString("\n"))

val connectedComponents = graph.connectedComponents.cache

val ccCounts = connectedComponents.vertices.map{case (\_, vertexId) => vertexId}.countByValue

val topComponent = ccCounts.toSeq.sortBy{case (componentId, count) => count}.reverse.head

val ccRdd = graph.vertices.innerJoin(connectedComponents.vertices) {

case (vertexId, hashTag, componentId) => (hashTag, componentId)

}

val mentionTopComponent = ccRdd

.filter{ case (vertexId, (hashTag, componentId)) => (componentId == topComponent.\_1) }

.map{ case (vertexId, (mention, componentId)) => mention }

val ccByScreenName = screenNames

.rdd

.map(r => {

val screenName = r.getAs[String]("screenName")

(screenName.hashCode().toLong, screenName)

})

.join(cc)

.map { case (id, (username, cc)) => (username, cc) }

println(ccByScreenName.take(10).mkString("\n"))