package com.twitter.simclusters\_v2.scalding.tweet\_similarity

import com.twitter.ml.api.DailySuffixFeatureSource

import com.twitter.ml.api.DataSetPipe

import com.twitter.ml.api.RichDataRecord

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.Execution

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.job.TwitterExecutionApp

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.scalding.common.Util

import com.twitter.simclusters\_v2.tweet\_similarity.TweetSimilarityFeatures

import java.util.TimeZone

object DatasetTopKAnalysisJob {

case class TweetPairWithStats(

queryTweet: TweetId,

candidateTweet: TweetId,

cooccurrenceCount: Double,

coengagementCount: Double,

coengagementRate: Double)

def getCoocurrenceTweetPairs(dataset: DataSetPipe): TypedPipe[TweetPairWithStats] = {

val featureContext = dataset.featureContext

dataset.records

.map { record =>

val richDataRecord = new RichDataRecord(record, featureContext)

val coengaged =

if (richDataRecord

.getFeatureValue(TweetSimilarityFeatures.Label)

.booleanValue) 1

else 0

(

(

richDataRecord.getFeatureValue(TweetSimilarityFeatures.QueryTweetId).toLong,

richDataRecord.getFeatureValue(TweetSimilarityFeatures.CandidateTweetId).toLong),

(1, coengaged)

)

}.sumByKey

.map {

case ((queryTweet, candidateTweet), (coocurrenceCount, coengagementCount)) =>

TweetPairWithStats(

queryTweet,

candidateTweet,

coocurrenceCount.toDouble,

coengagementCount.toDouble,

coengagementCount.toDouble / coocurrenceCount.toDouble

)

}

}

def getQueryTweetToCounts(dataset: DataSetPipe): TypedPipe[(Long, (Int, Int))] = {

val featureContext = dataset.featureContext

dataset.records.map { record =>

val richDataRecord = new RichDataRecord(record, featureContext)

val coengaged =

if (richDataRecord

.getFeatureValue(TweetSimilarityFeatures.Label)

.booleanValue) 1

else 0

(

richDataRecord.getFeatureValue(TweetSimilarityFeatures.QueryTweetId).toLong,

(1, coengaged)

)

}.sumByKey

}

def printGlobalTopKTweetPairsBy(

tweetPairs: TypedPipe[TweetPairWithStats],

k: Int,

orderByFnt: TweetPairWithStats => Double

): Execution[Unit] = {

val topKTweetPairs =

tweetPairs.groupAll

.sortedReverseTake(k)(Ordering.by(orderByFnt))

.values

topKTweetPairs.toIterableExecution.map { s =>

println(s.map(Util.prettyJsonMapper.writeValueAsString).mkString("\n"))

}

}

def printTweetTopKTweetsBy(

groupedBy: Grouped[TweetId, TweetPairWithStats],

k: Int,

orderByFnt: TweetPairWithStats => Double,

descending: Boolean = true

): Execution[Unit] = {

if (descending) {

println("TweetTopKTweets (descending order)")

groupedBy

.sortedReverseTake(k)(Ordering.by(orderByFnt))

.toIterableExecution

.map { record => println(record.toString()) }

} else {

println("TweetTopKTweets (ascending order)")

groupedBy

.sortedTake(k)(Ordering.by(orderByFnt))

.toIterableExecution

.map { record => println(record.toString()) }

}

}

def printTweetPairStatsExec(

tweetPairs: TypedPipe[TweetPairWithStats],

k: Int

): Execution[Unit] = {

Execution

.sequence(

Seq(

Util.printSummaryOfNumericColumn(

tweetPairs.map(\_.cooccurrenceCount),

Some("Tweet-pair Coocurrence Count")),

printGlobalTopKTweetPairsBy(

tweetPairs,

k,

{ tweetPairs => tweetPairs.cooccurrenceCount }),

Util.printSummaryOfNumericColumn(

tweetPairs.map(\_.coengagementCount),

Some("Tweet-pair Coengagement Count")),

printGlobalTopKTweetPairsBy(

tweetPairs,

k,

{ tweetPairs => tweetPairs.coengagementCount }),

Util.printSummaryOfNumericColumn(

tweetPairs.map(\_.coengagementRate),

Some("Tweet-pair Coengagement Rate")),

printGlobalTopKTweetPairsBy(tweetPairs, k, { tweetPairs => tweetPairs.coengagementRate })

)

).unit

}

def printPerQueryStatsExec(dataset: DataSetPipe, k: Int): Execution[Unit] = {

val queryToCounts = getQueryTweetToCounts(dataset)

val topKQueryTweetsByOccurrence =

queryToCounts.groupAll

.sortedReverseTake(k)(Ordering.by { case (\_, (cooccurrenceCount, \_)) => cooccurrenceCount })

.values

val topKQueryTweetsByEngagement =

queryToCounts.groupAll

.sortedReverseTake(k)(Ordering.by { case (\_, (\_, coengagementCount)) => coengagementCount })

.values

Execution

.sequence(

Seq(

Util.printSummaryOfNumericColumn(

queryToCounts.map(\_.\_2.\_1),

Some("Per-query Total Cooccurrence Count")),

topKQueryTweetsByOccurrence.toIterableExecution.map { s =>

println(s.map(Util.prettyJsonMapper.writeValueAsString).mkString("\n"))

},

Util.printSummaryOfNumericColumn(

queryToCounts.map(\_.\_2.\_2),

Some("Per-query Total Coengagement Count")),

topKQueryTweetsByEngagement.toIterableExecution.map { s =>

println(s.map(Util.prettyJsonMapper.writeValueAsString).mkString("\n"))

}

)

).unit

}

def runTweetTopKTweetsOutputExecs(

tweetPairs: TypedPipe[TweetPairWithStats],

k: Int,

outputPath: String

): Execution[Unit] = {

tweetPairs

.groupBy(\_.queryTweet)

.sortedReverseTake(k)(Ordering.by(\_.coengagementRate))

.writeExecution(TypedTsv(outputPath + "/topK\_by\_coengagement\_rate"))

}

}

/\*\* To run:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/tweet\_similarity:dataset\_topk\_analysis-adhoc \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--main-class com.twitter.simclusters\_v2.scalding.tweet\_similarity.DatasetTopKAnalysisAdhocApp -- \

--date 2020-02-19 \

--dataset\_path /user/cassowary/adhoc/training\_data/2020-02-19\_class\_balanced/train \

--output\_path /user/cassowary/adhoc/training\_data/2020-02-19\_class\_balanced/train/analysis

\* \*/

object DatasetTopKAnalysisAdhocApp extends TwitterExecutionApp {

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

def job: Execution[Unit] = Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

implicit val dateRange: DateRange = DateRange.parse(args.list("date"))

val dataset: DataSetPipe = DailySuffixFeatureSource(args("dataset\_path")).read

val outputPath: String = args("output\_path")

val topK: Int = args.int("top\_K", default = 10)

val tweetPairs = DatasetTopKAnalysisJob.getCoocurrenceTweetPairs(dataset)

Execution

.zip(

DatasetTopKAnalysisJob.printTweetPairStatsExec(tweetPairs, topK),

DatasetTopKAnalysisJob.runTweetTopKTweetsOutputExecs(tweetPairs, topK, outputPath),

DatasetTopKAnalysisJob.printPerQueryStatsExec(dataset, topK)

).unit

}

}

}

/\*\* To run:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/tweet\_similarity:dataset\_topk\_analysis-dump \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--main-class com.twitter.simclusters\_v2.scalding.tweet\_similarity.DatasetTopKAnalysisDumpApp -- \

--date 2020-02-01 \

--dataset\_path /user/cassowary/adhoc/training\_data/2020-02-01/train \

--tweets 1223105606757695490 \

--top\_K 100

\* \*/

object DatasetTopKAnalysisDumpApp extends TwitterExecutionApp {

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

def job: Execution[Unit] = Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

implicit val dateRange: DateRange = DateRange.parse(args.list("date"))

val dataset: DataSetPipe = DailySuffixFeatureSource(args("dataset\_path")).read

val tweets = args.list("tweets").map(\_.toLong).toSet

val topK: Int = args.int("top\_K", default = 100)

val tweetPairs = DatasetTopKAnalysisJob.getCoocurrenceTweetPairs(dataset)

if (tweets.isEmpty) {

Execution.from(println("Empty query tweets"))

} else {

val filteredGroupby = tweetPairs

.filter { record => tweets.contains(record.queryTweet) }

.groupBy(\_.queryTweet)

Execution

.zip(

//Top K

DatasetTopKAnalysisJob

.printTweetTopKTweetsBy(filteredGroupby, topK, pair => pair.coengagementCount),

//Bottom K

DatasetTopKAnalysisJob.printTweetTopKTweetsBy(

filteredGroupby,

topK,

pair => pair.coengagementCount,

descending = false)

).unit

}

}

}

}