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

import com.twitter.dal.client.dataset.TimePartitionedDALDataset

import com.twitter.ml.api.util.FDsl.\_

import com.twitter.ml.api.{DataRecord, DataSetPipe}

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DALWrite.D

import com.twitter.scalding\_internal.dalv2.dataset.DALWrite.\_

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

import com.twitter.util.Time

import java.util.Random

/\*\*

\* Collect training data for supervised tweet similarity

\*/

object TrainingDataCollectionUtil {

/\*\*

\* Split dataset into train and test based on time

\* @param dataset: input dataset

\* @param testStartDate: samples before/after testStartDate will be used for training/testing

\* @return (train dataset, test dataset)

\*/

def splitRecordsByTime(

dataset: DataSetPipe,

testStartDate: RichDate

): (DataSetPipe, DataSetPipe) = {

val (leftRecords, rightRecords) = dataset.records.partition { record =>

// record will be in training dataset when both tweets were engaged before testStartDate

(record.getFeatureValue(

TweetSimilarityFeatures.QueryTweetTimestamp) < testStartDate.timestamp) &

(record.getFeatureValue(

TweetSimilarityFeatures.CandidateTweetTimestamp) < testStartDate.timestamp)

}

(

DataSetPipe(leftRecords, dataset.featureContext),

DataSetPipe(rightRecords, dataset.featureContext))

}

/\*\*

\* Split dataset into train and test randomly based on query

\* @param dataset: input dataset

\* @param testRatio: ratio for test

\* @return (train dataset, test dataset)

\*/

def splitRecordsByQuery(dataset: DataSetPipe, testRatio: Double): (DataSetPipe, DataSetPipe) = {

val queryToRand = dataset.records

.map { record => record.getFeatureValue(TweetSimilarityFeatures.QueryTweetId) }

.distinct

.map { queryTweet => queryTweet -> new Random(Time.now.inMilliseconds).nextDouble() }

.forceToDisk

val (trainRecords, testRecords) = dataset.records

.groupBy { record => record.getFeatureValue(TweetSimilarityFeatures.QueryTweetId) }

.join(queryToRand)

.values

.partition {

case (\_, random) => random > testRatio

}

(

DataSetPipe(trainRecords.map { case (record, \_) => record }, dataset.featureContext),

DataSetPipe(testRecords.map { case (record, \_) => record }, dataset.featureContext))

}

/\*\*

\* Get the write exec for train and test datasets

\* @param dataset: input dataset

\* @param testStartDate: samples before/after testStartDate will be used for training/testing

\* @param outputPath: output path for the train/test datasets

\* @return execution of the the writing exec

\*/

def getTrainTestByTimeExec(

dataset: DataSetPipe,

testStartDate: RichDate,

trainDataset: TimePartitionedDALDataset[DataRecord],

testDataset: TimePartitionedDALDataset[DataRecord],

outputPath: String

)(

implicit dateRange: DateRange

): Execution[Unit] = {

val (trainDataSet, testDataSet) = splitRecordsByTime(dataset, testStartDate)

val trainExecution: Execution[Unit] = trainDataSet

.writeDALExecution(trainDataset, D.Daily, D.Suffix(s"$outputPath/train"), D.EBLzo())

val trainStatsExecution: Execution[Unit] =

getStatsExec(trainDataSet, s"$outputPath/train\_stats")

val testExecution: Execution[Unit] = testDataSet

.writeDALExecution(testDataset, D.Daily, D.Suffix(s"$outputPath/test"), D.EBLzo())

val testStatsExecution: Execution[Unit] = getStatsExec(testDataSet, s"$outputPath/test\_stats")

Execution.zip(trainExecution, trainStatsExecution, testExecution, testStatsExecution).unit

}

/\*\*

\* Get the write exec for train and test datasets

\* @param dataset: input dataset

\* @param testRatio: samples before/after testStartDate will be used for training/testing

\* @param outputPath: output path for the train/test datasets

\* @return execution of the the writing exec

\*/

def getTrainTestByQueryExec(

dataset: DataSetPipe,

testRatio: Double,

trainDataset: TimePartitionedDALDataset[DataRecord],

testDataset: TimePartitionedDALDataset[DataRecord],

outputPath: String

)(

implicit dateRange: DateRange

): Execution[Unit] = {

val (trainDataSet, testDataSet) = splitRecordsByQuery(dataset, testRatio)

val trainExecution: Execution[Unit] = trainDataSet

.writeDALExecution(trainDataset, D.Daily, D.Suffix(s"$outputPath/train"), D.EBLzo())

val trainStatsExecution: Execution[Unit] =

getStatsExec(trainDataSet, s"$outputPath/train\_stats")

val testExecution: Execution[Unit] = testDataSet

.writeDALExecution(testDataset, D.Daily, D.Suffix(s"$outputPath/test"), D.EBLzo())

val testStatsExecution: Execution[Unit] = getStatsExec(testDataSet, s"$outputPath/test\_stats")

Execution.zip(trainExecution, trainStatsExecution, testExecution, testStatsExecution).unit

}

/\*\*

\* Get the exec for reporting dataset stats

\* @param dataset: dataset of interest

\* @param outputPath: path for outputting the stats

\* @return exec

\*/

def getStatsExec(dataset: DataSetPipe, outputPath: String): Execution[Unit] = {

dataset.records

.map { rec =>

if (TweetSimilarityFeatures.isCoengaged(rec))

"total\_positive\_records" -> 1L

else

"total\_negative\_records" -> 1L

}

.sumByKey

.shard(1)

.writeExecution(TypedTsv(outputPath))

}

}