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

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.DataSetPipe

import com.twitter.scalding.\_

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding\_internal.dalv2.DAL

import com.twitter.scalding\_internal.dalv2.remote\_access.ExplicitLocation

import com.twitter.scalding\_internal.dalv2.remote\_access.Proc3Atla

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

import com.twitter.simclusters\_v2.hdfs\_sources.TweetSimilarityUnhydratedPairsSource

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

import com.twitter.simclusters\_v2.scalding.tweet\_similarity.TweetPairLabelCollectionUtil.FeaturedTweet

import com.twitter.simclusters\_v2.thriftscala.LabelledTweetPairs

import com.twitter.wtf.scalding.jobs.common.ScheduledExecutionApp

import java.util.TimeZone

/\*\*

\* Hydrate tweet pairs with features

\*/

object TrainingDataCollectionJob {

val LookbackDays = 2 //lookbackdays considered when looking for author information

val testLookbackHours = 2 //hours in test dataset if doing time-based train/test split

val testRatio = 0.1 //ratio for test dataset if doing query-based train/test split

def getHydratedDataPipe(

dateRange: DateRange,

useAuthorFeatures: Boolean,

unhydratedPairs: TypedPipe[LabelledTweetPairs]

)(

implicit timeZone: TimeZone

): DataSetPipe = {

val persistentEmbeddingRecords =

TypedPipe.from(new LogFavBasedPersistentTweetEmbeddingMhExportSource(range = dateRange))

val tweetAuthorPairs =

TweetPairLabelCollectionUtil.getTweetAuthorPairs(dateRange.prepend(Days(LookbackDays)))

val labelledPairs = unhydratedPairs

.map { labelledPair =>

(

FeaturedTweet(

labelledPair.queryFeaturedTweet.tweetId,

labelledPair.queryFeaturedTweet.timestamp,

None,

None),

FeaturedTweet(

labelledPair.candidateFeaturedTweet.tweetId,

labelledPair.candidateFeaturedTweet.timestamp,

None,

None),

labelledPair.label

)

}

TweetPairFeatureHydrationUtil.getDataSetPipeWithFeatures(

labelledPairs,

persistentEmbeddingRecords,

tweetAuthorPairs,

useAuthorFeatures)

}

def getTrainTestExec(

dataSetPipe: DataSetPipe,

splitBy: Option[String],

trainDataset: TimePartitionedDALDataset[DataRecord],

testDataset: TimePartitionedDALDataset[DataRecord],

outputPath: String

)(

implicit timeZone: TimeZone,

dateRange: DateRange

): Execution[Unit] = {

splitBy match {

case Some("time") =>

TrainingDataCollectionUtil.getTrainTestByTimeExec(

dataSetPipe,

dateRange.end - Hours(testLookbackHours),

trainDataset,

testDataset,

outputPath)(dateRange)

case Some("query\_tweet") =>

TrainingDataCollectionUtil.getTrainTestByQueryExec(

dataSetPipe,

testRatio,

trainDataset,

testDataset,

outputPath)(dateRange)

// Default at no splitting

case \_ =>

TrainingDataCollectionUtil.getTrainTestByQueryExec(

dataSetPipe,

0.0,

trainDataset,

testDataset,

outputPath)(dateRange)

}

}

}

/\*\* To run:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/tweet\_similarity:training\_data\_collection-adhoc \

--user cassowary \

--submitter hadoopnest2.atla.twitter.com \

--hadoop-properties "mapreduce.reduce.java.opts=-Xmx8000m mapreduce.reduce.memory.mb=8000 scalding.with.reducers.set.explicitly=true mapreduce.job.reduces=2000 mapreduce.task.timeout=0" \

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

--date 2020-04-15 \

--input\_path /user/cassowary/adhoc/unhydrated\_pairs/2020-04-15\_30min/ \

--output\_path /user/cassowary/adhoc/training\_data/2020-04-15\_30min\_2xneg\_qtweet\_split \

--split\_by query\_tweet

\* \*/

object TrainingDataCollectionAdhocApp extends TwitterExecutionApp {

implicit val timeZone: TimeZone = DateOps.UTC

implicit val dateParser: DateParser = DateParser.default

override def job: Execution[Unit] =

Execution.withId { implicit uniqueId =>

Execution.withArgs { args: Args =>

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

val useAuthorFeatures: Boolean = args.boolean("use\_author\_features")

val inputPath: String = args("input\_path")

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

val splitBy: Option[String] = args.optional("split\_by")

val labelledPairs = TypedPipe

.from(TweetSimilarityUnhydratedPairsSource(inputPath, dateRange))

val dataSetPipe = TrainingDataCollectionJob.getHydratedDataPipe(

dateRange,

useAuthorFeatures,

labelledPairs

)

TrainingDataCollectionJob.getTrainTestExec(

dataSetPipe,

splitBy,

TweetSimilarityTrainDatarecords30MinJavaDataset,

TweetSimilarityTestDatarecords30MinJavaDataset,

outputPath

)

}

}

}

/\*\*

capesospy-v2 update --build\_locally --start\_cron \

training\_data\_collection\_30min src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object TrainingDataCollection30MinScheduledApp extends ScheduledExecutionApp {

private val outputPath: String =

"/user/cassowary/processed/tweet\_similarity/training\_data\_30min"

override def batchIncrement: Duration = Hours(24)

override def firstTime: RichDate = RichDate("2020-03-26")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val useAuthorFeatures: Boolean = args.boolean("use\_author\_features")

val splitBy: Option[String] = args.optional("split\_by")

val unhydratedPairs = DAL

.read(TweetSimilarityUnhydratedPairs30MinScalaDataset, dateRange)

.withRemoteReadPolicy(ExplicitLocation(Proc3Atla))

.toTypedPipe

val dataSetPipe = TrainingDataCollectionJob.getHydratedDataPipe(

dateRange,

useAuthorFeatures,

unhydratedPairs

)

TrainingDataCollectionJob.getTrainTestExec(

dataSetPipe,

splitBy,

TweetSimilarityTrainDatarecords30MinJavaDataset,

TweetSimilarityTestDatarecords30MinJavaDataset,

outputPath)

}

}

/\*\*

capesospy-v2 update --build\_locally --start\_cron \

training\_data\_collection\_120min src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object TrainingDataCollection120MinScheduledApp extends ScheduledExecutionApp {

private val outputPath: String =

"/user/cassowary/processed/tweet\_similarity/training\_data\_120min"

override def batchIncrement: Duration = Hours(24)

override def firstTime: RichDate = RichDate("2020-03-26")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val useAuthorFeatures: Boolean = args.boolean("use\_author\_features")

val splitBy: Option[String] = args.optional("split\_by")

val unhydratedPairs = DAL

.read(TweetSimilarityUnhydratedPairs120MinScalaDataset, dateRange)

.withRemoteReadPolicy(ExplicitLocation(Proc3Atla))

.toTypedPipe

val dataSetPipe = TrainingDataCollectionJob.getHydratedDataPipe(

dateRange,

useAuthorFeatures,

unhydratedPairs

)

TrainingDataCollectionJob.getTrainTestExec(

dataSetPipe,

splitBy,

TweetSimilarityTrainDatarecords120MinJavaDataset,

TweetSimilarityTestDatarecords120MinJavaDataset,

outputPath)

}

}