package com.twitter.simclusters\_v2.scalding.evaluation

import com.twitter.ml.api.constant.SharedFeatures.AUTHOR\_ID

import com.twitter.ml.api.constant.SharedFeatures.TIMESTAMP

import com.twitter.ml.api.constant.SharedFeatures.TWEET\_ID

import com.twitter.ml.api.constant.SharedFeatures.USER\_ID

import com.twitter.ml.api.DailySuffixFeatureSource

import com.twitter.ml.api.DataSetPipe

import com.twitter.ml.api.RichDataRecord

import com.twitter.scalding.\_

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

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

import com.twitter.scalding\_internal.job.analytics\_batch.AnalyticsBatchExecution

import com.twitter.scalding\_internal.job.analytics\_batch.AnalyticsBatchExecutionArgs

import com.twitter.scalding\_internal.job.analytics\_batch.BatchDescription

import com.twitter.scalding\_internal.job.analytics\_batch.BatchFirstTime

import com.twitter.scalding\_internal.job.analytics\_batch.BatchIncrement

import com.twitter.scalding\_internal.job.analytics\_batch.TwitterScheduledExecutionApp

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

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

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

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

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

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

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures.IS\_LINGER\_IMPRESSION

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures.SOURCE\_AUTHOR\_ID

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures.SOURCE\_TWEET\_ID

import com.twitter.timelines.prediction.features.itl.ITLFeatures

import com.twitter.timelines.prediction.features.recap.RecapFeatures

import java.util.TimeZone

/\*\*

\* A scheduled version of the job to parse Timelines data for impressed and engaged tweets.

capesospy-v2 update|create --start\_cron tweet\_evaluation\_timelines\_reference\_batch src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc.yaml

\*/

object ScheduledTimelinesDataExtractionBatch extends TwitterScheduledExecutionApp {

val outputPath = "/user/cassowary/processed/tweet\_evaluation\_reference\_set/timelines"

private val firstTime: String = "2019-03-31"

private implicit val tz: TimeZone = DateOps.UTC

private implicit val parser: DateParser = DateParser.default

private val batchIncrement: Duration = Days(1)

private val execArgs = AnalyticsBatchExecutionArgs(

batchDesc = BatchDescription(this.getClass.getName.replace("$", "")),

firstTime = BatchFirstTime(RichDate(firstTime)),

lastTime = None,

batchIncrement = BatchIncrement(batchIncrement)

)

override def scheduledJob: Execution[Unit] = AnalyticsBatchExecution(execArgs) {

implicit dateRange =>

Execution.withId { implicit uniqueId =>

Execution.withArgs { args =>

val defaultSampleRate = 1.0

val recaps =

TimelinesEngagementDataExtractor.readTimelinesRecapTweets(

recapTweets =

DailySuffixFeatureSource(TimelinesEngagementDataExtractor.RecapTweetHdfsPath).read,

sampleRate = defaultSampleRate

)(dateRange)

val recTweets =

TimelinesEngagementDataExtractor.readTimelinesRecTweets(

recTweets =

DailySuffixFeatureSource(TimelinesEngagementDataExtractor.RecTweetHdfsPath).read,

sampleRate = defaultSampleRate

)(dateRange)

(recaps ++ recTweets).writeDALSnapshotExecution(

TweetEvaluationTimelinesReferenceSetScalaDataset,

D.Daily,

D.Suffix(outputPath),

D.EBLzo(),

dateRange.end

)

}

}

}

}

/\*\*

\* Ad-hoc version of the job to process a subset of the Timeline data, either to catch up with data

\* on a particular day, or to generate human readable data for debugging.

./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/evaluation:tweet\_evaluation\_timelines\_reference\_adhoc

oscar hdfs --screen --user cassowary --bundle tweet\_evaluation\_timelines\_reference\_adhoc \

--tool com.twitter.simclusters\_v2.scalding.evaluation.AdhocTimelinesDataExtraction \

-- --date 2018-11-15 --output\_dir /user/cassowary/your\_ldap/test\_htl\_data/recap --sample\_rate 0.01 \

--recap --rectweet --output\_tsv

\*/

object AdhocTimelinesDataExtraction extends TwitterExecutionApp {

@Override

def job: Execution[Unit] = {

Execution.withArgs { args =>

implicit val dateRange: DateRange =

DateRange.parse(args.list("date"))(DateOps.UTC, DateParser.default)

val outputDir = args("output\_dir")

val readRecTweet = args.boolean("rectweet")

val readRecap = args.boolean("recap")

val sampleRate = args.double("sample\_rate")

val useTsv = args.boolean("output\_tsv")

if (!readRecTweet && !readRecap) {

throw new IllegalArgumentException("Must read at least some data!")

}

val recTweets = if (readRecTweet) {

println("RecTweets are included in the dataset")

TimelinesEngagementDataExtractor.readTimelinesRecTweets(

recTweets =

DailySuffixFeatureSource(TimelinesEngagementDataExtractor.RecTweetHdfsPath).read,

sampleRate = sampleRate)(dateRange)

} else {

TypedPipe.empty

}

val recaps = if (readRecap) {

println("Recaps are included in the dataset")

TimelinesEngagementDataExtractor.readTimelinesRecapTweets(

recapTweets =

DailySuffixFeatureSource(TimelinesEngagementDataExtractor.RecapTweetHdfsPath).read,

sampleRate = sampleRate

)(dateRange)

} else {

TypedPipe.empty

}

val referenceTweets = recaps ++ recTweets

if (useTsv) {

// Write in plain text in tsv format for human readability

referenceTweets

.map(t => (t.targetUserId, t.impressedTweets))

.writeExecution(TypedTsv[(Long, Seq[ReferenceTweet])](outputDir))

} else {

// Write in compact thrift lzo format

referenceTweets

.writeExecution(TimelineDataExtractorFixedPathSource(outputDir))

}

}

}

}

/\*\*

\* Base class to provide functions to parse tweet engagement data from Home Timeline's data.

\* We are mainly interested in 2 tweet data sets from Home Timeline:

\* 1. Recap tweet: Tweets + RTs from user's follow graph. We are interested in out of network RTs.

\* 2. RecTweet: Out of network tweets not from user's follow graph.

\*/

object TimelinesEngagementDataExtractor {

val RecapTweetHdfsPath = "/atla/proc2/user/timelines/processed/suggests/recap/data\_records"

val RecTweetHdfsPath = "/atla/proc2/user/timelines/processed/injections/rectweet/data\_records"

// Timelines name the same feature differently depending on the surface area (ex. recap vs rectweet).

// For each data source we extract the features with different feature names. Detail:

def toRecapTweetLabels(record: RichDataRecord): TweetLabels = {

val isClicked = record.getFeatureValue(RecapFeatures.IS\_CLICKED)

val isFav = record.getFeatureValue(RecapFeatures.IS\_FAVORITED)

val isRT = record.getFeatureValue(RecapFeatures.IS\_RETWEETED)

val isQuoted = record.getFeatureValue(RecapFeatures.IS\_QUOTED)

val isReplied = record.getFeatureValue(RecapFeatures.IS\_REPLIED)

TweetLabels(isClicked, isFav, isRT, isQuoted, isReplied)

}

def toRecTweetLabels(record: RichDataRecord): TweetLabels = {

// Refer to ITLFeatures for more labels

val isClicked = record.getFeatureValue(ITLFeatures.IS\_CLICKED)

val isFav = record.getFeatureValue(ITLFeatures.IS\_FAVORITED)

val isRT = record.getFeatureValue(ITLFeatures.IS\_RETWEETED)

val isQuoted = record.getFeatureValue(ITLFeatures.IS\_QUOTED)

val isReplied = record.getFeatureValue(ITLFeatures.IS\_REPLIED)

TweetLabels(isClicked, isFav, isRT, isQuoted, isReplied)

}

/\*\*

\* Return Recap tweets, which are in-network tweets. Here we only filter for Retweets of tweets

\* that are outside the user's follow graph.

\*/

def readTimelinesRecapTweets(

recapTweets: DataSetPipe,

sampleRate: Double

)(

implicit dateRange: DateRange

): TypedPipe[ReferenceTweets] = {

// recapTweets are in network tweets. We want to discover RTs of OON tweets.

// For Retweets, we check IS\_RETWEET and use SOURCE\_TWEET\_ID, and then check

// PROBABLY\_FROM\_FOLLOWED\_AUTHOR, which filters in network tweet from user's top 1000 follow graph.

recapTweets.richRecords

.sample(sampleRate)

.filter { record =>

val isInDateRange = dateRange.contains(RichDate(record.getFeatureValue(TIMESTAMP).toLong))

val isLingeredImpression = record.getFeatureValue(IS\_LINGER\_IMPRESSION)

val isInNetwork =

record.getFeatureValue(RecapFeatures.PROBABLY\_FROM\_FOLLOWED\_AUTHOR) // approximate

val isRetweet = record.getFeatureValue(RecapFeatures.IS\_RETWEET)

isRetweet && (!isInNetwork) && isInDateRange && isLingeredImpression

}

.flatMap { record =>

for {

userId <- Option(record.getFeatureValue(USER\_ID)).map(\_.toLong)

sourceTweetId <- Option(record.getFeatureValue(SOURCE\_TWEET\_ID)).map(

\_.toLong

) // source tweetId is the RT id

sourceAuthorId <- Option(record.getFeatureValue(SOURCE\_AUTHOR\_ID)).map(\_.toLong)

timestamp <- Option(record.getFeatureValue(TIMESTAMP)).map(\_.toLong)

labels = toRecapTweetLabels(record)

} yield {

(

userId,

Seq(

ReferenceTweet(

sourceTweetId,

sourceAuthorId,

timestamp,

DisplayLocation.TimelinesRecap,

labels))

)

}

}

.sumByKey

.map { case (uid, tweetSeq) => ReferenceTweets(uid, tweetSeq) }

}

/\*\*

\* Return RecTweets, which are out of network tweets served in the Timeline.

\*/

def readTimelinesRecTweets(

recTweets: DataSetPipe,

sampleRate: Double

)(

implicit dateRange: DateRange

): TypedPipe[ReferenceTweets] = {

// recTweets contain strictly out of network injection tweets

recTweets.richRecords

.sample(sampleRate)

.filter { record =>

val isInDateRange = dateRange.contains(RichDate(record.getFeatureValue(TIMESTAMP).toLong))

val isLingeredImpression = record.getFeatureValue(IS\_LINGER\_IMPRESSION)

isInDateRange && isLingeredImpression

}

.flatMap { record =>

for {

userId <- Option(record.getFeatureValue(USER\_ID)).map(\_.toLong)

tweetId <- Option(record.getFeatureValue(TWEET\_ID)).map(\_.toLong)

authorId <- Option(record.getFeatureValue(AUTHOR\_ID)).map(\_.toLong)

timestamp <- Option(record.getFeatureValue(TIMESTAMP)).map(\_.toLong)

labels = toRecTweetLabels(record)

} yield {

(

userId,

Seq(

ReferenceTweet(

tweetId,

authorId,

timestamp,

DisplayLocation.TimelinesRectweet,

labels))

)

}

}

.sumByKey

.map { case (uid, tweetSeq) => ReferenceTweets(uid, tweetSeq) }

}

}