package com.twitter.simclusters\_v2.scalding.embedding.abuse

import com.twitter.ml.api.Feature

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.scalding.Args

import com.twitter.scalding.DateRange

import com.twitter.scalding.Execution

import com.twitter.scalding.UniqueID

import com.twitter.scalding.\_

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

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

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

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

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

import com.twitter.search.common.features.ExternalTweetFeature

import com.twitter.search.common.features.SearchContextFeature

import com.twitter.search.tweet\_ranking.scalding.datasets.TweetEngagementRawTrainingDataDailyJavaDataset

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

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

import com.twitter.simclusters\_v2.scalding.common.matrix.SparseMatrix

import com.twitter.simclusters\_v2.scalding.embedding.abuse.DataSources.NumBlocksP95

import com.twitter.simclusters\_v2.scalding.embedding.abuse.DataSources.getFlockBlocksSparseMatrix

import com.twitter.simclusters\_v2.scalding.embedding.abuse.DataSources.getUserInterestedInSparseMatrix

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil.UserId

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil

import com.twitter.simclusters\_v2.scalding.embedding.common.ExternalDataSources

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

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

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

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

import java.util.TimeZone

object AdhocAbuseSimClusterFeatureKeys {

val AbuseAuthorSearchKey = "abuseAuthorSearch"

val AbuseUserSearchKey = "abuseUserSearch"

val ImpressionUserSearchKey = "impressionUserSearch"

val ImpressionAuthorSearchKey = "impressionAuthorSearch"

val FlockBlocksAuthorKey = "blocksAuthorFlockDataset"

val FlockBlocksUserKey = "blocksUserFlockDataset"

val FavScoresAuthorKey = "favsAuthorFromFavGraph"

val FavScoresUserKey = "favsUserFromFavGraph"

}

/\*\*

\* Adhoc job that is still in development. The job builds features that are meant to be useful for

\* search.

\*

\* Features are built from existing SimCluster representations and the interaction graphs.

\*

\* Example command:

\* scalding remote run \

\* --target src/scala/com/twitter/simclusters\_v2/scalding/embedding/abuse:abuse-adhoc \

\* --main-class com.twitter.simclusters\_v2.scalding.embedding.abuse.AdhocAbuseSimClusterFeaturesScaldingJob \

\* --submitter hadoopnest1.atla.twitter.com --user cassowary \

\* --hadoop-properties "mapreduce.job.user.classpath.first=true" -- \

\* --hdfs --date 2020/11/24 2020/12/14 --partitionName second\_run --dalEnvironment Prod

\*/

object AdhocAbuseSimClusterFeaturesScaldingJob extends AdhocExecutionApp with CassowaryJob {

override def jobName: String = "AdhocAbuseScaldingJob"

import AdhocAbuseSimClusterFeatureKeys.\_

val tweetAuthorFeature = new Feature.Discrete(ExternalTweetFeature.TWEET\_AUTHOR\_ID.getName)

val searcherIdFeature = new Feature.Discrete(SearchContextFeature.SEARCHER\_ID.getName)

val isReportedFeature = new Feature.Binary(ExternalTweetFeature.IS\_REPORTED.getName)

val HalfLifeInDaysForFavScore = 100

private val outputPathThrift: String = EmbeddingUtil.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = false,

modelVersion = ModelVersion.Model20m145kUpdated,

pathSuffix = "abuse\_simcluster\_features"

)

def searchDataRecords(

)(

implicit dateRange: DateRange,

mode: Mode

) = {

DAL

.read(TweetEngagementRawTrainingDataDailyJavaDataset)

.withRemoteReadPolicy(AllowCrossDC)

.toDataSetPipe

.records

}

def abuseInteractionSearchGraph(

)(

implicit dateRange: DateRange,

mode: Mode

): SparseMatrix[UserId, UserId, Double] = {

val abuseMatrixEntries = searchDataRecords()

.flatMap { dataRecord =>

val sDataRecord = SRichDataRecord(dataRecord)

val authorIdOption = sDataRecord.getFeatureValueOpt(tweetAuthorFeature)

val userIdOption = sDataRecord.getFeatureValueOpt(searcherIdFeature)

val isReportedOption = sDataRecord.getFeatureValueOpt(isReportedFeature)

for {

isReported <- isReportedOption if isReported

authorId <- authorIdOption if authorId != 0

userId <- userIdOption if userId != 0

} yield {

(userId: UserId, authorId: UserId, 1.0)

}

}

SparseMatrix.apply[UserId, UserId, Double](abuseMatrixEntries)

}

def impressionInteractionSearchGraph(

)(

implicit dateRange: DateRange,

mode: Mode

): SparseMatrix[UserId, UserId, Double] = {

val impressionMatrixEntries = searchDataRecords

.flatMap { dataRecord =>

val sDataRecord = SRichDataRecord(dataRecord)

val authorIdOption = sDataRecord.getFeatureValueOpt(tweetAuthorFeature)

val userIdOption = sDataRecord.getFeatureValueOpt(searcherIdFeature)

for {

authorId <- authorIdOption if authorId != 0

userId <- userIdOption if userId != 0

} yield {

(userId: UserId, authorId: UserId, 1.0)

}

}

SparseMatrix.apply[UserId, UserId, Double](impressionMatrixEntries)

}

case class SingleSideScores(

unhealthyConsumerClusterScores: TypedPipe[(UserId, SimClustersEmbedding)],

unhealthyAuthorClusterScores: TypedPipe[(UserId, SimClustersEmbedding)],

healthyConsumerClusterScores: TypedPipe[(UserId, SimClustersEmbedding)],

healthyAuthorClusterScores: TypedPipe[(UserId, SimClustersEmbedding)])

def buildSearchAbuseScores(

normalizedSimClusterMatrix: SparseMatrix[UserId, ClusterId, Double],

unhealthyGraph: SparseMatrix[UserId, UserId, Double],

healthyGraph: SparseMatrix[UserId, UserId, Double]

): SingleSideScores = {

SingleSideScores(

unhealthyConsumerClusterScores = SingleSideInteractionTransformation

.clusterScoresFromGraphs(normalizedSimClusterMatrix, unhealthyGraph),

unhealthyAuthorClusterScores = SingleSideInteractionTransformation

.clusterScoresFromGraphs(normalizedSimClusterMatrix, unhealthyGraph.transpose),

healthyConsumerClusterScores = SingleSideInteractionTransformation

.clusterScoresFromGraphs(normalizedSimClusterMatrix, healthyGraph),

healthyAuthorClusterScores = SingleSideInteractionTransformation

.clusterScoresFromGraphs(normalizedSimClusterMatrix, healthyGraph.transpose)

)

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

Execution.getMode.flatMap { implicit mode =>

val normalizedSimClusterMatrix = getUserInterestedInSparseMatrix.rowL2Normalize

val abuseSearchGraph = abuseInteractionSearchGraph()

val impressionSearchGraph = impressionInteractionSearchGraph()

val searchAbuseScores = buildSearchAbuseScores(

normalizedSimClusterMatrix,

unhealthyGraph = abuseSearchGraph,

healthyGraph = impressionSearchGraph)

// Step 2a: Read FlockBlocks for unhealthy interactions and user-user-fav for healthy interactions

val flockBlocksSparseGraph =

getFlockBlocksSparseMatrix(NumBlocksP95, dateRange.prepend(Years(1)))

val favSparseGraph = SparseMatrix.apply[UserId, UserId, Double](

ExternalDataSources.getFavEdges(HalfLifeInDaysForFavScore))

val blocksAbuseScores = buildSearchAbuseScores(

normalizedSimClusterMatrix,

unhealthyGraph = flockBlocksSparseGraph,

healthyGraph = favSparseGraph

)

// Step 3. Combine all scores from different sources for users

val pairedScores = SingleSideInteractionTransformation.pairScores(

Map(

// User cluster scores built from the search abuse reports graph

AbuseUserSearchKey -> searchAbuseScores.unhealthyConsumerClusterScores,

// Author cluster scores built from the search abuse reports graph

AbuseAuthorSearchKey -> searchAbuseScores.unhealthyAuthorClusterScores,

// User cluster scores built from the search impression graph

ImpressionUserSearchKey -> searchAbuseScores.healthyConsumerClusterScores,

// Author cluster scores built from the search impression graph

ImpressionAuthorSearchKey -> searchAbuseScores.healthyAuthorClusterScores,

// User cluster scores built from flock blocks graph

FlockBlocksUserKey -> blocksAbuseScores.unhealthyConsumerClusterScores,

// Author cluster scores built from the flock blocks graph

FlockBlocksAuthorKey -> blocksAbuseScores.unhealthyAuthorClusterScores,

// User cluster scores built from the user-user fav graph

FavScoresUserKey -> blocksAbuseScores.healthyConsumerClusterScores,

// Author cluster scores built from the user-user fav graph

FavScoresAuthorKey -> blocksAbuseScores.healthyAuthorClusterScores

)

)

pairedScores.writeDALSnapshotExecution(

AdhocAbuseSimclusterFeaturesScalaDataset,

D.Daily,

D.Suffix(outputPathThrift),

D.Parquet,

dateRange.`end`,

partitions = Set(D.Partition("partition", args("partitionName"), D.PartitionType.String))

)

}

}

}