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

import com.google.common.annotations.VisibleForTesting

import com.twitter.scalding.\_

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

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

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

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

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

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

/\*\*

\* Logic for building a SimCluster represenation of interaction signals. The purpose of this job is

\* to model negative behavior (like abuse and blocks).

\*

\* This is a "SingleSide", because we are only considering one side of the interaction graph to

\* build these features. So for instance we would keep track of which simclusters are most likely to

\* get reported for abuse regardless of who reported it. Another job will be responsible for

\* building the simcluster to simcluster interaction matrix as described in the doc.

\*/

object SingleSideInteractionTransformation {

/\*\*

\* Compute a score for every SimCluster. The SimCluster score is a count of the number of

\* interactions for each SimCluster. For a user that has many SimClusters, we distribute each of

\* their interactions across all of these SimClusters.

\*

\* @param normalizedUserSimClusters Sparse matrix of User-SimCluster scores. Users are rows and

\* SimClusters are columns. This should already by L2normalized.

\* It is important that we normalize so that each interaction

\* only adds 1 to the counts.

\* @param interactionGraph Graph of interactions. Rows are the users, columns are not used.

\* All values in this graph are assumed to be positive; they are the number of

\* interactions.

\*

\* @return SingleSideClusterFeatures for each SimCluster that has user with an interaction.

\*/

def computeClusterFeatures(

normalizedUserSimClusters: SparseMatrix[UserId, ClusterId, Double],

interactionGraph: SparseMatrix[UserId, \_, Double]

): TypedPipe[SimClusterWithScore] = {

val numReportsForUserEntries = interactionGraph.rowL1Norms.map {

// turn into a vector where we use 1 as the column key for every entry.

case (user, count) => (user, 1, count)

}

val numReportsForUser = SparseMatrix[UserId, Int, Double](numReportsForUserEntries)

normalizedUserSimClusters.transpose

.multiplySparseMatrix(numReportsForUser)

.toTypedPipe

.map {

case (clusterId, \_, clusterScore: Double) =>

SimClusterWithScore(clusterId, clusterScore)

}

}

/\*\*

\* Given that we have the score for each SimCluster and the user's SimClusters, create a

\* representation of the user so that the new SimCluster scores are an estimate of the

\* interactions for this user.

\*

\* @param normalizedUserSimClusters sparse matrix of User-SimCluster scores. Users are rows and

\* SimClusters are columns. This should already be L2 normalized.

\* @param simClusterFeatures For each SimCluster, a score associated with this interaction type.

\*

\* @return SingleSideAbuseFeatures for each user the SimClusters and scores for this

\*/

@VisibleForTesting

private[abuse] def computeUserFeaturesFromClusters(

normalizedUserSimClusters: SparseMatrix[UserId, ClusterId, Double],

simClusterFeatures: TypedPipe[SimClusterWithScore]

): TypedPipe[(UserId, SimClustersEmbedding)] = {

normalizedUserSimClusters.toTypedPipe

.map {

case (userId, clusterId, score) =>

(clusterId, (userId, score))

}

.group

// There are at most 140k SimClusters. They should fit in memory

.hashJoin(simClusterFeatures.groupBy(\_.clusterId))

.map {

case (\_, ((userId, score), singleSideClusterFeatures)) =>

(

userId,

List(

SimClusterWithScore(

singleSideClusterFeatures.clusterId,

singleSideClusterFeatures.score \* score))

)

}

.sumByKey

.mapValues(SimClustersEmbedding.apply)

}

/\*\*

\* Combines all the different SimClustersEmbedding for a user into one

\* AdhocSingleSideClusterScores.

\*

\* @param interactionMap The key is an identifier for the embedding type. The typed pipe will have

\* embeddings of only for that type of embedding.

\* @return Typed pipe with one AdhocSingleSideClusterScores per user.

\*/

def pairScores(

interactionMap: Map[String, TypedPipe[(UserId, SimClustersEmbedding)]]

): TypedPipe[AdhocSingleSideClusterScores] = {

val combinedInteractions = interactionMap

.map {

case (interactionTypeName, userInteractionFeatures) =>

userInteractionFeatures.map {

case (userId, simClustersEmbedding) =>

(userId, List((interactionTypeName, simClustersEmbedding)))

}

}

.reduce[TypedPipe[(UserId, List[(String, SimClustersEmbedding)])]] {

case (list1, list2) =>

list1 ++ list2

}

.group

.sumByKey

combinedInteractions.toTypedPipe

.map {

case (userId, interactionFeatureList) =>

AdhocSingleSideClusterScores(

userId,

interactionFeatureList.toMap

)

}

}

/\*\*

\* Given the SimCluster and interaction graph get the user representation for this interaction.

\* See the documentation of the underlying methods for more details

\*

\* @param normalizedUserSimClusters sparse matrix of User-SimCluster scores. Users are rows and

\* SimClusters are columns. This should already by L2normalized.

\* @param interactionGraph Graph of interactions. Rows are the users, columns are not used.

\* All values in this graph are assumed to be positive; they are the number of

\* interactions.

\*

\* @return SimClustersEmbedding for all users in the give SimCluster graphs

\*/

def clusterScoresFromGraphs(

normalizedUserSimClusters: SparseMatrix[UserId, ClusterId, Double],

interactionGraph: SparseMatrix[UserId, \_, Double]

): TypedPipe[(UserId, SimClustersEmbedding)] = {

val clusterFeatures = computeClusterFeatures(normalizedUserSimClusters, interactionGraph)

computeUserFeaturesFromClusters(normalizedUserSimClusters, clusterFeatures)

}

}