package com.twitter.timelines.prediction.features.simcluster

import com.twitter.dal.personal\_data.thriftjava.PersonalDataType.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.ml.api.{Feature, FeatureContext}

import com.twitter.ml.api.Feature.{Continuous, SparseBinary, SparseContinuous}

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.conversion.\_

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.TypedAggregateGroup

import com.twitter.timelines.suggests.common.record.thriftscala.SuggestionRecord

import scala.collection.JavaConverters.\_

class SimclusterTweetFeatures(statsReceiver: StatsReceiver) extends CombineCountsBase {

import SimclusterTweetFeatures.\_

private[this] val scopedStatsReceiver = statsReceiver.scope(getClass.getSimpleName)

private[this] val invalidSimclusterModelVersion = scopedStatsReceiver

.counter("invalidSimclusterModelVersion")

private[this] val getFeaturesFromOverlappingSimclusterIdsCount = scopedStatsReceiver

.counter("getFeaturesFromOverlappingSimclusterIdsCount")

private[this] val emptySimclusterMaps = scopedStatsReceiver

.counter("emptySimclusterMaps")

private[this] val nonOverlappingSimclusterMaps = scopedStatsReceiver

.counter("nonOverlappingSimclusterMaps")

// Parameters required by CombineCountsBase

override val topK: Int = 5

override val hardLimit: Option[Int] = None

override val precomputedCountFeatures: Seq[Feature[\_]] = Seq(

SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_TWEET\_SCORE,

SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_COMBINED\_SCORE

)

private def getFeaturesFromOverlappingSimclusterIds(

userSimclustersInterestedInMap: Map[String, Double],

tweetSimclustersTopKMap: Map[String, Double]

): Map[Feature[\_], List[Double]] = {

getFeaturesFromOverlappingSimclusterIdsCount.incr

if (userSimclustersInterestedInMap.isEmpty || tweetSimclustersTopKMap.isEmpty) {

emptySimclusterMaps.incr

Map.empty

} else {

val overlappingSimclusterIds =

userSimclustersInterestedInMap.keySet intersect tweetSimclustersTopKMap.keySet

if (overlappingSimclusterIds.isEmpty) {

nonOverlappingSimclusterMaps.incr

Map.empty

} else {

val (combinedScores, tweetScores) = overlappingSimclusterIds.map { id =>

val tweetScore = tweetSimclustersTopKMap.getOrElse(id, 0.0)

val combinedScore = userSimclustersInterestedInMap.getOrElse(id, 0.0) \* tweetScore

(combinedScore, tweetScore)

}.unzip

Map(

SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_COMBINED\_SCORE -> combinedScores.toList,

SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_TWEET\_SCORE -> tweetScores.toList

)

}

}

}

def getCountFeaturesValuesMap(

suggestionRecord: SuggestionRecord,

simclustersTweetTopKMap: Map[String, Double]

): Map[Feature[\_], List[Double]] = {

val userSimclustersInterestedInMap = formatUserSimclustersInterestedIn(suggestionRecord)

val tweetSimclustersTopKMap = formatTweetSimclustersTopK(simclustersTweetTopKMap)

getFeaturesFromOverlappingSimclusterIds(userSimclustersInterestedInMap, tweetSimclustersTopKMap)

}

def filterByModelVersion(

simclustersMapOpt: Option[Map[String, Double]]

): Option[Map[String, Double]] = {

simclustersMapOpt.flatMap { simclustersMap =>

val filteredSimclustersMap = simclustersMap.filter {

case (clusterId, score) =>

// The clusterId format is ModelVersion.IntegerClusterId.ScoreType as specified at

// com.twitter.ml.featurestore.catalog.features.recommendations.SimClustersV2TweetTopClusters

clusterId.contains(SimclusterFeatures.SIMCLUSTER\_MODEL\_VERSION)

}

// The assumption is that the simclustersMap will contain clusterIds with the same modelVersion.

// We maintain this counter to make sure that the hardcoded modelVersion we are using is correct.

if (simclustersMap.size > filteredSimclustersMap.size) {

invalidSimclusterModelVersion.incr

}

if (filteredSimclustersMap.nonEmpty) Some(filteredSimclustersMap) else None

}

}

val allFeatures: Seq[Feature[\_]] = outputFeaturesPostMerge.toSeq ++ Seq(

SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_IDS,

SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_SCORES)

val featureContext = new FeatureContext(allFeatures: \_\*)

}

object SimclusterTweetFeatures {

val SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_IDS = new SparseBinary(

s"${SimclusterFeatures.prefix}.tweet\_topk\_cluster\_ids",

Set(InferredInterests).asJava

)

val SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_SCORES = new SparseContinuous(

s"${SimclusterFeatures.prefix}.tweet\_topk\_cluster\_scores",

Set(EngagementScore, InferredInterests).asJava

)

val SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_ID =

TypedAggregateGroup.sparseFeature(SIMCLUSTER\_TWEET\_TOPK\_CLUSTER\_IDS)

val SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_TWEET\_SCORE = new Continuous(

s"${SimclusterFeatures.prefix}.tweet\_topk\_sort\_by\_tweet\_score",

Set(EngagementScore, InferredInterests).asJava

)

val SIMCLUSTER\_TWEET\_TOPK\_SORT\_BY\_COMBINED\_SCORE = new Continuous(

s"${SimclusterFeatures.prefix}.tweet\_topk\_sort\_by\_combined\_score",

Set(EngagementScore, InferredInterests).asJava

)

def formatUserSimclustersInterestedIn(suggestionRecord: SuggestionRecord): Map[String, Double] = {

suggestionRecord.userSimclustersInterestedIn

.map { clustersUserIsInterestedIn =>

if (clustersUserIsInterestedIn.knownForModelVersion == SimclusterFeatures.SIMCLUSTER\_MODEL\_VERSION) {

clustersUserIsInterestedIn.clusterIdToScores.collect {

case (clusterId, scores) if scores.favScore.isDefined =>

(clusterId.toString, scores.favScore.get)

}

} else Map.empty[String, Double]

}.getOrElse(Map.empty[String, Double])

.toMap

}

def formatTweetSimclustersTopK(

simclustersTweetTopKMap: Map[String, Double]

): Map[String, Double] = {

simclustersTweetTopKMap.collect {

case (clusterId, score) =>

// The clusterId format is <ModelVersion.IntegerClusterId.ScoreType> as specified at

// com.twitter.ml.featurestore.catalog.features.recommendations.SimClustersV2TweetTopClusters

// and we want to extract the IntegerClusterId.

// The split function takes a regex; therefore, we need to escape . and we also need to escape

// \ since they are both special characters. Hence, the double \\.

val clusterIdSplit = clusterId.split("\\.")

val integerClusterId = clusterIdSplit(1) // The IntegerClusterId is at position 1.

(integerClusterId, score)

}

}

}