package com.twitter.simclusters\_v2.candidate\_source

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

object ClusterRanker extends Enumeration {

val RankByNormalizedFavScore: ClusterRanker.Value = Value

val RankByFavScore: ClusterRanker.Value = Value

val RankByFollowScore: ClusterRanker.Value = Value

val RankByLogFavScore: ClusterRanker.Value = Value

val RankByNormalizedLogFavScore: ClusterRanker.Value = Value

/\*\*

\* Given a map of clusters, sort out the top scoring clusters by a ranking scheme

\* provided by the caller

\*/

def getTopKClustersByScore(

clustersWithScores: Map[Int, UserToInterestedInClusterScores],

rankByScore: ClusterRanker.Value,

topK: Int

): Map[Int, Double] = {

val rankedClustersWithScores = clustersWithScores.map {

case (clusterId, score) =>

rankByScore match {

case ClusterRanker.RankByFavScore =>

(clusterId, (score.favScore.getOrElse(0.0), score.followScore.getOrElse(0.0)))

case ClusterRanker.RankByFollowScore =>

(clusterId, (score.followScore.getOrElse(0.0), score.favScore.getOrElse(0.0)))

case ClusterRanker.RankByLogFavScore =>

(clusterId, (score.logFavScore.getOrElse(0.0), score.followScore.getOrElse(0.0)))

case ClusterRanker.RankByNormalizedLogFavScore =>

(

clusterId,

(

score.logFavScoreClusterNormalizedOnly.getOrElse(0.0),

score.followScore.getOrElse(0.0)))

case ClusterRanker.RankByNormalizedFavScore =>

(

clusterId,

(

score.favScoreProducerNormalizedOnly.getOrElse(0.0),

score.followScore.getOrElse(0.0)))

case \_ =>

(

clusterId,

(

score.favScoreProducerNormalizedOnly.getOrElse(0.0),

score.followScore.getOrElse(0.0)))

}

}

rankedClustersWithScores.toSeq

.sortBy(\_.\_2) // sort in ascending order

.takeRight(topK)

.map { case (clusterId, scores) => clusterId -> math.max(scores.\_1, 1e-4) }

.toMap

}

}