package com.twitter.follow\_recommendations.common.rankers.weighted\_candidate\_source\_ranker

import com.twitter.follow\_recommendations.common.utils.RandomUtil

import com.twitter.follow\_recommendations.common.utils.MergeUtil

import com.twitter.follow\_recommendations.common.utils.Weighted

import com.twitter.follow\_recommendations.common.rankers.weighted\_candidate\_source\_ranker.WeightMethod.\_

import scala.util.Random

/\*\*

\* This ranker selects the next candidate source to select a candidate from. It supports

\* two kinds of algorithm, WeightedRandomSampling or WeightedRoundRobin. WeightedRandomSampling

\* pick the next candidate source randomly, WeightedRoundRobin picked the next candidate source

\* sequentially based on the weight of the candidate source. It is default to WeightedRandomSampling

\* if no weight method is provided.

\*

\* Example usage of this class:

\*

\* When use WeightedRandomSampling:

\* Input candidate sources and their weights are: {{CS1: 3}, {CS2: 2}, {CS3: 5}}

\* Ranked candidates sequence is not determined because of random sampling.

\* One possible output candidate sequence is: (CS1\_candidate1, CS2\_candidate1, CS2\_candidate2,

\* CS3\_candidate1, CS3\_candidates2, CS3\_candidate3, CS1\_candidate2, CS1\_candidate3,

\* CS3\_candidate4, CS3\_candidate5, CS1\_candidate4, CS1\_candidate5, CS2\_candidate6, CS2\_candidate3,...)

\*

\* When use WeightedRoundRobin:

\* Input candidate sources and their weights are: {{CS1: 3}, {CS2: 2}, {CS3: 5}}

\* Output candidate sequence is: (CS1\_candidate1, CS1\_candidate2, CS1\_candidate3,

\* CS2\_candidate1, CS2\_candidates2, CS3\_candidate1, CS3\_candidate2, CS3\_candidate3,

\* CS3\_candidate4, CS3\_candidate5, CS1\_candidate4, CS1\_candidate5, CS1\_candidate6, CS2\_candidate3,...)

\*

\* Note: CS1\_candidate1 means the first candidate in CS1 candidate source.

\*

\* @tparam A candidate source type

\* @tparam Rec Recommendation type

\* @param candidateSourceWeights relative weights for different candidate sources

\*/

class WeightedCandidateSourceBaseRanker[A, Rec](

candidateSourceWeights: Map[A, Double],

weightMethod: WeightMethod = WeightedRandomSampling,

randomSeed: Option[Long]) {

/\*\*

\* Creates a iterator over algorithms and calls next to return a Stream of candidates

\*

\*

\* @param candidateSources the set of candidate sources that are being sampled

\* @param candidateSourceWeights map of candidate source to weight

\* @param candidates the map of candidate source to the iterator of its results

\* @param weightMethod a enum to indict which weight method to use. Two values are supported

\* currently. When WeightedRandomSampling is set, the next candidate is picked from a candidate

\* source that is randomly chosen. When WeightedRoundRobin is set, the next candidate is picked

\* from current candidate source until the number of candidates reaches to the assigned weight of

\* the candidate source. The next call of this function will return a candidate from the next

\* candidate source which is after previous candidate source based on the order input

\* candidate source sequence.

\* @return stream of candidates

\*/

def stream(

candidateSources: Set[A],

candidateSourceWeights: Map[A, Double],

candidates: Map[A, Iterator[Rec]],

weightMethod: WeightMethod = WeightedRandomSampling,

random: Option[Random] = None

): Stream[Rec] = {

val weightedCandidateSource: Weighted[A] = new Weighted[A] {

override def apply(a: A): Double = candidateSourceWeights.getOrElse(a, 0)

}

/\*\*

\* Generates a stream of candidates.

\*

\* @param candidateSourceIter an iterator over candidate sources returned by the sampling procedure

\* @return stream of candidates

\*/

def next(candidateSourceIter: Iterator[A]): Stream[Rec] = {

val source = candidateSourceIter.next()

val it = candidates(source)

if (it.hasNext) {

val currCand = it.next()

currCand #:: next(candidateSourceIter)

} else {

assert(candidateSources.contains(source), "Selected source is not in candidate sources")

// Remove the depleted candidate source and re-sample

stream(candidateSources - source, candidateSourceWeights, candidates, weightMethod, random)

}

}

if (candidateSources.isEmpty)

Stream.empty

else {

val candidateSourceSeq = candidateSources.toSeq

val candidateSourceIter =

if (weightMethod == WeightMethod.WeightedRoundRobin) {

MergeUtil.weightedRoundRobin(candidateSourceSeq)(weightedCandidateSource).iterator

} else {

//default to weighted random sampling if no other weight method is provided

RandomUtil

.weightedRandomSamplingWithReplacement(

candidateSourceSeq,

random

)(weightedCandidateSource).iterator

}

next(candidateSourceIter)

}

}

def apply(input: Map[A, TraversableOnce[Rec]]): Stream[Rec] = {

stream(

input.keySet,

candidateSourceWeights,

input.map {

case (k, v) => k -> v.toIterator

}, // cannot do mapValues here, as that only returns a view

weightMethod,

randomSeed.map(new Random(\_))

)

}

}