package com.twitter.cr\_mixer.model

import com.twitter.contentrecommender.thriftscala.TweetInfo

import com.twitter.cr\_mixer.thriftscala.LineItemInfo

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

sealed trait Candidate {

val tweetId: TweetId

override def hashCode: Int = tweetId.toInt

}

case class TweetWithCandidateGenerationInfo(

tweetId: TweetId,

candidateGenerationInfo: CandidateGenerationInfo)

extends Candidate {

def getSimilarityScore: Double =

candidateGenerationInfo.similarityEngineInfo.score.getOrElse(0.0)

}

case class InitialCandidate(

tweetId: TweetId,

tweetInfo: TweetInfo,

candidateGenerationInfo: CandidateGenerationInfo)

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

candidateGenerationInfo.similarityEngineInfo.score.getOrElse(0.0)

/\*\*

\* The same candidate can be generated by multiple algorithms.

\* During blending, candidate deduping happens. In order to retain the candidateGenerationInfo

\* from different algorithms, we attach them to a list of potentialReasons.

\*/

def toBlendedCandidate(

potentialReasons: Seq[CandidateGenerationInfo],

): BlendedCandidate = {

BlendedCandidate(

tweetId,

tweetInfo,

candidateGenerationInfo,

potentialReasons,

)

}

// for experimental purposes only when bypassing interleave / ranking

def toRankedCandidate(): RankedCandidate = {

RankedCandidate(

tweetId,

tweetInfo,

0.0, // prediction score is default to 0.0 to help differentiate that it is a no-op

candidateGenerationInfo,

Seq(candidateGenerationInfo)

)

}

}

case class InitialAdsCandidate(

tweetId: TweetId,

lineItemInfo: Seq[LineItemInfo],

candidateGenerationInfo: CandidateGenerationInfo)

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

candidateGenerationInfo.similarityEngineInfo.score.getOrElse(0.0)

/\*\*

\* The same candidate can be generated by multiple algorithms.

\* During blending, candidate deduping happens. In order to retain the candidateGenerationInfo

\* from different algorithms, we attach them to a list of potentialReasons.

\*/

def toBlendedAdsCandidate(

potentialReasons: Seq[CandidateGenerationInfo],

): BlendedAdsCandidate = {

BlendedAdsCandidate(

tweetId,

lineItemInfo,

candidateGenerationInfo,

potentialReasons,

)

}

// for experimental purposes only when bypassing interleave / ranking

def toRankedAdsCandidate(): RankedAdsCandidate = {

RankedAdsCandidate(

tweetId,

lineItemInfo,

0.0, // prediction score is default to 0.0 to help differentiate that it is a no-op

candidateGenerationInfo,

Seq(candidateGenerationInfo)

)

}

}

case class BlendedCandidate(

tweetId: TweetId,

tweetInfo: TweetInfo,

reasonChosen: CandidateGenerationInfo,

potentialReasons: Seq[CandidateGenerationInfo])

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

reasonChosen.similarityEngineInfo.score.getOrElse(0.0)

assert(potentialReasons.contains(reasonChosen))

def toRankedCandidate(predictionScore: Double): RankedCandidate = {

RankedCandidate(

tweetId,

tweetInfo,

predictionScore,

reasonChosen,

potentialReasons

)

}

}

case class BlendedAdsCandidate(

tweetId: TweetId,

lineItemInfo: Seq[LineItemInfo],

reasonChosen: CandidateGenerationInfo,

potentialReasons: Seq[CandidateGenerationInfo])

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

reasonChosen.similarityEngineInfo.score.getOrElse(0.0)

assert(potentialReasons.contains(reasonChosen))

def toRankedAdsCandidate(predictionScore: Double): RankedAdsCandidate = {

RankedAdsCandidate(

tweetId,

lineItemInfo,

predictionScore,

reasonChosen,

potentialReasons

)

}

}

case class RankedCandidate(

tweetId: TweetId,

tweetInfo: TweetInfo,

predictionScore: Double,

reasonChosen: CandidateGenerationInfo,

potentialReasons: Seq[CandidateGenerationInfo])

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

reasonChosen.similarityEngineInfo.score.getOrElse(0.0)

assert(potentialReasons.contains(reasonChosen))

}

case class RankedAdsCandidate(

tweetId: TweetId,

lineItemInfo: Seq[LineItemInfo],

predictionScore: Double,

reasonChosen: CandidateGenerationInfo,

potentialReasons: Seq[CandidateGenerationInfo])

extends Candidate {

/\*\* \*

\* Get the Similarity Score of a Tweet from its CG Info. For instance,

\* If it is from a UnifiedTweetBasedSimilarityEngine, the score will be the weighted combined score

\* And if it is from a SimClustersANNSimilarityEngine, the score will be the SANN score

\*/

def getSimilarityScore: Double =

reasonChosen.similarityEngineInfo.score.getOrElse(0.0)

assert(potentialReasons.contains(reasonChosen))

}

case class TripTweetWithScore(tweetId: TweetId, score: Double) extends Candidate