package com.twitter.follow\_recommendations.common.models

import com.twitter.hermit.ml.models.Feature

import com.twitter.hermit.model.Algorithm

import com.twitter.hermit.model.Algorithm.Algorithm

import com.twitter.product\_mixer.core.model.common.identifier.CandidateSourceIdentifier

/\*\*

\* Used to keep track of a candidate's source not so much as a feature but for filtering candidate

\* from specific sources (eg. GizmoduckPredicate)

\*/

trait HasUserCandidateSourceDetails { candidateUser: CandidateUser =>

def userCandidateSourceDetails: Option[UserCandidateSourceDetails]

def getAlgorithm: Algorithm = {

val algorithm = for {

details <- userCandidateSourceDetails

identifier <- details.primaryCandidateSource

algorithm <- Algorithm.withNameOpt(identifier.name)

} yield algorithm

algorithm.getOrElse(throw new Exception("Algorithm missing on candidate user!"))

}

def getAllAlgorithms: Seq[Algorithm] = {

getCandidateSources.keys

.flatMap(identifier => Algorithm.withNameOpt(identifier.name)).toSeq

}

def getAddressBookMetadata: Option[AddressBookMetadata] = {

userCandidateSourceDetails.flatMap(\_.addressBookMetadata)

}

def getCandidateSources: Map[CandidateSourceIdentifier, Option[Double]] = {

userCandidateSourceDetails.map(\_.candidateSourceScores).getOrElse(Map.empty)

}

def getCandidateRanks: Map[CandidateSourceIdentifier, Int] = {

userCandidateSourceDetails.map(\_.candidateSourceRanks).getOrElse(Map.empty)

}

def getCandidateFeatures: Map[CandidateSourceIdentifier, Seq[Feature]] = {

userCandidateSourceDetails.map(\_.candidateSourceFeatures).getOrElse(Map.empty)

}

def getPrimaryCandidateSource: Option[CandidateSourceIdentifier] = {

userCandidateSourceDetails.flatMap(\_.primaryCandidateSource)

}

def withCandidateSource(source: CandidateSourceIdentifier): CandidateUser = {

withCandidateSourceAndScore(source, candidateUser.score)

}

def withCandidateSourceAndScore(

source: CandidateSourceIdentifier,

score: Option[Double]

): CandidateUser = {

withCandidateSourceScoreAndFeatures(source, score, Nil)

}

def withCandidateSourceAndFeatures(

source: CandidateSourceIdentifier,

features: Seq[Feature]

): CandidateUser = {

withCandidateSourceScoreAndFeatures(source, candidateUser.score, features)

}

def withCandidateSourceScoreAndFeatures(

source: CandidateSourceIdentifier,

score: Option[Double],

features: Seq[Feature]

): CandidateUser = {

val candidateSourceDetails =

candidateUser.userCandidateSourceDetails

.map { details =>

details.copy(

primaryCandidateSource = Some(source),

candidateSourceScores = details.candidateSourceScores + (source -> score),

candidateSourceFeatures = details.candidateSourceFeatures + (source -> features)

)

}.getOrElse(

UserCandidateSourceDetails(

Some(source),

Map(source -> score),

Map.empty,

None,

Map(source -> features)))

candidateUser.copy(

userCandidateSourceDetails = Some(candidateSourceDetails)

)

}

def addCandidateSourceScoresMap(

scoreMap: Map[CandidateSourceIdentifier, Option[Double]]

): CandidateUser = {

val candidateSourceDetails = candidateUser.userCandidateSourceDetails

.map { details =>

details.copy(candidateSourceScores = details.candidateSourceScores ++ scoreMap)

}.getOrElse(UserCandidateSourceDetails(scoreMap.keys.headOption, scoreMap, Map.empty, None))

candidateUser.copy(

userCandidateSourceDetails = Some(candidateSourceDetails)

)

}

def addCandidateSourceRanksMap(

rankMap: Map[CandidateSourceIdentifier, Int]

): CandidateUser = {

val candidateSourceDetails = candidateUser.userCandidateSourceDetails

.map { details =>

details.copy(candidateSourceRanks = details.candidateSourceRanks ++ rankMap)

}.getOrElse(UserCandidateSourceDetails(rankMap.keys.headOption, Map.empty, rankMap, None))

candidateUser.copy(

userCandidateSourceDetails = Some(candidateSourceDetails)

)

}

def addInfoPerRankingStage(

rankingStage: String,

scores: Option[Scores],

rank: Int

): CandidateUser = {

val scoresOpt: Option[Scores] = scores.orElse(candidateUser.scores)

val originalInfoPerRankingStage =

candidateUser.infoPerRankingStage.getOrElse(Map[String, RankingInfo]())

candidateUser.copy(

infoPerRankingStage =

Some(originalInfoPerRankingStage + (rankingStage -> RankingInfo(scoresOpt, Some(rank))))

)

}

def addAddressBookMetadataIfAvailable(

candidateSources: Seq[CandidateSourceIdentifier]

): CandidateUser = {

val addressBookMetadata = AddressBookMetadata(

inForwardPhoneBook =

candidateSources.contains(AddressBookMetadata.ForwardPhoneBookCandidateSource),

inReversePhoneBook =

candidateSources.contains(AddressBookMetadata.ReversePhoneBookCandidateSource),

inForwardEmailBook =

candidateSources.contains(AddressBookMetadata.ForwardEmailBookCandidateSource),

inReverseEmailBook =

candidateSources.contains(AddressBookMetadata.ReverseEmailBookCandidateSource)

)

val newCandidateSourceDetails = candidateUser.userCandidateSourceDetails

.map { details =>

details.copy(addressBookMetadata = Some(addressBookMetadata))

}.getOrElse(

UserCandidateSourceDetails(

None,

Map.empty,

Map.empty,

Some(addressBookMetadata),

Map.empty))

candidateUser.copy(

userCandidateSourceDetails = Some(newCandidateSourceDetails)

)

}

}