package com.twitter.cr\_mixer.candidate\_generation

import com.twitter.cr\_mixer.model.CandidateGenerationInfo

import com.twitter.cr\_mixer.model.InitialAdsCandidate

import com.twitter.cr\_mixer.model.ModelConfig

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.cr\_mixer.model.SimilarityEngineInfo

import com.twitter.cr\_mixer.model.SourceInfo

import com.twitter.cr\_mixer.model.TweetWithCandidateGenerationInfo

import com.twitter.cr\_mixer.model.TweetWithScore

import com.twitter.cr\_mixer.param.ConsumersBasedUserAdGraphParams

import com.twitter.cr\_mixer.param.ConsumerBasedWalsParams

import com.twitter.cr\_mixer.param.ConsumerEmbeddingBasedCandidateGenerationParams

import com.twitter.cr\_mixer.param.GlobalParams

import com.twitter.cr\_mixer.param.InterestedInParams

import com.twitter.cr\_mixer.param.ProducerBasedCandidateGenerationParams

import com.twitter.cr\_mixer.param.SimClustersANNParams

import com.twitter.cr\_mixer.param.TweetBasedCandidateGenerationParams

import com.twitter.cr\_mixer.param.decider.CrMixerDecider

import com.twitter.cr\_mixer.param.decider.DeciderConstants

import com.twitter.cr\_mixer.similarity\_engine.ConsumerBasedWalsSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.ConsumersBasedUserAdGraphSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.FilterUtil

import com.twitter.cr\_mixer.similarity\_engine.HnswANNEngineQuery

import com.twitter.cr\_mixer.similarity\_engine.HnswANNSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.ProducerBasedUserAdGraphSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.SimClustersANNSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.SimClustersANNSimilarityEngine.Query

import com.twitter.cr\_mixer.similarity\_engine.StandardSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.TweetBasedUserAdGraphSimilarityEngine

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

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

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

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi

import com.twitter.timelines.configapi.Params

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

@Singleton

case class AdsCandidateSourcesRouter @Inject() (

activePromotedTweetStore: ReadableStore[TweetId, Seq[LineItemInfo]],

decider: CrMixerDecider,

@Named(ModuleNames.SimClustersANNSimilarityEngine) simClustersANNSimilarityEngine: StandardSimilarityEngine[

Query,

TweetWithScore

],

@Named(ModuleNames.TweetBasedUserAdGraphSimilarityEngine)

tweetBasedUserAdGraphSimilarityEngine: StandardSimilarityEngine[

TweetBasedUserAdGraphSimilarityEngine.Query,

TweetWithScore

],

@Named(ModuleNames.ConsumersBasedUserAdGraphSimilarityEngine)

consumersBasedUserAdGraphSimilarityEngine: StandardSimilarityEngine[

ConsumersBasedUserAdGraphSimilarityEngine.Query,

TweetWithScore

],

@Named(ModuleNames.ProducerBasedUserAdGraphSimilarityEngine)

producerBasedUserAdGraphSimilarityEngine: StandardSimilarityEngine[

ProducerBasedUserAdGraphSimilarityEngine.Query,

TweetWithScore

],

@Named(ModuleNames.TweetBasedTwHINANNSimilarityEngine)

tweetBasedTwHINANNSimilarityEngine: HnswANNSimilarityEngine,

@Named(ModuleNames.ConsumerEmbeddingBasedTwHINANNSimilarityEngine) consumerTwHINANNSimilarityEngine: HnswANNSimilarityEngine,

@Named(ModuleNames.ConsumerBasedWalsSimilarityEngine)

consumerBasedWalsSimilarityEngine: StandardSimilarityEngine[

ConsumerBasedWalsSimilarityEngine.Query,

TweetWithScore

],

globalStats: StatsReceiver,

) {

import AdsCandidateSourcesRouter.\_

val stats: StatsReceiver = globalStats.scope(this.getClass.getSimpleName)

def fetchCandidates(

requestUserId: UserId,

sourceSignals: Set[SourceInfo],

realGraphSeeds: Map[UserId, Double],

params: configapi.Params

): Future[Seq[Seq[InitialAdsCandidate]]] = {

val simClustersANN1ConfigId = params(SimClustersANNParams.SimClustersANN1ConfigId)

val tweetBasedSANNMinScore = params(

TweetBasedCandidateGenerationParams.SimClustersMinScoreParam)

val tweetBasedSANN1Candidates =

if (params(TweetBasedCandidateGenerationParams.EnableSimClustersANN1Param)) {

Future.collect(

CandidateSourcesRouter.getTweetBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getSimClustersANNCandidates(

requestUserId,

Some(sourceInfo),

params,

simClustersANN1ConfigId,

tweetBasedSANNMinScore)

})

} else Future.value(Seq.empty)

val simClustersANN2ConfigId = params(SimClustersANNParams.SimClustersANN2ConfigId)

val tweetBasedSANN2Candidates =

if (params(TweetBasedCandidateGenerationParams.EnableSimClustersANN2Param)) {

Future.collect(

CandidateSourcesRouter.getTweetBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getSimClustersANNCandidates(

requestUserId,

Some(sourceInfo),

params,

simClustersANN2ConfigId,

tweetBasedSANNMinScore)

})

} else Future.value(Seq.empty)

val tweetBasedUagCandidates =

if (params(TweetBasedCandidateGenerationParams.EnableUAGParam)) {

Future.collect(

CandidateSourcesRouter.getTweetBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getTweetBasedUserAdGraphCandidates(Some(sourceInfo), params)

})

} else Future.value(Seq.empty)

val realGraphInNetworkBasedUagCandidates =

if (params(ConsumersBasedUserAdGraphParams.EnableSourceParam)) {

getRealGraphConsumersBasedUserAdGraphCandidates(realGraphSeeds, params).map(Seq(\_))

} else Future.value(Seq.empty)

val producerBasedUagCandidates =

if (params(ProducerBasedCandidateGenerationParams.EnableUAGParam)) {

Future.collect(

CandidateSourcesRouter.getProducerBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getProducerBasedUserAdGraphCandidates(Some(sourceInfo), params)

})

} else Future.value(Seq.empty)

val tweetBasedTwhinAdsCandidates =

if (params(TweetBasedCandidateGenerationParams.EnableTwHINParam)) {

Future.collect(

CandidateSourcesRouter.getTweetBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getTwHINAdsCandidates(

tweetBasedTwHINANNSimilarityEngine,

SimilarityEngineType.TweetBasedTwHINANN,

requestUserId,

Some(sourceInfo),

ModelConfig.DebuggerDemo)

})

} else Future.value(Seq.empty)

val producerBasedSANNMinScore = params(

ProducerBasedCandidateGenerationParams.SimClustersMinScoreParam)

val producerBasedSANN1Candidates =

if (params(ProducerBasedCandidateGenerationParams.EnableSimClustersANN1Param)) {

Future.collect(

CandidateSourcesRouter.getProducerBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getSimClustersANNCandidates(

requestUserId,

Some(sourceInfo),

params,

simClustersANN1ConfigId,

producerBasedSANNMinScore)

})

} else Future.value(Seq.empty)

val producerBasedSANN2Candidates =

if (params(ProducerBasedCandidateGenerationParams.EnableSimClustersANN2Param)) {

Future.collect(

CandidateSourcesRouter.getProducerBasedSourceInfo(sourceSignals).toSeq.map { sourceInfo =>

getSimClustersANNCandidates(

requestUserId,

Some(sourceInfo),

params,

simClustersANN2ConfigId,

producerBasedSANNMinScore)

})

} else Future.value(Seq.empty)

val interestedInMinScore = params(InterestedInParams.MinScoreParam)

val interestedInSANN1Candidates = if (params(InterestedInParams.EnableSimClustersANN1Param)) {

getSimClustersANNCandidates(

requestUserId,

None,

params,

simClustersANN1ConfigId,

interestedInMinScore).map(Seq(\_))

} else Future.value(Seq.empty)

val interestedInSANN2Candidates = if (params(InterestedInParams.EnableSimClustersANN2Param)) {

getSimClustersANNCandidates(

requestUserId,

None,

params,

simClustersANN2ConfigId,

interestedInMinScore).map(Seq(\_))

} else Future.value(Seq.empty)

val consumerTwHINAdsCandidates =

if (params(ConsumerEmbeddingBasedCandidateGenerationParams.EnableTwHINParam)) {

getTwHINAdsCandidates(

consumerTwHINANNSimilarityEngine,

SimilarityEngineType.ConsumerEmbeddingBasedTwHINANN,

requestUserId,

None,

ModelConfig.DebuggerDemo).map(Seq(\_))

} else Future.value(Seq.empty)

val consumerBasedWalsCandidates =

if (params(

ConsumerBasedWalsParams.EnableSourceParam

)) {

getConsumerBasedWalsCandidates(sourceSignals, params)

}.map {

Seq(\_)

}

else Future.value(Seq.empty)

Future

.collect(Seq(

tweetBasedSANN1Candidates,

tweetBasedSANN2Candidates,

tweetBasedUagCandidates,

tweetBasedTwhinAdsCandidates,

producerBasedUagCandidates,

producerBasedSANN1Candidates,

producerBasedSANN2Candidates,

realGraphInNetworkBasedUagCandidates,

interestedInSANN1Candidates,

interestedInSANN2Candidates,

consumerTwHINAdsCandidates,

consumerBasedWalsCandidates,

)).map(\_.flatten).map { tweetsWithCGInfoSeq =>

Future.collect(

tweetsWithCGInfoSeq.map(candidates => convertToInitialCandidates(candidates, stats)))

}.flatten.map { candidatesLists =>

val result = candidatesLists.filter(\_.nonEmpty)

stats.stat("numOfSequences").add(result.size)

stats.stat("flattenCandidatesWithDup").add(result.flatten.size)

result

}

}

private[candidate\_generation] def convertToInitialCandidates(

candidates: Seq[TweetWithCandidateGenerationInfo],

stats: StatsReceiver

): Future[Seq[InitialAdsCandidate]] = {

val tweetIds = candidates.map(\_.tweetId).toSet

stats.stat("initialCandidateSizeBeforeLineItemFilter").add(tweetIds.size)

Future.collect(activePromotedTweetStore.multiGet(tweetIds)).map { lineItemInfos =>

/\*\* \*

\* If lineItemInfo does not exist, we will filter out the promoted tweet as it cannot be targeted and ranked in admixer

\*/

val filteredCandidates = candidates.collect {

case candidate if lineItemInfos.getOrElse(candidate.tweetId, None).isDefined =>

val lineItemInfo = lineItemInfos(candidate.tweetId)

.getOrElse(throw new IllegalStateException("Check previous line's condition"))

InitialAdsCandidate(

tweetId = candidate.tweetId,

lineItemInfo = lineItemInfo,

candidate.candidateGenerationInfo

)

}

stats.stat("initialCandidateSizeAfterLineItemFilter").add(filteredCandidates.size)

filteredCandidates

}

}

private[candidate\_generation] def getSimClustersANNCandidates(

requestUserId: UserId,

sourceInfo: Option[SourceInfo],

params: configapi.Params,

configId: String,

minScore: Double

) = {

val simClustersModelVersion =

ModelVersions.Enum.enumToSimClustersModelVersionMap(params(GlobalParams.ModelVersionParam))

val embeddingType =

if (sourceInfo.isEmpty) {

params(InterestedInParams.InterestedInEmbeddingIdParam).embeddingType

} else getSimClustersANNEmbeddingType(sourceInfo.get)

val query = SimClustersANNSimilarityEngine.fromParams(

if (sourceInfo.isEmpty) InternalId.UserId(requestUserId) else sourceInfo.get.internalId,

embeddingType,

simClustersModelVersion,

configId,

params

)

// dark traffic to simclusters-ann-2

if (decider.isAvailable(DeciderConstants.enableSimClustersANN2DarkTrafficDeciderKey)) {

val simClustersANN2ConfigId = params(SimClustersANNParams.SimClustersANN2ConfigId)

val sann2Query = SimClustersANNSimilarityEngine.fromParams(

if (sourceInfo.isEmpty) InternalId.UserId(requestUserId) else sourceInfo.get.internalId,

embeddingType,

simClustersModelVersion,

simClustersANN2ConfigId,

params

)

simClustersANNSimilarityEngine

.getCandidates(sann2Query)

}

simClustersANNSimilarityEngine

.getCandidates(query).map(\_.getOrElse(Seq.empty)).map(\_.filter(\_.score > minScore).map {

tweetWithScore =>

val similarityEngineInfo = SimClustersANNSimilarityEngine

.toSimilarityEngineInfo(query, tweetWithScore.score)

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

sourceInfo,

similarityEngineInfo,

Seq(similarityEngineInfo)

))

})

}

private[candidate\_generation] def getProducerBasedUserAdGraphCandidates(

sourceInfo: Option[SourceInfo],

params: configapi.Params

) = {

val query = ProducerBasedUserAdGraphSimilarityEngine.fromParams(

sourceInfo.get.internalId,

params

)

producerBasedUserAdGraphSimilarityEngine

.getCandidates(query).map(\_.getOrElse(Seq.empty)).map(\_.map { tweetWithScore =>

val similarityEngineInfo = ProducerBasedUserAdGraphSimilarityEngine

.toSimilarityEngineInfo(tweetWithScore.score)

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

sourceInfo,

similarityEngineInfo,

Seq(similarityEngineInfo)

))

})

}

private[candidate\_generation] def getTweetBasedUserAdGraphCandidates(

sourceInfo: Option[SourceInfo],

params: configapi.Params

) = {

val query = TweetBasedUserAdGraphSimilarityEngine.fromParams(

sourceInfo.get.internalId,

params

)

tweetBasedUserAdGraphSimilarityEngine

.getCandidates(query).map(\_.getOrElse(Seq.empty)).map(\_.map { tweetWithScore =>

val similarityEngineInfo = TweetBasedUserAdGraphSimilarityEngine

.toSimilarityEngineInfo(tweetWithScore.score)

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

sourceInfo,

similarityEngineInfo,

Seq(similarityEngineInfo)

))

})

}

private[candidate\_generation] def getRealGraphConsumersBasedUserAdGraphCandidates(

realGraphSeeds: Map[UserId, Double],

params: configapi.Params

) = {

val query = ConsumersBasedUserAdGraphSimilarityEngine

.fromParams(realGraphSeeds, params)

// The internalId is a placeholder value. We do not plan to store the full seedUserId set.

val sourceInfo = SourceInfo(

sourceType = SourceType.RealGraphIn,

internalId = InternalId.UserId(0L),

sourceEventTime = None

)

consumersBasedUserAdGraphSimilarityEngine

.getCandidates(query).map(\_.getOrElse(Seq.empty)).map(\_.map { tweetWithScore =>

val similarityEngineInfo = ConsumersBasedUserAdGraphSimilarityEngine

.toSimilarityEngineInfo(tweetWithScore.score)

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

Some(sourceInfo),

similarityEngineInfo,

Seq.empty // Atomic Similarity Engine. Hence it has no contributing SEs

)

)

})

}

private[candidate\_generation] def getTwHINAdsCandidates(

similarityEngine: HnswANNSimilarityEngine,

similarityEngineType: SimilarityEngineType,

requestUserId: UserId,

sourceInfo: Option[SourceInfo], // if none, then it's consumer-based similarity engine

model: String

): Future[Seq[TweetWithCandidateGenerationInfo]] = {

val internalId =

if (sourceInfo.nonEmpty) sourceInfo.get.internalId else InternalId.UserId(requestUserId)

similarityEngine

.getCandidates(buildHnswANNQuery(internalId, model)).map(\_.getOrElse(Seq.empty)).map(\_.map {

tweetWithScore =>

val similarityEngineInfo = SimilarityEngineInfo(

similarityEngineType = similarityEngineType,

modelId = Some(model),

score = Some(tweetWithScore.score))

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

None,

similarityEngineInfo,

Seq(similarityEngineInfo)

))

})

}

private[candidate\_generation] def getConsumerBasedWalsCandidates(

sourceSignals: Set[SourceInfo],

params: configapi.Params

): Future[Seq[TweetWithCandidateGenerationInfo]] = {

// Fetch source signals and filter them based on age.

val signals = FilterUtil.tweetSourceAgeFilter(

getConsumerBasedWalsSourceInfo(sourceSignals).toSeq,

params(ConsumerBasedWalsParams.MaxTweetSignalAgeHoursParam))

val candidatesOptFut = consumerBasedWalsSimilarityEngine.getCandidates(

ConsumerBasedWalsSimilarityEngine.fromParams(signals, params)

)

val tweetsWithCandidateGenerationInfoOptFut = candidatesOptFut.map {

\_.map { tweetsWithScores =>

val sortedCandidates = tweetsWithScores.sortBy(-\_.score)

val filteredCandidates =

FilterUtil.tweetAgeFilter(sortedCandidates, params(GlobalParams.MaxTweetAgeHoursParam))

consumerBasedWalsSimilarityEngine.getScopedStats

.stat("filteredCandidates\_size").add(filteredCandidates.size)

val tweetsWithCandidateGenerationInfo = filteredCandidates.map { tweetWithScore =>

{

val similarityEngineInfo =

ConsumerBasedWalsSimilarityEngine.toSimilarityEngineInfo(tweetWithScore.score)

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

None,

similarityEngineInfo,

Seq.empty // Atomic Similarity Engine. Hence it has no contributing SEs

)

)

}

}

val maxCandidateNum = params(GlobalParams.MaxCandidateNumPerSourceKeyParam)

tweetsWithCandidateGenerationInfo.take(maxCandidateNum)

}

}

for {

tweetsWithCandidateGenerationInfoOpt <- tweetsWithCandidateGenerationInfoOptFut

} yield tweetsWithCandidateGenerationInfoOpt.toSeq.flatten

}

}

object AdsCandidateSourcesRouter {

def getSimClustersANNEmbeddingType(

sourceInfo: SourceInfo

): EmbeddingType = {

sourceInfo.sourceType match {

case SourceType.TweetFavorite | SourceType.Retweet | SourceType.OriginalTweet |

SourceType.Reply | SourceType.TweetShare | SourceType.NotificationClick |

SourceType.GoodTweetClick | SourceType.VideoTweetQualityView |

SourceType.VideoTweetPlayback50 =>

EmbeddingType.LogFavLongestL2EmbeddingTweet

case SourceType.UserFollow | SourceType.UserRepeatedProfileVisit | SourceType.RealGraphOon |

SourceType.FollowRecommendation | SourceType.UserTrafficAttributionProfileVisit |

SourceType.GoodProfileClick | SourceType.TwiceUserId =>

EmbeddingType.FavBasedProducer

case \_ => throw new IllegalArgumentException("sourceInfo.sourceType not supported")

}

}

def buildHnswANNQuery(internalId: InternalId, modelId: String): HnswANNEngineQuery = {

HnswANNEngineQuery(

sourceId = internalId,

modelId = modelId,

params = Params.Empty

)

}

def getConsumerBasedWalsSourceInfo(

sourceSignals: Set[SourceInfo]

): Set[SourceInfo] = {

val AllowedSourceTypesForConsumerBasedWalsSE = Set(

SourceType.TweetFavorite.value,

SourceType.Retweet.value,

SourceType.TweetDontLike.value, //currently no-op

SourceType.TweetReport.value, //currently no-op

SourceType.AccountMute.value, //currently no-op

SourceType.AccountBlock.value //currently no-op

)

sourceSignals.collect {

case sourceInfo

if AllowedSourceTypesForConsumerBasedWalsSE.contains(sourceInfo.sourceType.value) =>

sourceInfo

}

}

}