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

import com.twitter.contentrecommender.thriftscala.TweetInfo

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

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

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

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.TripTweetWithScore

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

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

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

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

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

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

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

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

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

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

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

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

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

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.ProducerBasedUnifiedSimilarityEngine

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

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

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

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

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.TweetId

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.timelines.configapi

import com.twitter.util.Future

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

/\*\*

\* Route the SourceInfo to the associated Candidate Engines.

\*/

@Singleton

case class CandidateSourcesRouter @Inject() (

customizedRetrievalCandidateGeneration: CustomizedRetrievalCandidateGeneration,

simClustersInterestedInCandidateGeneration: SimClustersInterestedInCandidateGeneration,

@Named(ModuleNames.TweetBasedUnifiedSimilarityEngine)

tweetBasedUnifiedSimilarityEngine: StandardSimilarityEngine[

TweetBasedUnifiedSimilarityEngine.Query,

TweetWithCandidateGenerationInfo

],

@Named(ModuleNames.ProducerBasedUnifiedSimilarityEngine)

producerBasedUnifiedSimilarityEngine: StandardSimilarityEngine[

ProducerBasedUnifiedSimilarityEngine.Query,

TweetWithCandidateGenerationInfo

],

@Named(ModuleNames.ConsumerEmbeddingBasedTripSimilarityEngine)

consumerEmbeddingBasedTripSimilarityEngine: StandardSimilarityEngine[

TripEngineQuery,

TripTweetWithScore

],

@Named(ModuleNames.ConsumerEmbeddingBasedTwHINANNSimilarityEngine)

consumerBasedTwHINANNSimilarityEngine: HnswANNSimilarityEngine,

@Named(ModuleNames.ConsumerEmbeddingBasedTwoTowerANNSimilarityEngine)

consumerBasedTwoTowerSimilarityEngine: HnswANNSimilarityEngine,

@Named(ModuleNames.ConsumersBasedUserVideoGraphSimilarityEngine)

consumersBasedUserVideoGraphSimilarityEngine: StandardSimilarityEngine[

ConsumersBasedUserVideoGraphSimilarityEngine.Query,

TweetWithScore

],

@Named(ModuleNames.UserTweetEntityGraphSimilarityEngine) userTweetEntityGraphSimilarityEngine: StandardSimilarityEngine[

UserTweetEntityGraphSimilarityEngine.Query,

TweetWithScoreAndSocialProof

],

@Named(ModuleNames.ConsumerBasedWalsSimilarityEngine)

consumerBasedWalsSimilarityEngine: StandardSimilarityEngine[

ConsumerBasedWalsSimilarityEngine.Query,

TweetWithScore

],

tweetInfoStore: ReadableStore[TweetId, TweetInfo],

globalStats: StatsReceiver,

) {

import CandidateSourcesRouter.\_

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

def fetchCandidates(

requestUserId: UserId,

sourceSignals: Set[SourceInfo],

sourceGraphs: Map[String, Option[GraphSourceInfo]],

params: configapi.Params,

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

val tweetBasedCandidatesFuture = getCandidates(

getTweetBasedSourceInfo(sourceSignals),

params,

TweetBasedUnifiedSimilarityEngine.fromParams,

tweetBasedUnifiedSimilarityEngine.getCandidates)

val producerBasedCandidatesFuture =

getCandidates(

getProducerBasedSourceInfo(sourceSignals),

params,

ProducerBasedUnifiedSimilarityEngine.fromParams(\_, \_),

producerBasedUnifiedSimilarityEngine.getCandidates

)

val simClustersInterestedInBasedCandidatesFuture =

getCandidatesPerSimilarityEngineModel(

requestUserId,

params,

SimClustersInterestedInCandidateGeneration.fromParams,

simClustersInterestedInCandidateGeneration.get)

val consumerEmbeddingBasedLogFavBasedTripCandidatesFuture =

if (params(

ConsumerEmbeddingBasedCandidateGenerationParams.EnableLogFavBasedSimClustersTripParam)) {

getSimClustersTripCandidates(

params,

ConsumerEmbeddingBasedTripSimilarityEngine.fromParams(

ModelConfig.ConsumerLogFavBasedInterestedInEmbedding,

InternalId.UserId(requestUserId),

params

),

consumerEmbeddingBasedTripSimilarityEngine

).map {

Seq(\_)

}

} else

Future.Nil

val consumersBasedUvgRealGraphInCandidatesFuture =

if (params(ConsumersBasedUserVideoGraphParams.EnableSourceParam)) {

val realGraphInGraphSourceInfoOpt =

getGraphSourceInfoBySourceType(SourceType.RealGraphIn.name, sourceGraphs)

getGraphBasedCandidates(

params,

ConsumersBasedUserVideoGraphSimilarityEngine

.fromParamsForRealGraphIn(

realGraphInGraphSourceInfoOpt

.map { graphSourceInfo => graphSourceInfo.seedWithScores }.getOrElse(Map.empty),

params),

consumersBasedUserVideoGraphSimilarityEngine,

ConsumersBasedUserVideoGraphSimilarityEngine.toSimilarityEngineInfo,

realGraphInGraphSourceInfoOpt

).map {

Seq(\_)

}

} else Future.Nil

val consumerEmbeddingBasedFollowBasedTripCandidatesFuture =

if (params(

ConsumerEmbeddingBasedCandidateGenerationParams.EnableFollowBasedSimClustersTripParam)) {

getSimClustersTripCandidates(

params,

ConsumerEmbeddingBasedTripSimilarityEngine.fromParams(

ModelConfig.ConsumerFollowBasedInterestedInEmbedding,

InternalId.UserId(requestUserId),

params

),

consumerEmbeddingBasedTripSimilarityEngine

).map {

Seq(\_)

}

} else

Future.Nil

val consumerBasedWalsCandidatesFuture =

if (params(

ConsumerBasedWalsParams.EnableSourceParam

)) {

getConsumerBasedWalsCandidates(sourceSignals, params)

}.map { Seq(\_) }

else Future.Nil

val consumerEmbeddingBasedTwHINCandidatesFuture =

if (params(ConsumerEmbeddingBasedCandidateGenerationParams.EnableTwHINParam)) {

getHnswCandidates(

params,

ConsumerEmbeddingBasedTwHINSimilarityEngine.fromParams(

InternalId.UserId(requestUserId),

params),

consumerBasedTwHINANNSimilarityEngine

).map { Seq(\_) }

} else Future.Nil

val consumerEmbeddingBasedTwoTowerCandidatesFuture =

if (params(ConsumerEmbeddingBasedCandidateGenerationParams.EnableTwoTowerParam)) {

getHnswCandidates(

params,

ConsumerEmbeddingBasedTwoTowerSimilarityEngine.fromParams(

InternalId.UserId(requestUserId),

params),

consumerBasedTwoTowerSimilarityEngine

).map {

Seq(\_)

}

} else Future.Nil

val customizedRetrievalBasedCandidatesFuture =

getCandidatesPerSimilarityEngineModel(

requestUserId,

params,

CustomizedRetrievalCandidateGeneration.fromParams,

customizedRetrievalCandidateGeneration.get)

Future

.collect(

Seq(

tweetBasedCandidatesFuture,

producerBasedCandidatesFuture,

simClustersInterestedInBasedCandidatesFuture,

consumerBasedWalsCandidatesFuture,

consumerEmbeddingBasedLogFavBasedTripCandidatesFuture,

consumerEmbeddingBasedFollowBasedTripCandidatesFuture,

consumerEmbeddingBasedTwHINCandidatesFuture,

consumerEmbeddingBasedTwoTowerCandidatesFuture,

consumersBasedUvgRealGraphInCandidatesFuture,

customizedRetrievalBasedCandidatesFuture

)).map { candidatesList =>

// remove empty innerSeq

val result = candidatesList.flatten.filter(\_.nonEmpty)

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

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

result

}

}

private def getGraphBasedCandidates[QueryType](

params: configapi.Params,

query: EngineQuery[QueryType],

engine: StandardSimilarityEngine[QueryType, TweetWithScore],

toSimilarityEngineInfo: Double => SimilarityEngineInfo,

graphSourceInfoOpt: Option[GraphSourceInfo] = None

): Future[Seq[InitialCandidate]] = {

val candidatesOptFut = engine.getCandidates(query)

val tweetsWithCandidateGenerationInfoOptFut = candidatesOptFut.map {

\_.map { tweetsWithScores =>

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

engine.getScopedStats.stat("sortedCandidates\_size").add(sortedCandidates.size)

val tweetsWithCandidateGenerationInfo = sortedCandidates.map { tweetWithScore =>

{

val similarityEngineInfo = toSimilarityEngineInfo(tweetWithScore.score)

val sourceInfo = graphSourceInfoOpt.map { graphSourceInfo =>

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

SourceInfo(

sourceType = graphSourceInfo.sourceType,

internalId = InternalId.UserId(0L),

sourceEventTime = None

)

}

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

sourceInfo,

similarityEngineInfo,

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

)

)

}

}

val maxCandidateNum = params(GlobalParams.MaxCandidateNumPerSourceKeyParam)

tweetsWithCandidateGenerationInfo.take(maxCandidateNum)

}

}

for {

tweetsWithCandidateGenerationInfoOpt <- tweetsWithCandidateGenerationInfoOptFut

initialCandidates <- convertToInitialCandidates(

tweetsWithCandidateGenerationInfoOpt.toSeq.flatten)

} yield initialCandidates

}

private def getCandidates[QueryType](

sourceSignals: Set[SourceInfo],

params: configapi.Params,

fromParams: (SourceInfo, configapi.Params) => QueryType,

getFunc: QueryType => Future[Option[Seq[TweetWithCandidateGenerationInfo]]]

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

val queries = sourceSignals.map { sourceInfo =>

fromParams(sourceInfo, params)

}.toSeq

Future

.collect {

queries.map { query =>

for {

candidates <- getFunc(query)

prefilterCandidates <- convertToInitialCandidates(candidates.toSeq.flatten)

} yield {

prefilterCandidates

}

}

}

}

private def getConsumerBasedWalsCandidates(

sourceSignals: Set[SourceInfo],

params: configapi.Params

): Future[Seq[InitialCandidate]] = {

// 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

initialCandidates <- convertToInitialCandidates(

tweetsWithCandidateGenerationInfoOpt.toSeq.flatten)

} yield initialCandidates

}

private def getSimClustersTripCandidates(

params: configapi.Params,

query: TripEngineQuery,

engine: StandardSimilarityEngine[

TripEngineQuery,

TripTweetWithScore

],

): Future[Seq[InitialCandidate]] = {

val tweetsWithCandidatesGenerationInfoOptFut =

engine.getCandidates(EngineQuery(query, params)).map {

\_.map {

\_.map { tweetWithScore =>

// define filters

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

None,

SimilarityEngineInfo(

SimilarityEngineType.ExploreTripOfflineSimClustersTweets,

None,

Some(tweetWithScore.score)),

Seq.empty

)

)

}

}

}

for {

tweetsWithCandidateGenerationInfoOpt <- tweetsWithCandidatesGenerationInfoOptFut

initialCandidates <- convertToInitialCandidates(

tweetsWithCandidateGenerationInfoOpt.toSeq.flatten)

} yield initialCandidates

}

private def getHnswCandidates(

params: configapi.Params,

query: HnswANNEngineQuery,

engine: HnswANNSimilarityEngine,

): Future[Seq[InitialCandidate]] = {

val candidatesOptFut = engine.getCandidates(query)

val tweetsWithCandidateGenerationInfoOptFut = candidatesOptFut.map {

\_.map { tweetsWithScores =>

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

val filteredCandidates =

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

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

val tweetsWithCandidateGenerationInfo = filteredCandidates.map { tweetWithScore =>

{

val similarityEngineInfo =

engine.toSimilarityEngineInfo(query, 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

initialCandidates <- convertToInitialCandidates(

tweetsWithCandidateGenerationInfoOpt.toSeq.flatten)

} yield initialCandidates

}

/\*\*

\* Returns candidates from each similarity engine separately.

\* For 1 requestUserId, it will fetch results from each similarity engine e\_i,

\* and returns Seq[Seq[TweetCandidate]].

\*/

private def getCandidatesPerSimilarityEngineModel[QueryType](

requestUserId: UserId,

params: configapi.Params,

fromParams: (InternalId, configapi.Params) => QueryType,

getFunc: QueryType => Future[

Option[Seq[Seq[TweetWithCandidateGenerationInfo]]]

]

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

val query = fromParams(InternalId.UserId(requestUserId), params)

getFunc(query).flatMap { candidatesPerSimilarityEngineModelOpt =>

val candidatesPerSimilarityEngineModel = candidatesPerSimilarityEngineModelOpt.toSeq.flatten

Future.collect {

candidatesPerSimilarityEngineModel.map(convertToInitialCandidates)

}

}

}

private[candidate\_generation] def convertToInitialCandidates(

candidates: Seq[TweetWithCandidateGenerationInfo],

): Future[Seq[InitialCandidate]] = {

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

Future.collect(tweetInfoStore.multiGet(tweetIds)).map { tweetInfos =>

/\*\*\*

\* If tweetInfo does not exist, we will filter out this tweet candidate.

\*/

candidates.collect {

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

val tweetInfo = tweetInfos(candidate.tweetId)

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

InitialCandidate(

tweetId = candidate.tweetId,

tweetInfo = tweetInfo,

candidate.candidateGenerationInfo

)

}

}

}

}

object CandidateSourcesRouter {

def getGraphSourceInfoBySourceType(

sourceTypeStr: String,

sourceGraphs: Map[String, Option[GraphSourceInfo]]

): Option[GraphSourceInfo] = {

sourceGraphs.getOrElse(sourceTypeStr, None)

}

def getTweetBasedSourceInfo(

sourceSignals: Set[SourceInfo]

): Set[SourceInfo] = {

sourceSignals.collect {

case sourceInfo

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

sourceInfo

}

}

def getProducerBasedSourceInfo(

sourceSignals: Set[SourceInfo]

): Set[SourceInfo] = {

sourceSignals.collect {

case sourceInfo

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

sourceInfo

}

}

def getConsumerBasedWalsSourceInfo(

sourceSignals: Set[SourceInfo]

): Set[SourceInfo] = {

sourceSignals.collect {

case sourceInfo

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

sourceInfo

}

}

/\*\*\*

\* Signal funneling should not exist in CG or even in any SimilarityEngine.

\* They will be in Router, or eventually, in CrCandidateGenerator.

\*/

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

)

val AllowedSourceTypesForTweetBasedUnifiedSE = Set(

SourceType.TweetFavorite.value,

SourceType.Retweet.value,

SourceType.OriginalTweet.value,

SourceType.Reply.value,

SourceType.TweetShare.value,

SourceType.NotificationClick.value,

SourceType.GoodTweetClick.value,

SourceType.VideoTweetQualityView.value,

SourceType.VideoTweetPlayback50.value,

SourceType.TweetAggregation.value,

)

val AllowedSourceTypesForProducerBasedUnifiedSE = Set(

SourceType.UserFollow.value,

SourceType.UserRepeatedProfileVisit.value,

SourceType.RealGraphOon.value,

SourceType.FollowRecommendation.value,

SourceType.UserTrafficAttributionProfileVisit.value,

SourceType.GoodProfileClick.value,

SourceType.ProducerAggregation.value,

)

}