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

import com.twitter.cr\_mixer.candidate\_generation.CustomizedRetrievalCandidateGeneration.Query

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

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

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

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

import com.twitter.cr\_mixer.param.CustomizedRetrievalBasedCandidateGenerationParams.\_

import com.twitter.cr\_mixer.param.CustomizedRetrievalBasedTwhinParams.\_

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

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

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

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

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

import com.twitter.cr\_mixer.util.InterleaveUtil

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateSource

import com.twitter.frigate.common.base.Stats

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

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.timelines.configapi

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

import scala.collection.mutable.ArrayBuffer

/\*\*

\* A candidate generator that fetches similar tweets from multiple customized retrieval based candidate sources

\*

\* Different from [[TweetBasedCandidateGeneration]], this store returns candidates from different

\* similarity engines without blending. In other words, this class shall not be thought of as a

\* Unified Similarity Engine. It is a CG that calls multiple singular Similarity Engines.

\*/

@Singleton

case class CustomizedRetrievalCandidateGeneration @Inject() (

@Named(ModuleNames.TwhinCollabFilterSimilarityEngine)

twhinCollabFilterSimilarityEngine: LookupSimilarityEngine[

TwhinCollabFilterSimilarityEngine.Query,

TweetWithScore

],

@Named(ModuleNames.DiffusionBasedSimilarityEngine)

diffusionBasedSimilarityEngine: LookupSimilarityEngine[

DiffusionBasedSimilarityEngine.Query,

TweetWithScore

],

statsReceiver: StatsReceiver)

extends CandidateSource[

Query,

Seq[TweetWithCandidateGenerationInfo]

] {

override def name: String = this.getClass.getSimpleName

private val stats = statsReceiver.scope(name)

private val fetchCandidatesStat = stats.scope("fetchCandidates")

/\*\*

\* For each Similarity Engine Model, return a list of tweet candidates

\*/

override def get(

query: Query

): Future[Option[Seq[Seq[TweetWithCandidateGenerationInfo]]]] = {

query.internalId match {

case InternalId.UserId(\_) =>

Stats.trackOption(fetchCandidatesStat) {

val twhinCollabFilterForFollowCandidatesFut = if (query.enableTwhinCollabFilter) {

twhinCollabFilterSimilarityEngine.getCandidates(query.twhinCollabFilterFollowQuery)

} else Future.None

val twhinCollabFilterForEngagementCandidatesFut =

if (query.enableTwhinCollabFilter) {

twhinCollabFilterSimilarityEngine.getCandidates(

query.twhinCollabFilterEngagementQuery)

} else Future.None

val twhinMultiClusterForFollowCandidatesFut = if (query.enableTwhinMultiCluster) {

twhinCollabFilterSimilarityEngine.getCandidates(query.twhinMultiClusterFollowQuery)

} else Future.None

val twhinMultiClusterForEngagementCandidatesFut =

if (query.enableTwhinMultiCluster) {

twhinCollabFilterSimilarityEngine.getCandidates(

query.twhinMultiClusterEngagementQuery)

} else Future.None

val diffusionBasedSimilarityEngineCandidatesFut = if (query.enableRetweetBasedDiffusion) {

diffusionBasedSimilarityEngine.getCandidates(query.diffusionBasedSimilarityEngineQuery)

} else Future.None

Future

.join(

twhinCollabFilterForFollowCandidatesFut,

twhinCollabFilterForEngagementCandidatesFut,

twhinMultiClusterForFollowCandidatesFut,

twhinMultiClusterForEngagementCandidatesFut,

diffusionBasedSimilarityEngineCandidatesFut

).map {

case (

twhinCollabFilterForFollowCandidates,

twhinCollabFilterForEngagementCandidates,

twhinMultiClusterForFollowCandidates,

twhinMultiClusterForEngagementCandidates,

diffusionBasedSimilarityEngineCandidates) =>

val maxCandidateNumPerSourceKey = 200

val twhinCollabFilterForFollowWithCGInfo =

getTwhinCollabCandidatesWithCGInfo(

twhinCollabFilterForFollowCandidates,

maxCandidateNumPerSourceKey,

query.twhinCollabFilterFollowQuery,

)

val twhinCollabFilterForEngagementWithCGInfo =

getTwhinCollabCandidatesWithCGInfo(

twhinCollabFilterForEngagementCandidates,

maxCandidateNumPerSourceKey,

query.twhinCollabFilterEngagementQuery,

)

val twhinMultiClusterForFollowWithCGInfo =

getTwhinCollabCandidatesWithCGInfo(

twhinMultiClusterForFollowCandidates,

maxCandidateNumPerSourceKey,

query.twhinMultiClusterFollowQuery,

)

val twhinMultiClusterForEngagementWithCGInfo =

getTwhinCollabCandidatesWithCGInfo(

twhinMultiClusterForEngagementCandidates,

maxCandidateNumPerSourceKey,

query.twhinMultiClusterEngagementQuery,

)

val retweetBasedDiffusionWithCGInfo =

getDiffusionBasedCandidatesWithCGInfo(

diffusionBasedSimilarityEngineCandidates,

maxCandidateNumPerSourceKey,

query.diffusionBasedSimilarityEngineQuery,

)

val twhinCollabCandidateSourcesToBeInterleaved =

ArrayBuffer[Seq[TweetWithCandidateGenerationInfo]](

twhinCollabFilterForFollowWithCGInfo,

twhinCollabFilterForEngagementWithCGInfo,

)

val twhinMultiClusterCandidateSourcesToBeInterleaved =

ArrayBuffer[Seq[TweetWithCandidateGenerationInfo]](

twhinMultiClusterForFollowWithCGInfo,

twhinMultiClusterForEngagementWithCGInfo,

)

val interleavedTwhinCollabCandidates =

InterleaveUtil.interleave(twhinCollabCandidateSourcesToBeInterleaved)

val interleavedTwhinMultiClusterCandidates =

InterleaveUtil.interleave(twhinMultiClusterCandidateSourcesToBeInterleaved)

val twhinCollabFilterResults =

if (interleavedTwhinCollabCandidates.nonEmpty) {

Some(interleavedTwhinCollabCandidates.take(maxCandidateNumPerSourceKey))

} else None

val twhinMultiClusterResults =

if (interleavedTwhinMultiClusterCandidates.nonEmpty) {

Some(interleavedTwhinMultiClusterCandidates.take(maxCandidateNumPerSourceKey))

} else None

val diffusionResults =

if (retweetBasedDiffusionWithCGInfo.nonEmpty) {

Some(retweetBasedDiffusionWithCGInfo.take(maxCandidateNumPerSourceKey))

} else None

Some(

Seq(

twhinCollabFilterResults,

twhinMultiClusterResults,

diffusionResults

).flatten)

}

}

case \_ =>

throw new IllegalArgumentException("sourceId\_is\_not\_userId\_cnt")

}

}

/\*\* Returns a list of tweets that are generated less than `maxTweetAgeHours` hours ago \*/

private def tweetAgeFilter(

candidates: Seq[TweetWithScore],

maxTweetAgeHours: Duration

): Seq[TweetWithScore] = {

// Tweet IDs are approximately chronological (see http://go/snowflake),

// so we are building the earliest tweet id once

// The per-candidate logic here then be candidate.tweetId > earliestPermittedTweetId, which is far cheaper.

val earliestTweetId = SnowflakeId.firstIdFor(Time.now - maxTweetAgeHours)

candidates.filter { candidate => candidate.tweetId >= earliestTweetId }

}

/\*\*

\* AgeFilters tweetCandidates with stats

\* Only age filter logic is effective here (through tweetAgeFilter). This function acts mostly for metric logging.

\*/

private def ageFilterWithStats(

offlineInterestedInCandidates: Seq[TweetWithScore],

maxTweetAgeHours: Duration,

scopedStatsReceiver: StatsReceiver

): Seq[TweetWithScore] = {

scopedStatsReceiver.stat("size").add(offlineInterestedInCandidates.size)

val candidates = offlineInterestedInCandidates.map { candidate =>

TweetWithScore(candidate.tweetId, candidate.score)

}

val filteredCandidates = tweetAgeFilter(candidates, maxTweetAgeHours)

scopedStatsReceiver.stat(f"filtered\_size").add(filteredCandidates.size)

if (filteredCandidates.isEmpty) scopedStatsReceiver.counter(f"empty").incr()

filteredCandidates

}

private def getTwhinCollabCandidatesWithCGInfo(

tweetCandidates: Option[Seq[TweetWithScore]],

maxCandidateNumPerSourceKey: Int,

twhinCollabFilterQuery: LookupEngineQuery[

TwhinCollabFilterSimilarityEngine.Query

],

): Seq[TweetWithCandidateGenerationInfo] = {

val twhinTweets = tweetCandidates match {

case Some(tweetsWithScores) =>

tweetsWithScores.map { tweetWithScore =>

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

None,

TwhinCollabFilterSimilarityEngine

.toSimilarityEngineInfo(twhinCollabFilterQuery, tweetWithScore.score),

Seq.empty

)

)

}

case \_ => Seq.empty

}

twhinTweets.take(maxCandidateNumPerSourceKey)

}

private def getDiffusionBasedCandidatesWithCGInfo(

tweetCandidates: Option[Seq[TweetWithScore]],

maxCandidateNumPerSourceKey: Int,

diffusionBasedSimilarityEngineQuery: LookupEngineQuery[

DiffusionBasedSimilarityEngine.Query

],

): Seq[TweetWithCandidateGenerationInfo] = {

val diffusionTweets = tweetCandidates match {

case Some(tweetsWithScores) =>

tweetsWithScores.map { tweetWithScore =>

TweetWithCandidateGenerationInfo(

tweetWithScore.tweetId,

CandidateGenerationInfo(

None,

DiffusionBasedSimilarityEngine

.toSimilarityEngineInfo(diffusionBasedSimilarityEngineQuery, tweetWithScore.score),

Seq.empty

)

)

}

case \_ => Seq.empty

}

diffusionTweets.take(maxCandidateNumPerSourceKey)

}

}

object CustomizedRetrievalCandidateGeneration {

case class Query(

internalId: InternalId,

maxCandidateNumPerSourceKey: Int,

maxTweetAgeHours: Duration,

// twhinCollabFilter

enableTwhinCollabFilter: Boolean,

twhinCollabFilterFollowQuery: LookupEngineQuery[

TwhinCollabFilterSimilarityEngine.Query

],

twhinCollabFilterEngagementQuery: LookupEngineQuery[

TwhinCollabFilterSimilarityEngine.Query

],

// twhinMultiCluster

enableTwhinMultiCluster: Boolean,

twhinMultiClusterFollowQuery: LookupEngineQuery[

TwhinCollabFilterSimilarityEngine.Query

],

twhinMultiClusterEngagementQuery: LookupEngineQuery[

TwhinCollabFilterSimilarityEngine.Query

],

enableRetweetBasedDiffusion: Boolean,

diffusionBasedSimilarityEngineQuery: LookupEngineQuery[

DiffusionBasedSimilarityEngine.Query

],

)

def fromParams(

internalId: InternalId,

params: configapi.Params

): Query = {

val twhinCollabFilterFollowQuery =

TwhinCollabFilterSimilarityEngine.fromParams(

internalId,

params(CustomizedRetrievalBasedTwhinCollabFilterFollowSource),

params)

val twhinCollabFilterEngagementQuery =

TwhinCollabFilterSimilarityEngine.fromParams(

internalId,

params(CustomizedRetrievalBasedTwhinCollabFilterEngagementSource),

params)

val twhinMultiClusterFollowQuery =

TwhinCollabFilterSimilarityEngine.fromParams(

internalId,

params(CustomizedRetrievalBasedTwhinMultiClusterFollowSource),

params)

val twhinMultiClusterEngagementQuery =

TwhinCollabFilterSimilarityEngine.fromParams(

internalId,

params(CustomizedRetrievalBasedTwhinMultiClusterEngagementSource),

params)

val diffusionBasedSimilarityEngineQuery =

DiffusionBasedSimilarityEngine.fromParams(

internalId,

params(CustomizedRetrievalBasedRetweetDiffusionSource),

params)

Query(

internalId = internalId,

maxCandidateNumPerSourceKey = params(GlobalParams.MaxCandidateNumPerSourceKeyParam),

maxTweetAgeHours = params(GlobalParams.MaxTweetAgeHoursParam),

// twhinCollabFilter

enableTwhinCollabFilter = params(EnableTwhinCollabFilterClusterParam),

twhinCollabFilterFollowQuery = twhinCollabFilterFollowQuery,

twhinCollabFilterEngagementQuery = twhinCollabFilterEngagementQuery,

enableTwhinMultiCluster = params(EnableTwhinMultiClusterParam),

twhinMultiClusterFollowQuery = twhinMultiClusterFollowQuery,

twhinMultiClusterEngagementQuery = twhinMultiClusterEngagementQuery,

enableRetweetBasedDiffusion = params(EnableRetweetBasedDiffusionParam),

diffusionBasedSimilarityEngineQuery = diffusionBasedSimilarityEngineQuery

)

}

}