package com.twitter.timelineranker.uteg\_liked\_by\_tweets

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.recos.recos\_common.thriftscala.SocialProofType

import com.twitter.recos.user\_tweet\_entity\_graph.thriftscala.TweetRecommendation

import com.twitter.servo.util.FutureArrow

import com.twitter.servo.util.Gate

import com.twitter.storehaus.Store

import com.twitter.timelineranker.common.\_

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.core.DependencyTransformer

import com.twitter.timelineranker.core.HydratedCandidatesAndFeaturesEnvelope

import com.twitter.timelineranker.model.CandidateTweetsResult

import com.twitter.timelineranker.model.RecapQuery

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.monitoring.UsersSearchResultMonitoringTransform

import com.twitter.timelineranker.parameters.recap.RecapParams

import com.twitter.timelineranker.parameters.uteg\_liked\_by\_tweets.UtegLikedByTweetsParams

import com.twitter.timelineranker.parameters.monitoring.MonitoringParams

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.timelineranker.util.CopyContentFeaturesIntoHydratedTweetsTransform

import com.twitter.timelineranker.util.CopyContentFeaturesIntoThriftTweetFeaturesTransform

import com.twitter.timelineranker.visibility.FollowGraphDataProvider

import com.twitter.timelines.clients.gizmoduck.GizmoduckClient

import com.twitter.timelines.clients.manhattan.UserMetadataClient

import com.twitter.timelines.clients.relevance\_search.SearchClient

import com.twitter.timelines.clients.tweetypie.TweetyPieClient

import com.twitter.timelines.clients.user\_tweet\_entity\_graph.UserTweetEntityGraphClient

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.uteg\_utils.UTEGRecommendationsFilterBuilder

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.timelines.util.stats.RequestStatsReceiver

import com.twitter.util.Future

class UtegLikedByTweetsSource(

userTweetEntityGraphClient: UserTweetEntityGraphClient,

gizmoduckClient: GizmoduckClient,

searchClient: SearchClient,

tweetyPieClient: TweetyPieClient,

userMetadataClient: UserMetadataClient,

followGraphDataProvider: FollowGraphDataProvider,

contentFeaturesCache: Store[TweetId, ContentFeatures],

statsReceiver: StatsReceiver) {

private[this] val socialProofTypes = Seq(SocialProofType.Favorite)

private[this] val baseScope = statsReceiver.scope("utegLikedByTweetsSource")

private[this] val requestStats = RequestStatsReceiver(baseScope)

private[this] val failOpenScope = baseScope.scope("failOpen")

private[this] val userProfileHandler = new FailOpenHandler(failOpenScope, "userProfileInfo")

private[this] val userLanguagesHandler = new FailOpenHandler(failOpenScope, "userLanguages")

private[this] val debugAuthorsMonitoringProvider =

DependencyProvider.from(MonitoringParams.DebugAuthorsAllowListParam)

private[this] val maxFollowedUsersProvider =

DependencyProvider.value(RecapParams.MaxFollowedUsers.default)

private[this] val followGraphDataTransform =

new FollowGraphDataTransform(followGraphDataProvider, maxFollowedUsersProvider)

private[this] val searchResultsTransform =

new UtegLikedByTweetsSearchResultsTransform(

searchClient = searchClient,

statsReceiver = baseScope,

relevanceSearchProvider =

DependencyProvider.from(UtegLikedByTweetsParams.EnableRelevanceSearchParam)

)

private[this] val userProfileInfoTransform =

new UserProfileInfoTransform(userProfileHandler, gizmoduckClient)

private[this] val languagesTransform =

new UserLanguagesTransform(userLanguagesHandler, userMetadataClient)

private[this] val candidateGenerationTransform = new CandidateGenerationTransform(baseScope)

private[this] val maxCandidatesToFetchFromUtegProvider = DependencyProvider { query =>

query.utegLikedByTweetsOptions

.map(\_.utegCount).getOrElse(

query.utegLikedByTweetsOptions match {

case Some(opts) =>

if (opts.isInNetwork) query.params(UtegLikedByTweetsParams.DefaultUTEGInNetworkCount)

else query.params(UtegLikedByTweetsParams.DefaultUTEGOutOfNetworkCount)

case None => 0

}

)

}

private[this] def isInNetwork(envelope: CandidateEnvelope): Boolean =

isInNetwork(envelope.query)

private[this] def isInNetwork(query: RecapQuery): Boolean =

query.utegLikedByTweetsOptions.exists(\_.isInNetwork)

private[this] def isInNetwork(hydratedEnvelope: HydratedCandidatesAndFeaturesEnvelope): Boolean =

isInNetwork(hydratedEnvelope.candidateEnvelope)

private[this] val recommendationsFilter =

DependencyTransformer.partition[Seq[TweetRecommendation], Seq[TweetRecommendation]](

gate = Gate[RecapQuery](f = (query: RecapQuery) => isInNetwork(query)),

ifTrue = DependencyTransformer.identity,

ifFalse = new UTEGRecommendationsFilterBuilder[RecapQuery](

enablingGate =

RecapQuery.paramGate(UtegLikedByTweetsParams.UTEGRecommendationsFilter.EnableParam),

excludeTweetGate =

RecapQuery.paramGate(UtegLikedByTweetsParams.UTEGRecommendationsFilter.ExcludeTweetParam),

excludeRetweetGate = RecapQuery.paramGate(

UtegLikedByTweetsParams.UTEGRecommendationsFilter.ExcludeRetweetParam),

excludeReplyGate =

RecapQuery.paramGate(UtegLikedByTweetsParams.UTEGRecommendationsFilter.ExcludeReplyParam),

excludeQuoteGate = RecapQuery.paramGate(

UtegLikedByTweetsParams.UTEGRecommendationsFilter.ExcludeQuoteTweetParam

),

statsReceiver = baseScope

).build

)

private[this] val utegResultsTransform = new UTEGResultsTransform(

userTweetEntityGraphClient,

maxCandidatesToFetchFromUtegProvider,

recommendationsFilter,

socialProofTypes

)

private[this] val earlybirdScoreMultiplierProvider =

DependencyProvider.from(UtegLikedByTweetsParams.EarlybirdScoreMultiplierParam)

private[this] val maxCandidatesToReturnToCallerProvider = DependencyProvider { query =>

query.maxCount.getOrElse(query.params(UtegLikedByTweetsParams.DefaultMaxTweetCount))

}

private[this] val minNumFavedByUserIdsProvider = DependencyProvider { query =>

query.params(UtegLikedByTweetsParams.MinNumFavoritedByUserIdsParam)

}

private[this] val removeTweetsAuthoredBySeedSetForOutOfNetworkPipeline =

FutureArrow.choose[CandidateEnvelope, CandidateEnvelope](

predicate = isInNetwork,

ifTrue = FutureArrow.identity,

ifFalse = new UsersSearchResultMonitoringTransform(

name = "RemoveCandidatesAuthoredByWeightedFollowingsTransform",

RemoveCandidatesAuthoredByWeightedFollowingsTransform,

baseScope,

debugAuthorsMonitoringProvider

)

)

private[this] val minNumFavoritedByUserIdsFilterTransform =

FutureArrow.choose[CandidateEnvelope, CandidateEnvelope](

predicate = isInNetwork,

ifTrue = FutureArrow.identity,

ifFalse = new UsersSearchResultMonitoringTransform(

name = "MinNumNonAuthorFavoritedByUserIdsFilterTransform",

new MinNumNonAuthorFavoritedByUserIdsFilterTransform(

minNumFavoritedByUserIdsProvider = minNumFavedByUserIdsProvider

),

baseScope,

debugAuthorsMonitoringProvider

)

)

private[this] val includeRandomTweetProvider =

DependencyProvider.from(UtegLikedByTweetsParams.IncludeRandomTweetParam)

private[this] val includeSingleRandomTweetProvider =

DependencyProvider.from(UtegLikedByTweetsParams.IncludeSingleRandomTweetParam)

private[this] val probabilityRandomTweetProvider =

DependencyProvider.from(UtegLikedByTweetsParams.ProbabilityRandomTweetParam)

private[this] val markRandomTweetTransform = new MarkRandomTweetTransform(

includeRandomTweetProvider = includeRandomTweetProvider,

includeSingleRandomTweetProvider = includeSingleRandomTweetProvider,

probabilityRandomTweetProvider = probabilityRandomTweetProvider,

)

private[this] val combinedScoreTruncateTransform =

FutureArrow.choose[CandidateEnvelope, CandidateEnvelope](

predicate = isInNetwork,

ifTrue = FutureArrow.identity,

ifFalse = new CombinedScoreAndTruncateTransform(

maxTweetCountProvider = maxCandidatesToReturnToCallerProvider,

earlybirdScoreMultiplierProvider = earlybirdScoreMultiplierProvider,

numAdditionalRepliesProvider =

DependencyProvider.from(UtegLikedByTweetsParams.NumAdditionalRepliesParam),

statsReceiver = baseScope

)

)

private[this] val excludeRecommendedRepliesToNonFollowedUsersGate: Gate[RecapQuery] =

RecapQuery.paramGate(

UtegLikedByTweetsParams.UTEGRecommendationsFilter.ExcludeRecommendedRepliesToNonFollowedUsersParam)

private[this] def enableUseFollowGraphDataForRecommendedReplies(

envelope: CandidateEnvelope

): Boolean =

excludeRecommendedRepliesToNonFollowedUsersGate(envelope.query)

val dynamicHydratedTweetsFilter: FutureArrow[CandidateEnvelope, CandidateEnvelope] =

FutureArrow.choose[CandidateEnvelope, CandidateEnvelope](

predicate = enableUseFollowGraphDataForRecommendedReplies,

ifTrue = new TweetKindOptionHydratedTweetsFilterTransform(

useFollowGraphData = true,

useSourceTweets = false,

statsReceiver = baseScope

),

ifFalse = new TweetKindOptionHydratedTweetsFilterTransform(

useFollowGraphData = false,

useSourceTweets = false,

statsReceiver = baseScope

)

)

private[this] val trimToMatchSearchResultsTransform =

new UsersSearchResultMonitoringTransform(

name = "TrimToMatchSearchResultsTransform",

new TrimToMatchSearchResultsTransform(

hydrateReplyRootTweetProvider = DependencyProvider.False,

statsReceiver = baseScope

),

baseScope,

debugAuthorsMonitoringProvider

)

// combine score and truncate tweet candidates immediately after

private[this] val hydrationAndFilteringPipeline =

CreateCandidateEnvelopeTransform

.andThen(followGraphDataTransform)

.andThen(utegResultsTransform)

.andThen(searchResultsTransform)

// For out of network tweets, remove tweets whose author is contained in the weighted following seed set passed into TLR

.andThen(removeTweetsAuthoredBySeedSetForOutOfNetworkPipeline)

.andThen(minNumFavoritedByUserIdsFilterTransform)

.andThen(CandidateTweetHydrationTransform)

.andThen(markRandomTweetTransform)

.andThen(dynamicHydratedTweetsFilter)

.andThen(TrimToMatchHydratedTweetsTransform)

.andThen(combinedScoreTruncateTransform)

.andThen(trimToMatchSearchResultsTransform)

// runs the main pipeline in parallel with fetching user profile info and user languages

private[this] val featureHydrationDataTransform = new FeatureHydrationDataTransform(

hydrationAndFilteringPipeline,

languagesTransform,

userProfileInfoTransform

)

private[this] val contentFeaturesHydrationTransform =

new ContentFeaturesHydrationTransformBuilder(

tweetyPieClient,

contentFeaturesCache,

enableContentFeaturesGate =

RecapQuery.paramGate(UtegLikedByTweetsParams.EnableContentFeaturesHydrationParam),

enableTokensInContentFeaturesGate =

RecapQuery.paramGate(UtegLikedByTweetsParams.EnableTokensInContentFeaturesHydrationParam),

enableTweetTextInContentFeaturesGate = RecapQuery.paramGate(

UtegLikedByTweetsParams.EnableTweetTextInContentFeaturesHydrationParam),

enableConversationControlContentFeaturesGate = RecapQuery.paramGate(

UtegLikedByTweetsParams.EnableConversationControlInContentFeaturesHydrationParam),

enableTweetMediaHydrationGate = RecapQuery.paramGate(

UtegLikedByTweetsParams.EnableTweetMediaHydrationParam

),

hydrateInReplyToTweets = true,

statsReceiver = baseScope

).build()

// use OutOfNetworkTweetsSearchFeaturesHydrationTransform for rectweets

private[this] val tweetsSearchFeaturesHydrationTransform =

FutureArrow

.choose[HydratedCandidatesAndFeaturesEnvelope, HydratedCandidatesAndFeaturesEnvelope](

predicate = isInNetwork,

ifTrue = InNetworkTweetsSearchFeaturesHydrationTransform,

ifFalse = OutOfNetworkTweetsSearchFeaturesHydrationTransform

)

private[this] def hydratesContentFeatures(

hydratedEnvelope: HydratedCandidatesAndFeaturesEnvelope

): Boolean =

hydratedEnvelope.candidateEnvelope.query.hydratesContentFeatures.getOrElse(true)

private[this] val contentFeaturesTransformer = FutureArrow.choose(

predicate = hydratesContentFeatures,

ifTrue = contentFeaturesHydrationTransform

.andThen(CopyContentFeaturesIntoThriftTweetFeaturesTransform)

.andThen(CopyContentFeaturesIntoHydratedTweetsTransform),

ifFalse = FutureArrow[

HydratedCandidatesAndFeaturesEnvelope,

HydratedCandidatesAndFeaturesEnvelope

](Future.value) // empty transformer

)

private[this] val featureHydrationPipeline =

featureHydrationDataTransform

.andThen(tweetsSearchFeaturesHydrationTransform)

.andThen(SocialProofAndUTEGScoreHydrationTransform)

.andThen(contentFeaturesTransformer)

.andThen(candidateGenerationTransform)

def get(query: RecapQuery): Future[CandidateTweetsResult] = {

requestStats.addEventStats {

featureHydrationPipeline(query)

}

}

def get(queries: Seq[RecapQuery]): Future[Seq[CandidateTweetsResult]] = {

Future.collect(queries.map(get))

}

}