package com.twitter.home\_mixer.util.earlybird

import com.twitter.search.common.constants.{thriftscala => scc}

import com.twitter.search.common.features.{thriftscala => sc}

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant.\_

import com.twitter.search.common.util.lang.ThriftLanguageUtil

import com.twitter.search.earlybird.{thriftscala => eb}

import com.twitter.timelines.earlybird.common.utils.InNetworkEngagement

object EarlybirdResponseUtil {

private[earlybird] val Mentions: String = "mentions"

private[earlybird] val Hashtags: String = "hashtags"

private val CharsToRemoveFromMentions: Set[Char] = "@".toSet

private val CharsToRemoveFromHashtags: Set[Char] = "#".toSet

// Default value of settings of ThriftTweetFeatures.

private[earlybird] val DefaultEarlybirdFeatures: sc.ThriftTweetFeatures = sc.ThriftTweetFeatures()

private[earlybird] val DefaultCount = 0

private[earlybird] val DefaultLanguage = 0

private[earlybird] val DefaultScore = 0.0

private[earlybird] def getTweetCountByAuthorId(

searchResults: Seq[eb.ThriftSearchResult]

): Map[Long, Int] = {

searchResults

.groupBy { result =>

result.metadata.map(\_.fromUserId).getOrElse(0L)

}.mapValues(\_.size).withDefaultValue(0)

}

private[earlybird] def getLanguage(uiLanguageCode: Option[String]): Option[scc.ThriftLanguage] = {

uiLanguageCode.flatMap { languageCode =>

scc.ThriftLanguage.get(ThriftLanguageUtil.getThriftLanguageOf(languageCode).getValue)

}

}

private def getMentions(result: eb.ThriftSearchResult): Seq[String] = {

val facetLabels = result.metadata.flatMap(\_.facetLabels).getOrElse(Seq.empty)

getFacets(facetLabels, Mentions, CharsToRemoveFromMentions)

}

private def getHashtags(result: eb.ThriftSearchResult): Seq[String] = {

val facetLabels = result.metadata.flatMap(\_.facetLabels).getOrElse(Seq.empty)

getFacets(facetLabels, Hashtags, CharsToRemoveFromHashtags)

}

private def getFacets(

facetLabels: Seq[eb.ThriftFacetLabel],

facetName: String,

charsToRemove: Set[Char]

): Seq[String] = {

facetLabels.filter(\_.fieldName == facetName).map(\_.label.filterNot(charsToRemove))

}

private def isUserMentioned(

screenName: Option[String],

mentions: Seq[String],

mentionsInSourceTweet: Seq[String]

): Boolean =

isUserMentioned(screenName, mentions) || isUserMentioned(screenName, mentionsInSourceTweet)

private def isUserMentioned(

screenName: Option[String],

mentions: Seq[String]

): Boolean = {

screenName

.exists { screenName => mentions.exists(\_.equalsIgnoreCase(screenName)) }

}

private[earlybird] def isUsersMainLanguage(

tweetLanguage: scc.ThriftLanguage,

userLanguages: Seq[scc.ThriftLanguage]

): Boolean = {

(tweetLanguage != scc.ThriftLanguage.Unknown) && userLanguages.headOption.contains(

tweetLanguage)

}

private[earlybird] def isUsersLanguage(

tweetLanguage: scc.ThriftLanguage,

userLanguages: Seq[scc.ThriftLanguage]

): Boolean = {

(tweetLanguage != scc.ThriftLanguage.Unknown) && userLanguages.contains(tweetLanguage)

}

private[earlybird] def isUILanguage(

tweetLanguage: scc.ThriftLanguage,

uiLanguage: Option[scc.ThriftLanguage]

): Boolean = {

(tweetLanguage != scc.ThriftLanguage.Unknown) && uiLanguage.contains(tweetLanguage)

}

private def getBooleanOptFeature(

featureName: EarlybirdFieldConstant,

resultMapOpt: Option[scala.collection.Map[Int, Boolean]],

defaultValue: Boolean = false,

): Option[Boolean] = {

resultMapOpt.map {

\_.getOrElse(featureName.getFieldId, defaultValue)

}

}

private def getDoubleAsIntOptFeature(

featureName: EarlybirdFieldConstant,

resultMapOpt: Option[scala.collection.Map[Int, Double]]

): Option[Int] = {

if (resultMapOpt.exists(\_.contains(featureName.getFieldId)))

resultMapOpt

.map {

\_.get(featureName.getFieldId)

}

.flatMap { doubleValue =>

doubleValue.map(\_.toInt)

}

else

None

}

private def getIntOptFeature(

featureName: EarlybirdFieldConstant,

resultMapOpt: Option[scala.collection.Map[Int, Int]]

): Option[Int] = {

if (resultMapOpt.exists(\_.contains(featureName.getFieldId)))

resultMapOpt.flatMap {

\_.get(featureName.getFieldId)

}

else

None

}

def getTweetThriftFeaturesByTweetId(

searcherUserId: Long,

screenName: Option[String],

userLanguages: Seq[scc.ThriftLanguage],

uiLanguageCode: Option[String] = None,

followedUserIds: Set[Long],

mutuallyFollowingUserIds: Set[Long],

searchResults: Seq[eb.ThriftSearchResult],

sourceTweetSearchResults: Seq[eb.ThriftSearchResult],

): Map[Long, sc.ThriftTweetFeatures] = {

val allSearchResults = searchResults ++ sourceTweetSearchResults

val sourceTweetSearchResultById =

sourceTweetSearchResults.map(result => (result.id -> result)).toMap

val inNetworkEngagement =

InNetworkEngagement(followedUserIds.toSeq, mutuallyFollowingUserIds, allSearchResults)

searchResults.map { searchResult =>

val features = getThriftTweetFeaturesFromSearchResult(

searcherUserId,

screenName,

userLanguages,

getLanguage(uiLanguageCode),

getTweetCountByAuthorId(searchResults),

followedUserIds,

mutuallyFollowingUserIds,

sourceTweetSearchResultById,

inNetworkEngagement,

searchResult

)

(searchResult.id -> features)

}.toMap

}

private[earlybird] def getThriftTweetFeaturesFromSearchResult(

searcherUserId: Long,

screenName: Option[String],

userLanguages: Seq[scc.ThriftLanguage],

uiLanguage: Option[scc.ThriftLanguage],

tweetCountByAuthorId: Map[Long, Int],

followedUserIds: Set[Long],

mutuallyFollowingUserIds: Set[Long],

sourceTweetSearchResultById: Map[Long, eb.ThriftSearchResult],

inNetworkEngagement: InNetworkEngagement,

searchResult: eb.ThriftSearchResult,

): sc.ThriftTweetFeatures = {

val applyFeatures = (applyUserIndependentFeatures(

searchResult

)(\_)).andThen(

applyUserDependentFeatures(

searcherUserId,

screenName,

userLanguages,

uiLanguage,

tweetCountByAuthorId,

followedUserIds,

mutuallyFollowingUserIds,

sourceTweetSearchResultById,

inNetworkEngagement,

searchResult

)(\_)

)

val tweetFeatures = searchResult.tweetFeatures.getOrElse(DefaultEarlybirdFeatures)

applyFeatures(tweetFeatures)

}

private[earlybird] def applyUserIndependentFeatures(

result: eb.ThriftSearchResult

)(

thriftTweetFeatures: sc.ThriftTweetFeatures

): sc.ThriftTweetFeatures = {

val features = result.metadata

.map { metadata =>

val isRetweet = metadata.isRetweet.getOrElse(false)

val isReply = metadata.isReply.getOrElse(false)

// Facets.

val mentions = getMentions(result)

val hashtags = getHashtags(result)

val searchResultSchemaFeatures = metadata.extraMetadata.flatMap(\_.features)

val booleanSearchResultSchemaFeatures = searchResultSchemaFeatures.flatMap(\_.boolValues)

val intSearchResultSchemaFeatures = searchResultSchemaFeatures.flatMap(\_.intValues)

val doubleSearchResultSchemaFeatures = searchResultSchemaFeatures.flatMap(\_.doubleValues)

thriftTweetFeatures.copy(

// Info about the Tweet.

isRetweet = isRetweet,

isOffensive = metadata.isOffensive.getOrElse(false),

isReply = isReply,

fromVerifiedAccount = metadata.fromVerifiedAccount.getOrElse(false),

cardType = metadata.cardType,

signature = metadata.signature,

language = metadata.language,

isAuthorNSFW = metadata.isUserNSFW.getOrElse(false),

isAuthorBot = metadata.isUserBot.getOrElse(false),

isAuthorSpam = metadata.isUserSpam.getOrElse(false),

isSensitiveContent =

metadata.extraMetadata.flatMap(\_.isSensitiveContent).getOrElse(false),

isAuthorProfileEgg = metadata.extraMetadata.flatMap(\_.profileIsEggFlag).getOrElse(false),

isAuthorNew = metadata.extraMetadata.flatMap(\_.isUserNewFlag).getOrElse(false),

linkLanguage = metadata.extraMetadata.flatMap(\_.linkLanguage).getOrElse(DefaultLanguage),

// Info about Tweet content/media.

hasCard = metadata.hasCard.getOrElse(false),

hasImage = metadata.hasImage.getOrElse(false),

hasNews = metadata.hasNews.getOrElse(false),

hasVideo = metadata.hasVideo.getOrElse(false),

hasConsumerVideo = metadata.hasConsumerVideo.getOrElse(false),

hasProVideo = metadata.hasProVideo.getOrElse(false),

hasVine = metadata.hasVine.getOrElse(false),

hasPeriscope = metadata.hasPeriscope.getOrElse(false),

hasNativeVideo = metadata.hasNativeVideo.getOrElse(false),

hasNativeImage = metadata.hasNativeImage.getOrElse(false),

hasLink = metadata.hasLink.getOrElse(false),

hasVisibleLink = metadata.hasVisibleLink.getOrElse(false),

hasTrend = metadata.hasTrend.getOrElse(false),

hasMultipleHashtagsOrTrends = metadata.hasMultipleHashtagsOrTrends.getOrElse(false),

hasQuote = metadata.extraMetadata.flatMap(\_.hasQuote),

urlsList = metadata.tweetUrls.map {

\_.map(\_.originalUrl)

},

hasMultipleMedia =

metadata.extraMetadata.flatMap(\_.hasMultipleMediaFlag).getOrElse(false),

visibleTokenRatio = getIntOptFeature(VISIBLE\_TOKEN\_RATIO, intSearchResultSchemaFeatures),

// Various counts.

favCount = metadata.favCount.getOrElse(DefaultCount),

replyCount = metadata.replyCount.getOrElse(DefaultCount),

retweetCount = metadata.retweetCount.getOrElse(DefaultCount),

quoteCount = metadata.extraMetadata.flatMap(\_.quotedCount),

embedsImpressionCount = metadata.embedsImpressionCount.getOrElse(DefaultCount),

embedsUrlCount = metadata.embedsUrlCount.getOrElse(DefaultCount),

videoViewCount = metadata.videoViewCount.getOrElse(DefaultCount),

numMentions = metadata.extraMetadata.flatMap(\_.numMentions).getOrElse(DefaultCount),

numHashtags = metadata.extraMetadata.flatMap(\_.numHashtags).getOrElse(DefaultCount),

favCountV2 = metadata.extraMetadata.flatMap(\_.favCountV2),

replyCountV2 = metadata.extraMetadata.flatMap(\_.replyCountV2),

retweetCountV2 = metadata.extraMetadata.flatMap(\_.retweetCountV2),

weightedFavoriteCount = metadata.extraMetadata.flatMap(\_.weightedFavCount),

weightedReplyCount = metadata.extraMetadata.flatMap(\_.weightedReplyCount),

weightedRetweetCount = metadata.extraMetadata.flatMap(\_.weightedRetweetCount),

weightedQuoteCount = metadata.extraMetadata.flatMap(\_.weightedQuoteCount),

embedsImpressionCountV2 =

getDoubleAsIntOptFeature(EMBEDS\_IMPRESSION\_COUNT\_V2, doubleSearchResultSchemaFeatures),

embedsUrlCountV2 =

getDoubleAsIntOptFeature(EMBEDS\_URL\_COUNT\_V2, doubleSearchResultSchemaFeatures),

decayedFavoriteCount =

getDoubleAsIntOptFeature(DECAYED\_FAVORITE\_COUNT, doubleSearchResultSchemaFeatures),

decayedRetweetCount =

getDoubleAsIntOptFeature(DECAYED\_RETWEET\_COUNT, doubleSearchResultSchemaFeatures),

decayedReplyCount =

getDoubleAsIntOptFeature(DECAYED\_REPLY\_COUNT, doubleSearchResultSchemaFeatures),

decayedQuoteCount =

getDoubleAsIntOptFeature(DECAYED\_QUOTE\_COUNT, doubleSearchResultSchemaFeatures),

fakeFavoriteCount =

getDoubleAsIntOptFeature(FAKE\_FAVORITE\_COUNT, doubleSearchResultSchemaFeatures),

fakeRetweetCount =

getDoubleAsIntOptFeature(FAKE\_RETWEET\_COUNT, doubleSearchResultSchemaFeatures),

fakeReplyCount =

getDoubleAsIntOptFeature(FAKE\_REPLY\_COUNT, doubleSearchResultSchemaFeatures),

fakeQuoteCount =

getDoubleAsIntOptFeature(FAKE\_QUOTE\_COUNT, doubleSearchResultSchemaFeatures),

// Scores.

textScore = metadata.textScore.getOrElse(DefaultScore),

earlybirdScore = metadata.score.getOrElse(DefaultScore),

parusScore = metadata.parusScore.getOrElse(DefaultScore),

userRep = metadata.userRep.getOrElse(DefaultScore),

pBlockScore = metadata.extraMetadata.flatMap(\_.pBlockScore),

toxicityScore = metadata.extraMetadata.flatMap(\_.toxicityScore),

pSpammyTweetScore = metadata.extraMetadata.flatMap(\_.pSpammyTweetScore),

pReportedTweetScore = metadata.extraMetadata.flatMap(\_.pReportedTweetScore),

pSpammyTweetContent = metadata.extraMetadata.flatMap(\_.spammyTweetContentScore),

// Safety Signals

labelAbusiveFlag =

getBooleanOptFeature(LABEL\_ABUSIVE\_FLAG, booleanSearchResultSchemaFeatures),

labelAbusiveHiRclFlag =

getBooleanOptFeature(LABEL\_ABUSIVE\_HI\_RCL\_FLAG, booleanSearchResultSchemaFeatures),

labelDupContentFlag =

getBooleanOptFeature(LABEL\_DUP\_CONTENT\_FLAG, booleanSearchResultSchemaFeatures),

labelNsfwHiPrcFlag =

getBooleanOptFeature(LABEL\_NSFW\_HI\_PRC\_FLAG, booleanSearchResultSchemaFeatures),

labelNsfwHiRclFlag =

getBooleanOptFeature(LABEL\_NSFW\_HI\_RCL\_FLAG, booleanSearchResultSchemaFeatures),

labelSpamFlag = getBooleanOptFeature(LABEL\_SPAM\_FLAG, booleanSearchResultSchemaFeatures),

labelSpamHiRclFlag =

getBooleanOptFeature(LABEL\_SPAM\_HI\_RCL\_FLAG, booleanSearchResultSchemaFeatures),

// Periscope Features

periscopeExists =

getBooleanOptFeature(PERISCOPE\_EXISTS, booleanSearchResultSchemaFeatures),

periscopeHasBeenFeatured =

getBooleanOptFeature(PERISCOPE\_HAS\_BEEN\_FEATURED, booleanSearchResultSchemaFeatures),

periscopeIsCurrentlyFeatured = getBooleanOptFeature(

PERISCOPE\_IS\_CURRENTLY\_FEATURED,

booleanSearchResultSchemaFeatures),

periscopeIsFromQualitySource = getBooleanOptFeature(

PERISCOPE\_IS\_FROM\_QUALITY\_SOURCE,

booleanSearchResultSchemaFeatures),

periscopeIsLive =

getBooleanOptFeature(PERISCOPE\_IS\_LIVE, booleanSearchResultSchemaFeatures),

// Last Engagement Features

lastFavSinceCreationHrs =

getIntOptFeature(LAST\_FAVORITE\_SINCE\_CREATION\_HRS, intSearchResultSchemaFeatures),

lastRetweetSinceCreationHrs =

getIntOptFeature(LAST\_RETWEET\_SINCE\_CREATION\_HRS, intSearchResultSchemaFeatures),

lastReplySinceCreationHrs =

getIntOptFeature(LAST\_REPLY\_SINCE\_CREATION\_HRS, intSearchResultSchemaFeatures),

lastQuoteSinceCreationHrs =

getIntOptFeature(LAST\_QUOTE\_SINCE\_CREATION\_HRS, intSearchResultSchemaFeatures),

likedByUserIds = metadata.extraMetadata.flatMap(\_.likedByUserIds),

mentionsList = if (mentions.nonEmpty) Some(mentions) else None,

hashtagsList = if (hashtags.nonEmpty) Some(hashtags) else None,

isComposerSourceCamera =

getBooleanOptFeature(COMPOSER\_SOURCE\_IS\_CAMERA\_FLAG, booleanSearchResultSchemaFeatures),

)

}

.getOrElse(thriftTweetFeatures)

features

}

private def applyUserDependentFeatures(

searcherUserId: Long,

screenName: Option[String],

userLanguages: Seq[scc.ThriftLanguage],

uiLanguage: Option[scc.ThriftLanguage],

tweetCountByAuthorId: Map[Long, Int],

followedUserIds: Set[Long],

mutuallyFollowingUserIds: Set[Long],

sourceTweetSearchResultById: Map[Long, eb.ThriftSearchResult],

inNetworkEngagement: InNetworkEngagement,

result: eb.ThriftSearchResult

)(

thriftTweetFeatures: sc.ThriftTweetFeatures

): sc.ThriftTweetFeatures = {

result.metadata

.map { metadata =>

val isRetweet = metadata.isRetweet.getOrElse(false)

val sourceTweet =

if (isRetweet) sourceTweetSearchResultById.get(metadata.sharedStatusId)

else None

val mentionsInSourceTweet = sourceTweet.map(getMentions).getOrElse(Seq.empty)

val isReply = metadata.isReply.getOrElse(false)

val replyToSearcher = isReply && (metadata.referencedTweetAuthorId == searcherUserId)

val replyOther = isReply && !replyToSearcher

val retweetOther = isRetweet && (metadata.referencedTweetAuthorId != searcherUserId)

val tweetLanguage = metadata.language.getOrElse(scc.ThriftLanguage.Unknown)

val referencedTweetAuthorId =

if (metadata.referencedTweetAuthorId > 0) Some(metadata.referencedTweetAuthorId) else None

val inReplyToUserId = if (!isRetweet) referencedTweetAuthorId else None

thriftTweetFeatures.copy(

// Info about the Tweet.

fromSearcher = metadata.fromUserId == searcherUserId,

probablyFromFollowedAuthor = followedUserIds.contains(metadata.fromUserId),

fromMutualFollow = mutuallyFollowingUserIds.contains(metadata.fromUserId),

replySearcher = replyToSearcher,

replyOther = replyOther,

retweetOther = retweetOther,

mentionSearcher = isUserMentioned(screenName, getMentions(result), mentionsInSourceTweet),

// Info about Tweet content/media.

matchesSearcherMainLang = isUsersMainLanguage(tweetLanguage, userLanguages),

matchesSearcherLangs = isUsersLanguage(tweetLanguage, userLanguages),

matchesUILang = isUILanguage(tweetLanguage, uiLanguage),

// Various counts.

prevUserTweetEngagement =

metadata.extraMetadata.flatMap(\_.prevUserTweetEngagement).getOrElse(DefaultCount),

tweetCountFromUserInSnapshot = tweetCountByAuthorId(metadata.fromUserId),

bidirectionalReplyCount = inNetworkEngagement.biDirectionalReplyCounts(result.id),

unidirectionalReplyCount = inNetworkEngagement.uniDirectionalReplyCounts(result.id),

bidirectionalRetweetCount = inNetworkEngagement.biDirectionalRetweetCounts(result.id),

unidirectionalRetweetCount = inNetworkEngagement.uniDirectionalRetweetCounts(result.id),

conversationCount = inNetworkEngagement.descendantReplyCounts(result.id),

directedAtUserIdIsInFirstDegree =

if (isReply) inReplyToUserId.map(followedUserIds.contains) else None,

)

}

.getOrElse(thriftTweetFeatures)

}

}