package com.twitter.timelineranker.util

import com.twitter.search.earlybird.thriftscala.ThriftSearchResult

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.model.UserId

object SearchResultUtil {

val DefaultScore = 0.0

def getScore(result: ThriftSearchResult): Double = {

result.metadata.flatMap(\_.score).filterNot(\_.isNaN).getOrElse(DefaultScore)

}

def isRetweet(result: ThriftSearchResult): Boolean = {

result.metadata.flatMap(\_.isRetweet).getOrElse(false)

}

def isReply(result: ThriftSearchResult): Boolean = {

result.metadata.flatMap(\_.isReply).getOrElse(false)

}

def isEligibleReply(result: ThriftSearchResult): Boolean = {

isReply(result) && !isRetweet(result)

}

def authorId(result: ThriftSearchResult): Option[UserId] = {

// fromUserId defaults to 0L if unset. None is cleaner

result.metadata.map(\_.fromUserId).filter(\_ != 0L)

}

def referencedTweetAuthorId(result: ThriftSearchResult): Option[UserId] = {

// referencedTweetAuthorId defaults to 0L by default. None is cleaner

result.metadata.map(\_.referencedTweetAuthorId).filter(\_ != 0L)

}

/\*\*

\* Extended replies are replies, that are not retweets (see below), from a followed userId

\* towards a non-followed userId.

\*

\* In Thrift SearchResult it is possible to have both isRetweet and isReply set to true,

\* in the case of the retweeted reply. This is confusing edge case as the retweet object

\* is not itself a reply, but the original tweet is reply.

\*/

def isExtendedReply(followedUserIds: Seq[UserId])(result: ThriftSearchResult): Boolean = {

isEligibleReply(result) &&

authorId(result).exists(followedUserIds.contains(\_)) && // author is followed

referencedTweetAuthorId(result).exists(!followedUserIds.contains(\_)) // referenced author is not

}

/\*\*

\* If a tweet is a reply that is not a retweet, and both the user follows both the reply author

\* and the reply parent's author

\*/

def isInNetworkReply(followedUserIds: Seq[UserId])(result: ThriftSearchResult): Boolean = {

isEligibleReply(result) &&

authorId(result).exists(followedUserIds.contains(\_)) && // author is followed

referencedTweetAuthorId(result).exists(followedUserIds.contains(\_)) // referenced author is

}

/\*\*

\* If a tweet is a retweet, and user follows author of outside tweet but not following author of

\* source/inner tweet. This tweet is also called oon-retweet

\*/

def isOutOfNetworkRetweet(followedUserIds: Seq[UserId])(result: ThriftSearchResult): Boolean = {

isRetweet(result) &&

authorId(result).exists(followedUserIds.contains(\_)) && // author is followed

referencedTweetAuthorId(result).exists(!followedUserIds.contains(\_)) // referenced author is not

}

/\*\*

\* From official documentation in thrift on sharedStatusId:

\* When isRetweet (or packed features equivalent) is true, this is the status id of the

\* original tweet. When isReply and getReplySource are true, this is the status id of the

\* original tweet. In all other circumstances this is 0.

\*

\* If a tweet is a retweet of a reply, this is the status id of the reply (the original tweet

\* of the retweet), not the reply's in-reply-to tweet status id.

\*/

def getSourceTweetId(result: ThriftSearchResult): Option[TweetId] = {

result.metadata.map(\_.sharedStatusId).filter(\_ != 0L)

}

def getRetweetSourceTweetId(result: ThriftSearchResult): Option[TweetId] = {

if (isRetweet(result)) {

getSourceTweetId(result)

} else {

None

}

}

def getInReplyToTweetId(result: ThriftSearchResult): Option[TweetId] = {

if (isReply(result)) {

getSourceTweetId(result)

} else {

None

}

}

def getReplyRootTweetId(result: ThriftSearchResult): Option[TweetId] = {

if (isEligibleReply(result)) {

for {

meta <- result.metadata

extraMeta <- meta.extraMetadata

conversationId <- extraMeta.conversationId

} yield {

conversationId

}

} else {

None

}

}

/\*\*

\* For retweet: selfTweetId + sourceTweetId, (however selfTweetId is redundant here, since Health

\* score retweet by tweetId == sourceTweetId)

\* For replies: selfTweetId + immediate ancestor tweetId + root ancestor tweetId.

\* Use set to de-duplicate the case when source tweet == root tweet. (like A->B, B is root and source).

\*/

def getOriginalTweetIdAndAncestorTweetIds(searchResult: ThriftSearchResult): Set[TweetId] = {

Set(searchResult.id) ++

SearchResultUtil.getSourceTweetId(searchResult).toSet ++

SearchResultUtil.getReplyRootTweetId(searchResult).toSet

}

}