package com.twitter.home\_mixer.util

import com.twitter.home\_mixer.model.HomeFeatures.CachedScoredTweetsFeature

import com.twitter.home\_mixer.{thriftscala => hmt}

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.model.common.identifier.CandidatePipelineIdentifier

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.util.Time

object CachedScoredTweetsHelper {

def tweetImpressionsAndCachedScoredTweets(

features: FeatureMap,

candidatePipelineIdentifier: CandidatePipelineIdentifier

): Seq[Long] = {

val tweetImpressions = TweetImpressionsHelper.tweetImpressions(features)

val cachedScoredTweets = features

.getOrElse(CachedScoredTweetsFeature, Seq.empty)

.filter { tweet =>

tweet.candidatePipelineIdentifier.exists(

CandidatePipelineIdentifier(\_).equals(candidatePipelineIdentifier))

}.map(\_.tweetId)

(tweetImpressions ++ cachedScoredTweets).toSeq

}

def tweetImpressionsAndCachedScoredTweetsInRange(

features: FeatureMap,

candidatePipelineIdentifier: CandidatePipelineIdentifier,

maxNumImpressions: Int,

sinceTime: Time,

untilTime: Time

): Seq[Long] =

tweetImpressionsAndCachedScoredTweets(features, candidatePipelineIdentifier)

.filter { tweetId => SnowflakeId.isSnowflakeId(tweetId) }

.filter { tweetId =>

val creationTime = SnowflakeId.timeFromId(tweetId)

sinceTime <= creationTime && untilTime >= creationTime

}.take(maxNumImpressions)

def unseenCachedScoredTweets(

features: FeatureMap

): Seq[hmt.ScoredTweet] = {

val seenTweetIds = TweetImpressionsHelper.tweetImpressions(features)

features

.getOrElse(CachedScoredTweetsFeature, Seq.empty)

.filter(tweet => !seenTweetIds.contains(tweet.tweetId))

}

}