package com.twitter.cr\_mixer.similarity\_engine

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.search.earlybird.thriftscala.EarlybirdRequest

import com.twitter.search.earlybird.thriftscala.EarlybirdService

import com.twitter.search.earlybird.thriftscala.ThriftSearchQuery

import com.twitter.util.Time

import com.twitter.search.common.query.thriftjava.thriftscala.CollectorParams

import com.twitter.search.common.ranking.thriftscala.ThriftRankingParams

import com.twitter.search.common.ranking.thriftscala.ThriftScoringFunctionType

import com.twitter.search.earlybird.thriftscala.ThriftSearchRelevanceOptions

import javax.inject.Inject

import javax.inject.Singleton

import EarlybirdSimilarityEngineBase.\_

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.similarity\_engine.EarlybirdTensorflowBasedSimilarityEngine.EarlybirdTensorflowBasedSearchQuery

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.EarlybirdClientId

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.FacetsToFetch

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.GetCollectorTerminationParams

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.GetEarlybirdQuery

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.MetadataOptions

import com.twitter.cr\_mixer.util.EarlybirdSearchUtil.GetNamedDisjunctions

import com.twitter.search.earlybird.thriftscala.ThriftSearchRankingMode

import com.twitter.simclusters\_v2.common.TweetId

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.util.Duration

@Singleton

case class EarlybirdTensorflowBasedSimilarityEngine @Inject() (

earlybirdSearchClient: EarlybirdService.MethodPerEndpoint,

timeoutConfig: TimeoutConfig,

stats: StatsReceiver)

extends EarlybirdSimilarityEngineBase[EarlybirdTensorflowBasedSearchQuery] {

import EarlybirdTensorflowBasedSimilarityEngine.\_

override val statsReceiver: StatsReceiver = stats.scope(this.getClass.getSimpleName)

override def getEarlybirdRequest(

query: EarlybirdTensorflowBasedSearchQuery

): Option[EarlybirdRequest] = {

if (query.seedUserIds.nonEmpty)

Some(

EarlybirdRequest(

searchQuery = getThriftSearchQuery(query, timeoutConfig.earlybirdServerTimeout),

clientHost = None,

clientRequestID = None,

clientId = Some(EarlybirdClientId),

clientRequestTimeMs = Some(Time.now.inMilliseconds),

cachingParams = None,

timeoutMs = timeoutConfig.earlybirdServerTimeout.inMilliseconds.intValue(),

facetRequest = None,

termStatisticsRequest = None,

debugMode = 0,

debugOptions = None,

searchSegmentId = None,

returnStatusType = None,

successfulResponseThreshold = None,

querySource = None,

getOlderResults = Some(false),

followedUserIds = Some(query.seedUserIds),

adjustedProtectedRequestParams = None,

adjustedFullArchiveRequestParams = None,

getProtectedTweetsOnly = Some(false),

retokenizeSerializedQuery = None,

skipVeryRecentTweets = true,

experimentClusterToUse = None

))

else None

}

}

object EarlybirdTensorflowBasedSimilarityEngine {

case class EarlybirdTensorflowBasedSearchQuery(

searcherUserId: Option[UserId],

seedUserIds: Seq[UserId],

maxNumTweets: Int,

beforeTweetIdExclusive: Option[TweetId],

afterTweetIdExclusive: Option[TweetId],

filterOutRetweetsAndReplies: Boolean,

useTensorflowRanking: Boolean,

excludedTweetIds: Set[TweetId],

maxNumHitsPerShard: Int)

extends EarlybirdSearchQuery

private def getThriftSearchQuery(

query: EarlybirdTensorflowBasedSearchQuery,

processingTimeout: Duration

): ThriftSearchQuery =

ThriftSearchQuery(

serializedQuery = GetEarlybirdQuery(

query.beforeTweetIdExclusive,

query.afterTweetIdExclusive,

query.excludedTweetIds,

query.filterOutRetweetsAndReplies).map(\_.serialize),

fromUserIDFilter64 = Some(query.seedUserIds),

numResults = query.maxNumTweets,

// Whether to collect conversation IDs. Remove it for now.

// collectConversationId = Gate.True(), // true for Home

rankingMode = ThriftSearchRankingMode.Relevance,

relevanceOptions = Some(getRelevanceOptions),

collectorParams = Some(

CollectorParams(

// numResultsToReturn defines how many results each EB shard will return to search root

numResultsToReturn = 1000,

// terminationParams.maxHitsToProcess is used for early terminating per shard results fetching.

terminationParams =

GetCollectorTerminationParams(query.maxNumHitsPerShard, processingTimeout)

)),

facetFieldNames = Some(FacetsToFetch),

resultMetadataOptions = Some(MetadataOptions),

searcherId = query.searcherUserId,

searchStatusIds = None,

namedDisjunctionMap = GetNamedDisjunctions(query.excludedTweetIds)

)

// The specific values of recap relevance/reranking options correspond to

// experiment: enable\_recap\_reranking\_2988,timeline\_internal\_disable\_recap\_filter

// bucket : enable\_rerank,disable\_filter

private def getRelevanceOptions: ThriftSearchRelevanceOptions = {

ThriftSearchRelevanceOptions(

proximityScoring = true,

maxConsecutiveSameUser = Some(2),

rankingParams = Some(getTensorflowBasedRankingParams),

maxHitsToProcess = Some(500),

maxUserBlendCount = Some(3),

proximityPhraseWeight = 9.0,

returnAllResults = Some(true)

)

}

private def getTensorflowBasedRankingParams: ThriftRankingParams = {

ThriftRankingParams(

`type` = Some(ThriftScoringFunctionType.TensorflowBased),

selectedTensorflowModel = Some("timelines\_rectweet\_replica"),

minScore = -1.0e100,

applyBoosts = false,

authorSpecificScoreAdjustments = None

)

}

}