package com.twitter.simclustersann.candidate\_source

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Stats

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

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

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

import com.twitter.simclusters\_v2.thriftscala.SimClustersEmbeddingId

import com.twitter.simclustersann.thriftscala.SimClustersANNConfig

import com.twitter.simclustersann.thriftscala.SimClustersANNTweetCandidate

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

/\*\*

\* This store looks for tweets whose similarity is close to a Source SimClustersEmbeddingId.

\*

\* Approximate cosine similarity is the core algorithm to drive this store.

\*

\* Step 1 - 4 are in "fetchCandidates" method.

\* 1. Retrieve the SimClusters Embedding by the SimClustersEmbeddingId

\* 2. Fetch top N clusters' top tweets from the clusterTweetCandidatesStore (TopTweetsPerCluster index).

\* 3. Calculate all the tweet candidates' dot-product or approximate cosine similarity to source tweets.

\* 4. Take top M tweet candidates by the step 3's score

\*/

case class SimClustersANNCandidateSource(

approximateCosineSimilarity: ApproximateCosineSimilarity,

clusterTweetCandidatesStore: ReadableStore[ClusterId, Seq[(TweetId, Double)]],

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding],

statsReceiver: StatsReceiver) {

private val stats = statsReceiver.scope(this.getClass.getName)

private val fetchSourceEmbeddingStat = stats.scope("fetchSourceEmbedding")

private val fetchCandidatesStat = stats.scope("fetchCandidates")

private val candidateScoresStat = stats.stat("candidateScoresMap")

def get(

query: SimClustersANNCandidateSource.Query

): Future[Option[Seq[SimClustersANNTweetCandidate]]] = {

val sourceEmbeddingId = query.sourceEmbeddingId

val config = query.config

for {

maybeSimClustersEmbedding <- Stats.track(fetchSourceEmbeddingStat) {

simClustersEmbeddingStore.get(query.sourceEmbeddingId)

}

maybeFilteredCandidates <- maybeSimClustersEmbedding match {

case Some(sourceEmbedding) =>

for {

candidates <- Stats.trackSeq(fetchCandidatesStat) {

fetchCandidates(sourceEmbeddingId, sourceEmbedding, config)

}

} yield {

fetchCandidatesStat

.stat(sourceEmbeddingId.embeddingType.name, sourceEmbeddingId.modelVersion.name).add(

candidates.size)

Some(candidates)

}

case None =>

fetchCandidatesStat

.stat(sourceEmbeddingId.embeddingType.name, sourceEmbeddingId.modelVersion.name).add(0)

Future.None

}

} yield {

maybeFilteredCandidates

}

}

private def fetchCandidates(

sourceEmbeddingId: SimClustersEmbeddingId,

sourceEmbedding: SimClustersEmbedding,

config: SimClustersANNConfig

): Future[Seq[SimClustersANNTweetCandidate]] = {

val clusterIds =

sourceEmbedding

.truncate(config.maxScanClusters).getClusterIds()

.toSet

Future

.collect {

clusterTweetCandidatesStore.multiGet(clusterIds)

}.map { clusterTweetsMap =>

approximateCosineSimilarity(

sourceEmbedding = sourceEmbedding,

sourceEmbeddingId = sourceEmbeddingId,

config = config,

candidateScoresStat = (i: Int) => candidateScoresStat.add(i),

clusterTweetsMap = clusterTweetsMap

).map {

case (tweetId, score) =>

SimClustersANNTweetCandidate(

tweetId = tweetId,

score = score

)

}

}

}

}

object SimClustersANNCandidateSource {

case class Query(

sourceEmbeddingId: SimClustersEmbeddingId,

config: SimClustersANNConfig)

}