package com.twitter.simclusters\_v2.candidate\_source

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateSource

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

import com.twitter.simclusters\_v2.candidate\_source.HeavyRanker.UniformScoreStoreRanker

import com.twitter.simclusters\_v2.candidate\_source.SimClustersANNCandidateSource.SimClustersANNConfig

import com.twitter.simclusters\_v2.candidate\_source.SimClustersANNCandidateSource.SimClustersTweetCandidate

import com.twitter.simclusters\_v2.common.ModelVersions.\_

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.summingbird.stores.ClusterKey

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

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

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

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

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

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

import com.twitter.simclusters\_v2.thriftscala.{Score => ThriftScore}

import com.twitter.simclusters\_v2.thriftscala.{ScoreId => ThriftScoreId}

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import scala.collection.mutable

/\*\*

\* 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

\* Step 5-6 are in "reranking" method.

\* 5. Calculate the similarity score between source and candidates.

\* 6. Return top N candidates by the step 5's score.

\*

\* Warning: Only turn off the step 5 for User InterestedIn candidate generation. It's the only use

\* case in Recos that we use dot-product to rank the tweet candidates.

\*/

case class SimClustersANNCandidateSource(

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

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding],

heavyRanker: HeavyRanker.HeavyRanker,

configs: Map[EmbeddingType, SimClustersANNConfig],

statsReceiver: StatsReceiver)

extends CandidateSource[SimClustersANNCandidateSource.Query, SimClustersTweetCandidate] {

import SimClustersANNCandidateSource.\_

override val name: String = this.getClass.getName

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

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

protected val fetchCandidateEmbeddingsStat = stats.scope("fetchCandidateEmbeddings")

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

private val rerankingStat = stats.scope("reranking")

override def get(

query: SimClustersANNCandidateSource.Query

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

val sourceEmbeddingId = query.sourceEmbeddingId

loadConfig(query) match {

case Some(config) =>

for {

maybeSimClustersEmbedding <- Stats.track(fetchSourceEmbeddingStat) {

simClustersEmbeddingStore.get(query.sourceEmbeddingId)

}

maybeFilteredCandidates <- maybeSimClustersEmbedding match {

case Some(sourceEmbedding) =>

for {

rawCandidates <- Stats.trackSeq(fetchCandidatesStat) {

fetchCandidates(sourceEmbeddingId, config, sourceEmbedding)

}

rankedCandidates <- Stats.trackSeq(rerankingStat) {

reranking(sourceEmbeddingId, config, rawCandidates)

}

} yield {

fetchCandidatesStat

.stat(

sourceEmbeddingId.embeddingType.name,

sourceEmbeddingId.modelVersion.name).add(rankedCandidates.size)

Some(rankedCandidates)

}

case None =>

fetchCandidatesStat

.stat(

sourceEmbeddingId.embeddingType.name,

sourceEmbeddingId.modelVersion.name).add(0)

Future.None

}

} yield {

maybeFilteredCandidates

}

case \_ =>

// Skip over queries whose config is not defined

Future.None

}

}

private def fetchCandidates(

sourceEmbeddingId: SimClustersEmbeddingId,

config: SimClustersANNConfig,

sourceEmbedding: SimClustersEmbedding

): Future[Seq[SimClustersTweetCandidate]] = {

val now = Time.now

val earliestTweetId = SnowflakeId.firstIdFor(now - config.maxTweetCandidateAge)

val latestTweetId = SnowflakeId.firstIdFor(now - config.minTweetCandidateAge)

val clusterIds =

sourceEmbedding

.truncate(config.maxScanClusters).clusterIds

.map { clusterId: ClusterId =>

ClusterKey(clusterId, sourceEmbeddingId.modelVersion, config.candidateEmbeddingType)

}.toSet

Future

.collect {

clusterTweetCandidatesStore.multiGet(clusterIds)

}.map { clusterTweetsMap =>

// Use Mutable map to optimize performance. The method is thread-safe.

// Set initial map size to around p75 of map size distribution to avoid too many copying

// from extending the size of the mutable hashmap

val candidateScoresMap =

new SimClustersANNCandidateSource.HashMap[TweetId, Double](InitialCandidateMapSize)

val candidateNormalizationMap =

new SimClustersANNCandidateSource.HashMap[TweetId, Double](InitialCandidateMapSize)

clusterTweetsMap.foreach {

case (ClusterKey(clusterId, \_, \_, \_), Some(tweetScores))

if sourceEmbedding.contains(clusterId) =>

val sourceClusterScore = sourceEmbedding.getOrElse(clusterId)

for (i <- 0 until Math.min(tweetScores.size, config.maxTopTweetsPerCluster)) {

val (tweetId, score) = tweetScores(i)

if (!parseTweetId(sourceEmbeddingId).contains(tweetId) &&

tweetId >= earliestTweetId && tweetId <= latestTweetId) {

candidateScoresMap.put(

tweetId,

candidateScoresMap.getOrElse(tweetId, 0.0) + score \* sourceClusterScore)

if (config.enablePartialNormalization) {

candidateNormalizationMap

.put(tweetId, candidateNormalizationMap.getOrElse(tweetId, 0.0) + score \* score)

}

}

}

case \_ => ()

}

stats.stat("candidateScoresMap").add(candidateScoresMap.size)

stats.stat("candidateNormalizationMap").add(candidateNormalizationMap.size)

// Re-Rank the candidate by configuration

val processedCandidateScores = candidateScoresMap.map {

case (candidateId, score) =>

// Enable Partial Normalization

val processedScore =

if (config.enablePartialNormalization) {

// We applied the "log" version of partial normalization when we rank candidates

// by log cosine similarity

if (config.rankingAlgorithm == ScoringAlgorithm.PairEmbeddingLogCosineSimilarity) {

score / sourceEmbedding.l2norm / math.log(

1 + candidateNormalizationMap(candidateId))

} else {

score / sourceEmbedding.l2norm / math.sqrt(candidateNormalizationMap(candidateId))

}

} else score

SimClustersTweetCandidate(candidateId, processedScore, sourceEmbeddingId)

}.toSeq

processedCandidateScores

.sortBy(-\_.score)

}

}

private def reranking(

sourceEmbeddingId: SimClustersEmbeddingId,

config: SimClustersANNConfig,

candidates: Seq[SimClustersTweetCandidate]

): Future[Seq[SimClustersTweetCandidate]] = {

val rankedCandidates = if (config.enableHeavyRanking) {

heavyRanker

.rank(

scoringAlgorithm = config.rankingAlgorithm,

sourceEmbeddingId = sourceEmbeddingId,

candidateEmbeddingType = config.candidateEmbeddingType,

minScore = config.minScore,

candidates = candidates.take(config.maxReRankingCandidates)

).map(\_.sortBy(-\_.score))

} else {

Future.value(candidates)

}

rankedCandidates.map(\_.take(config.maxNumResults))

}

private[candidate\_source] def loadConfig(query: Query): Option[SimClustersANNConfig] = {

configs.get(query.sourceEmbeddingId.embeddingType).map { baseConfig =>

// apply overrides if any

query.overrideConfig match {

case Some(overrides) =>

baseConfig.copy(

maxNumResults = overrides.maxNumResults.getOrElse(baseConfig.maxNumResults),

maxTweetCandidateAge =

overrides.maxTweetCandidateAge.getOrElse(baseConfig.maxTweetCandidateAge),

minScore = overrides.minScore.getOrElse(baseConfig.minScore),

candidateEmbeddingType =

overrides.candidateEmbeddingType.getOrElse(baseConfig.candidateEmbeddingType),

enablePartialNormalization =

overrides.enablePartialNormalization.getOrElse(baseConfig.enablePartialNormalization),

enableHeavyRanking =

overrides.enableHeavyRanking.getOrElse(baseConfig.enableHeavyRanking),

rankingAlgorithm = overrides.rankingAlgorithm.getOrElse(baseConfig.rankingAlgorithm),

maxReRankingCandidates =

overrides.maxReRankingCandidates.getOrElse(baseConfig.maxReRankingCandidates),

maxTopTweetsPerCluster =

overrides.maxTopTweetsPerCluster.getOrElse(baseConfig.maxTopTweetsPerCluster),

maxScanClusters = overrides.maxScanClusters.getOrElse(baseConfig.maxScanClusters),

minTweetCandidateAge =

overrides.minTweetCandidateAge.getOrElse(baseConfig.minTweetCandidateAge)

)

case \_ => baseConfig

}

}

}

}

object SimClustersANNCandidateSource {

final val ProductionMaxNumResults = 200

final val InitialCandidateMapSize = 16384

def apply(

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

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding],

uniformScoringStore: ReadableStore[ThriftScoreId, ThriftScore],

configs: Map[EmbeddingType, SimClustersANNConfig],

statsReceiver: StatsReceiver

) = new SimClustersANNCandidateSource(

clusterTweetCandidatesStore = clusterTweetCandidatesStore,

simClustersEmbeddingStore = simClustersEmbeddingStore,

heavyRanker = new UniformScoreStoreRanker(uniformScoringStore, statsReceiver),

configs = configs,

statsReceiver = statsReceiver

)

private def parseTweetId(embeddingId: SimClustersEmbeddingId): Option[TweetId] = {

embeddingId.internalId match {

case InternalId.TweetId(tweetId) =>

Some(tweetId)

case \_ =>

None

}

}

case class Query(

sourceEmbeddingId: SimClustersEmbeddingId,

// Only override the config in DDG and Debuggers.

// Use Post-filter for the holdbacks for better cache hit rate.

overrideConfig: Option[SimClustersANNConfigOverride] = None)

case class SimClustersTweetCandidate(

tweetId: TweetId,

score: Double,

sourceEmbeddingId: SimClustersEmbeddingId)

class HashMap[A, B](initSize: Int) extends mutable.HashMap[A, B] {

override def initialSize: Int = initSize // 16 - by default

}

/\*\*

\* The Configuration of Each SimClusters ANN Candidate Source.

\* Expect One SimClusters Embedding Type mapping to a SimClusters ANN Configuration in Production.

\*/

case class SimClustersANNConfig(

// The max number of candidates for a ANN Query

// Please don't override this value in Production.

maxNumResults: Int = ProductionMaxNumResults,

// The max tweet candidate duration from now.

maxTweetCandidateAge: Duration,

// The min score of the candidates

minScore: Double,

// The Candidate Embedding Type of Tweet.

candidateEmbeddingType: EmbeddingType,

// Enables normalization of approximate SimClusters vectors to remove popularity bias

enablePartialNormalization: Boolean,

// Whether to enable Embedding Similarity ranking

enableHeavyRanking: Boolean,

// The ranking algorithm for Source Candidate Similarity

rankingAlgorithm: ScoringAlgorithm,

// The max number of candidates in ReRanking Step

maxReRankingCandidates: Int,

// The max number of Top Tweets from every cluster tweet index

maxTopTweetsPerCluster: Int,

// The max number of Clusters in the source Embeddings.

maxScanClusters: Int,

// The min tweet candidate duration from now.

minTweetCandidateAge: Duration)

/\*\*

\* Contains same fields as [[SimClustersANNConfig]], to specify which fields are to be overriden

\* for experimental purposes.

\*

\* All fields in this class must be optional.

\*/

case class SimClustersANNConfigOverride(

maxNumResults: Option[Int] = None,

maxTweetCandidateAge: Option[Duration] = None,

minScore: Option[Double] = None,

candidateEmbeddingType: Option[EmbeddingType] = None,

enablePartialNormalization: Option[Boolean] = None,

enableHeavyRanking: Option[Boolean] = None,

rankingAlgorithm: Option[ScoringAlgorithm] = None,

maxReRankingCandidates: Option[Int] = None,

maxTopTweetsPerCluster: Option[Int] = None,

maxScanClusters: Option[Int] = None,

minTweetCandidateAge: Option[Duration] = None,

enableLookbackSource: Option[Boolean] = None)

final val DefaultMaxTopTweetsPerCluster = 200

final val DefaultEnableHeavyRanking = false

object SimClustersANNConfig {

val DefaultSimClustersANNConfig: SimClustersANNConfig =

SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.7,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = false,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = 200,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

)

}

val LookbackMediaMinDays: Int = 0

val LookbackMediaMaxDays: Int = 2

val LookbackMediaMaxTweetsPerDay: Int = 2000

val maxTopTweetsPerCluster: Int =

(LookbackMediaMaxDays - LookbackMediaMinDays + 1) \* LookbackMediaMaxTweetsPerDay

val LookbackMediaTweetConfig: Map[EmbeddingType, SimClustersANNConfig] = {

val candidateEmbeddingType = EmbeddingType.LogFavLongestL2EmbeddingTweet

val minTweetAge = LookbackMediaMinDays.days

val maxTweetAge =

LookbackMediaMaxDays.days - 1.hour // To compensate for the cache TTL that might push the tweet age beyond max age

val rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity

val maxScanClusters = 50

val minScore = 0.5

Map(

EmbeddingType.FavBasedProducer -> SimClustersANNConfig(

minTweetCandidateAge = minTweetAge,

maxTweetCandidateAge = maxTweetAge,

minScore =

minScore, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = candidateEmbeddingType,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = rankingAlgorithm,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = maxTopTweetsPerCluster,

maxScanClusters = maxScanClusters,

),

EmbeddingType.LogFavLongestL2EmbeddingTweet -> SimClustersANNConfig(

minTweetCandidateAge = minTweetAge,

maxTweetCandidateAge = maxTweetAge,

minScore =

minScore, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = candidateEmbeddingType,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = rankingAlgorithm,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = maxTopTweetsPerCluster,

maxScanClusters = maxScanClusters,

),

EmbeddingType.FavTfgTopic -> SimClustersANNConfig(

minTweetCandidateAge = minTweetAge,

maxTweetCandidateAge = maxTweetAge,

minScore = minScore,

candidateEmbeddingType = candidateEmbeddingType,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = rankingAlgorithm,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = 200,

maxScanClusters = maxScanClusters,

),

EmbeddingType.LogFavBasedKgoApeTopic -> SimClustersANNConfig(

minTweetCandidateAge = minTweetAge,

maxTweetCandidateAge = maxTweetAge,

minScore = minScore,

candidateEmbeddingType = candidateEmbeddingType,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = rankingAlgorithm,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = 200,

maxScanClusters = maxScanClusters,

),

)

}

val DefaultConfigMappings: Map[EmbeddingType, SimClustersANNConfig] = Map(

EmbeddingType.FavBasedProducer -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedMaxpoolingAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedAverageAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedBooktypeMaxpoolingAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedLargestDimMaxpoolingAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedLouvainMaxpoolingAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedConnectedMaxpoolingAddressBookFromIIAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.RelaxedAggregatableLogFavBasedProducer -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.25, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 250,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavLongestL2EmbeddingTweet -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.3, // for twistly candidates. To specify a higher threshold, use a post-filter

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.FilteredUserInterestedInFromPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.7, // unused, heavy ranking disabled

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = false,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm =

ScoringAlgorithm.PairEmbeddingCosineSimilarity, // Unused, heavy ranking disabled

maxReRankingCandidates = 150, // unused, heavy ranking disabled

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.FilteredUserInterestedIn -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.7, // unused, heavy ranking disabled

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = false,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm =

ScoringAlgorithm.PairEmbeddingCosineSimilarity, // Unused, heavy ranking disabled

maxReRankingCandidates = 150, // unused, heavy ranking disabled

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.UnfilteredUserInterestedIn -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingLogCosineSimilarity,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.FollowBasedUserInterestedInFromAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 200,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedUserInterestedInFromAPE -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 200,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.FavTfgTopic -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.5,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.LogFavBasedKgoApeTopic -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.5,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 400,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

),

EmbeddingType.UserNextInterestedIn -> SimClustersANNConfig(

maxTweetCandidateAge = 1.days,

minScore = 0.0,

candidateEmbeddingType = EmbeddingType.LogFavBasedTweet,

enablePartialNormalization = true,

enableHeavyRanking = DefaultEnableHeavyRanking,

rankingAlgorithm = ScoringAlgorithm.PairEmbeddingCosineSimilarity,

maxReRankingCandidates = 200,

maxTopTweetsPerCluster = DefaultMaxTopTweetsPerCluster,

maxScanClusters = 50,

minTweetCandidateAge = 0.seconds

)

)

/\*\*

\* Only cache the candidates if it's not Consumer-source. For example, TweetSource, ProducerSource,

\* TopicSource. We don't cache consumer-sources (e.g. UserInterestedIn) since a cached consumer

\* object is going rarely hit, since it can't be shared by multiple users.

\*/

val CacheableShortTTLEmbeddingTypes: Set[EmbeddingType] =

Set(

EmbeddingType.FavBasedProducer,

EmbeddingType.LogFavLongestL2EmbeddingTweet,

)

val CacheableLongTTLEmbeddingTypes: Set[EmbeddingType] =

Set(

EmbeddingType.FavTfgTopic,

EmbeddingType.LogFavBasedKgoApeTopic

)

}