package com.twitter.simclusters\_v2.score

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object SimClustersEmbeddingPairScoreStore {

/\*\*

\* Internal Instance of a SimClusters Embedding based Pair Score store.

\*/

private case class SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding],

score: (SimClustersEmbedding, SimClustersEmbedding) => Future[Option[Double]])

extends PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] {

override val compositeKey1: SimClustersEmbeddingPairScoreId => SimClustersEmbeddingId =

\_.embeddingId1

override val compositeKey2: SimClustersEmbeddingPairScoreId => SimClustersEmbeddingId =

\_.embeddingId2

override def underlyingStore1: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] =

simClustersEmbeddingStore

override def underlyingStore2: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding] =

simClustersEmbeddingStore

override def fromThriftScoreId: ThriftScoreId => SimClustersEmbeddingPairScoreId =

SimClustersEmbeddingPairScoreId.fromThriftScoreId

}

def buildDotProductStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def dotProduct: (SimClustersEmbedding, SimClustersEmbedding) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.dotProduct(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

dotProduct

)

}

def buildCosineSimilarityStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def cosineSimilarity: (SimClustersEmbedding, SimClustersEmbedding) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.cosineSimilarity(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

cosineSimilarity

)

}

def buildLogCosineSimilarityStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def logNormCosineSimilarity: (

SimClustersEmbedding,

SimClustersEmbedding

) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.logNormCosineSimilarity(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

logNormCosineSimilarity

)

}

def buildExpScaledCosineSimilarityStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def expScaledCosineSimilarity: (

SimClustersEmbedding,

SimClustersEmbedding

) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.expScaledCosineSimilarity(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

expScaledCosineSimilarity

)

}

def buildJaccardSimilarityStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def jaccardSimilarity: (

SimClustersEmbedding,

SimClustersEmbedding

) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.jaccardSimilarity(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

jaccardSimilarity

)

}

def buildEuclideanDistanceStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def euclideanDistance: (

SimClustersEmbedding,

SimClustersEmbedding

) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.euclideanDistance(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

euclideanDistance

)

}

def buildManhattanDistanceStore(

simClustersEmbeddingStore: ReadableStore[SimClustersEmbeddingId, SimClustersEmbedding]

): PairScoreStore[

SimClustersEmbeddingPairScoreId,

SimClustersEmbeddingId,

SimClustersEmbeddingId,

SimClustersEmbedding,

SimClustersEmbedding

] = {

def manhattanDistance: (

SimClustersEmbedding,

SimClustersEmbedding

) => Future[Option[Double]] = {

case (embedding1, embedding2) =>

Future.value(Some(embedding1.manhattanDistance(embedding2)))

}

SimClustersEmbeddingInternalPairScoreStore(

simClustersEmbeddingStore,

manhattanDistance

)

}

}