package com.twitter.ann.hnsw

import com.twitter.ann.common.EmbeddingType.EmbeddingVector

import com.twitter.ann.common.{Cosine, Distance, InnerProduct, Metric}

private[hnsw] object DistanceFunctionGenerator {

def apply[T, D <: Distance[D]](

metric: Metric[D],

idToEmbeddingFn: (T) => EmbeddingVector

): DistanceFunctionGenerator[T] = {

// Use InnerProduct for cosine and normalize the vectors before appending and querying.

val updatedMetric = metric match {

case Cosine => InnerProduct

case \_ => metric

}

val distFnIndex = new DistanceFunction[T, T] {

override def distance(id1: T, id2: T) =

updatedMetric.absoluteDistance(

idToEmbeddingFn(id1),

idToEmbeddingFn(id2)

)

}

val distFnQuery = new DistanceFunction[EmbeddingVector, T] {

override def distance(embedding: EmbeddingVector, id: T) =

updatedMetric.absoluteDistance(embedding, idToEmbeddingFn(id))

}

DistanceFunctionGenerator(distFnIndex, distFnQuery, metric == Cosine)

}

}

private[hnsw] case class DistanceFunctionGenerator[T](

index: DistanceFunction[T, T],

query: DistanceFunction[EmbeddingVector, T],

shouldNormalize: Boolean)