package com.twitter.simclusters\_v2.scalding.embedding

import com.twitter.dal.client.dataset.KeyValDALDataset

import com.twitter.recos.entities.thriftscala.Entity

import com.twitter.recos.entities.thriftscala.Hashtag

import com.twitter.recos.entities.thriftscala.SemanticCoreEntity

import com.twitter.scalding.\_

import com.twitter.scalding\_internal.dalv2.DALWrite.\_

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.\_

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil

import com.twitter.simclusters\_v2.scalding.embedding.common.EmbeddingUtil.\_

import com.twitter.simclusters\_v2.scalding.embedding.common.EntityEmbeddingUtil

import com.twitter.simclusters\_v2.scalding.embedding.common.SimClustersEmbeddingJob

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

SimClustersEmbedding => ThriftSimClustersEmbedding,

\_

}

import com.twitter.wtf.entity\_real\_graph.common.EntityUtil

import com.twitter.wtf.entity\_real\_graph.thriftscala.EntityType

import com.twitter.wtf.scalding.jobs.common.AdhocExecutionApp

import com.twitter.wtf.scalding.jobs.common.DataSources

import com.twitter.wtf.scalding.jobs.common.ScheduledExecutionApp

import java.util.TimeZone

/\*\*

\* $ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:entity\_embeddings\_job-adhoc

\*

\* ---------------------- Deploy to atla ----------------------

\* $ scalding remote run \

--main-class com.twitter.simclusters\_v2.scalding.embedding.EntityToSimClustersEmbeddingAdhocApp \

--target src/scala/com/twitter/simclusters\_v2/scalding/embedding:entity\_embeddings\_job-adhoc \

--user recos-platform \

-- --date 2019-09-09 --model-version 20M\_145K\_updated --entity-type SemanticCore

\*/

object EntityToSimClustersEmbeddingAdhocApp extends AdhocExecutionApp {

import EmbeddingUtil.\_

import EntityEmbeddingUtil.\_

import EntityToSimClustersEmbeddingsJob.\_

import EntityUtil.\_

import SimClustersEmbeddingJob.\_

def writeOutput(

embeddings: TypedPipe[(SimClustersEmbeddingId, (ClusterId, EmbeddingScore))],

topKEmbeddings: TypedPipe[(SimClustersEmbeddingId, Seq[(ClusterId, EmbeddingScore)])],

jobConfig: EntityEmbeddingsJobConfig

): Execution[Unit] = {

val toSimClusterEmbeddingExec = topKEmbeddings

.mapValues(SimClustersEmbedding.apply(\_).toThrift)

.writeExecution(

AdhocKeyValSources.entityToClustersSource(

EntityToSimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

isReverseIndex = false,

jobConfig.modelVersion,

jobConfig.entityType)))

val fromSimClusterEmbeddingExec =

toReverseIndexSimClusterEmbedding(embeddings, jobConfig.topK)

.writeExecution(

AdhocKeyValSources.clusterToEntitiesSource(

EntityToSimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

isReverseIndex = true,

jobConfig.modelVersion,

jobConfig.entityType)))

Execution.zip(toSimClusterEmbeddingExec, fromSimClusterEmbeddingExec).unit

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val jobConfig = EntityEmbeddingsJobConfig(args, isAdhoc = true)

val numReducers = args.getOrElse("m", "1000").toInt

/\*

Using the ERG daily dataset in the adhoc job for quick prototyping, note that there may be

issues with scaling the job when productionizing on ERG aggregated dataset.

\*/

val entityRealGraphSource = DataSources.entityRealGraphDailyDataSetSource

val entityUserMatrix: TypedPipe[(Entity, (UserId, Double))] =

(jobConfig.entityType match {

case EntityType.SemanticCore =>

getEntityUserMatrix(entityRealGraphSource, jobConfig.halfLife, EntityType.SemanticCore)

case EntityType.Hashtag =>

getEntityUserMatrix(entityRealGraphSource, jobConfig.halfLife, EntityType.Hashtag)

case \_ =>

throw new IllegalArgumentException(

s"Argument [--entity-type] must be provided. Supported options [${EntityType.SemanticCore.name}, ${EntityType.Hashtag.name}]")

}).forceToDisk

val normalizedUserEntityMatrix =

getNormalizedTransposeInputMatrix(entityUserMatrix, numReducers = Some(numReducers))

//determine which data source to use based on model version

val simClustersSource = jobConfig.modelVersion match {

case ModelVersion.Model20m145kUpdated =>

InterestedInSources.simClustersInterestedInUpdatedSource(dateRange, timeZone)

case \_ =>

InterestedInSources.simClustersInterestedInDec11Source(dateRange, timeZone)

}

val embeddings = computeEmbeddings(

simClustersSource,

normalizedUserEntityMatrix,

scoreExtractors,

ModelVersion.Model20m145kUpdated,

toSimClustersEmbeddingId(jobConfig.modelVersion),

numReducers = Some(numReducers \* 2)

)

val topKEmbeddings =

embeddings.group

.sortedReverseTake(jobConfig.topK)(Ordering.by(\_.\_2))

.withReducers(numReducers)

writeOutput(embeddings, topKEmbeddings, jobConfig)

}

}

/\*\*

\* $ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:semantic\_core\_entity\_embeddings\_2020\_job

\* $ capesospy-v2 update \

--build\_locally \

--start\_cron semantic\_core\_entity\_embeddings\_2020\_job src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object SemanticCoreEntityEmbeddings2020App extends EntityToSimClustersEmbeddingApp

trait EntityToSimClustersEmbeddingApp extends ScheduledExecutionApp {

import EmbeddingUtil.\_

import EntityEmbeddingUtil.\_

import EntityToSimClustersEmbeddingsJob.\_

import EntityUtil.\_

import SimClustersEmbeddingJob.\_

override val firstTime: RichDate = RichDate("2023-01-01")

override val batchIncrement: Duration = Days(7)

private def writeOutput(

embeddings: TypedPipe[(SimClustersEmbeddingId, (ClusterId, EmbeddingScore))],

topKEmbeddings: TypedPipe[(SimClustersEmbeddingId, Seq[(ClusterId, EmbeddingScore)])],

jobConfig: EntityEmbeddingsJobConfig,

clusterEmbeddingsDataset: KeyValDALDataset[

KeyVal[SimClustersEmbeddingId, ThriftSimClustersEmbedding]

],

entityEmbeddingsDataset: KeyValDALDataset[KeyVal[SimClustersEmbeddingId, InternalIdEmbedding]]

): Execution[Unit] = {

val toSimClustersEmbeddings =

topKEmbeddings

.mapValues(SimClustersEmbedding.apply(\_).toThrift)

.map {

case (entityId, topSimClusters) => KeyVal(entityId, topSimClusters)

}

.writeDALVersionedKeyValExecution(

clusterEmbeddingsDataset,

D.Suffix(

EntityToSimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

isReverseIndex = false,

jobConfig.modelVersion,

jobConfig.entityType))

)

val fromSimClustersEmbeddings =

toReverseIndexSimClusterEmbedding(embeddings, jobConfig.topK)

.map {

case (embeddingId, internalIdsWithScore) =>

KeyVal(embeddingId, internalIdsWithScore)

}

.writeDALVersionedKeyValExecution(

entityEmbeddingsDataset,

D.Suffix(

EntityToSimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

isReverseIndex = true,

jobConfig.modelVersion,

jobConfig.entityType))

)

Execution.zip(toSimClustersEmbeddings, fromSimClustersEmbeddings).unit

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val jobConfig = EntityEmbeddingsJobConfig(args, isAdhoc = false)

val embeddingsDataset = EntityEmbeddingsSources.getEntityEmbeddingsDataset(

jobConfig.entityType,

ModelVersions.toKnownForModelVersion(jobConfig.modelVersion)

)

val reverseIndexEmbeddingsDataset =

EntityEmbeddingsSources.getReverseIndexedEntityEmbeddingsDataset(

jobConfig.entityType,

ModelVersions.toKnownForModelVersion(jobConfig.modelVersion)

)

val entityRealGraphSource =

DataSources.entityRealGraphAggregationDataSetSource(dateRange.embiggen(Days(7)))

val entityUserMatrix: TypedPipe[(Entity, (UserId, Double))] =

getEntityUserMatrix(

entityRealGraphSource,

jobConfig.halfLife,

jobConfig.entityType).forceToDisk

val normalizedUserEntityMatrix = getNormalizedTransposeInputMatrix(entityUserMatrix)

val simClustersEmbedding = jobConfig.modelVersion match {

case ModelVersion.Model20m145k2020 =>

val simClustersSource2020 =

InterestedInSources.simClustersInterestedIn2020Source(dateRange, timeZone)

computeEmbeddings(

simClustersSource2020,

normalizedUserEntityMatrix,

scoreExtractors,

ModelVersion.Model20m145k2020,

toSimClustersEmbeddingId(ModelVersion.Model20m145k2020)

)

case modelVersion =>

throw new IllegalArgumentException(s"Model Version ${modelVersion.name} not supported")

}

val topKEmbeddings =

simClustersEmbedding.group.sortedReverseTake(jobConfig.topK)(Ordering.by(\_.\_2))

val simClustersEmbeddingsExec =

writeOutput(

simClustersEmbedding,

topKEmbeddings,

jobConfig,

embeddingsDataset,

reverseIndexEmbeddingsDataset)

// We don't support embeddingsLite for the 2020 model version.

val embeddingsLiteExec = if (jobConfig.modelVersion == ModelVersion.Model20m145kUpdated) {

topKEmbeddings

.collect {

case (

SimClustersEmbeddingId(

EmbeddingType.FavBasedSematicCoreEntity,

ModelVersion.Model20m145kUpdated,

InternalId.EntityId(entityId)),

clustersWithScores) =>

entityId -> clustersWithScores

}

.flatMap {

case (entityId, clustersWithScores) =>

clustersWithScores.map {

case (clusterId, score) => EmbeddingsLite(entityId, clusterId, score)

}

case \_ => Nil

}.writeDALSnapshotExecution(

SimclustersV2EmbeddingsLiteScalaDataset,

D.Daily,

D.Suffix(embeddingsLitePath(ModelVersion.Model20m145kUpdated, "fav\_based")),

D.EBLzo(),

dateRange.end)

} else {

Execution.unit

}

Execution

.zip(simClustersEmbeddingsExec, embeddingsLiteExec).unit

}

}

object EntityToSimClustersEmbeddingsJob {

def toSimClustersEmbeddingId(

modelVersion: ModelVersion

): (Entity, ScoreType.ScoreType) => SimClustersEmbeddingId = {

case (Entity.SemanticCore(SemanticCoreEntity(entityId, \_)), ScoreType.FavScore) =>

SimClustersEmbeddingId(

EmbeddingType.FavBasedSematicCoreEntity,

modelVersion,

InternalId.EntityId(entityId))

case (Entity.SemanticCore(SemanticCoreEntity(entityId, \_)), ScoreType.FollowScore) =>

SimClustersEmbeddingId(

EmbeddingType.FollowBasedSematicCoreEntity,

modelVersion,

InternalId.EntityId(entityId))

case (Entity.Hashtag(Hashtag(hashtag)), ScoreType.FavScore) =>

SimClustersEmbeddingId(

EmbeddingType.FavBasedHashtagEntity,

modelVersion,

InternalId.Hashtag(hashtag))

case (Entity.Hashtag(Hashtag(hashtag)), ScoreType.FollowScore) =>

SimClustersEmbeddingId(

EmbeddingType.FollowBasedHashtagEntity,

modelVersion,

InternalId.Hashtag(hashtag))

case (scoreType, entity) =>

throw new IllegalArgumentException(

s"(ScoreType, Entity) ($scoreType, ${entity.toString}) not supported")

}

/\*\*

\* Generates the output path for the Entity Embeddings Job.

\*

\* Example Adhoc: /user/recos-platform/processed/adhoc/simclusters\_embeddings/hashtag/model\_20m\_145k\_updated

\* Example Prod: /atla/proc/user/cassowary/processed/simclusters\_embeddings/semantic\_core/model\_20m\_145k\_dec11

\*

\*/

def getHdfsPath(

isAdhoc: Boolean,

isManhattanKeyVal: Boolean,

isReverseIndex: Boolean,

modelVersion: ModelVersion,

entityType: EntityType

): String = {

val reverseIndex = if (isReverseIndex) "reverse\_index/" else ""

val entityTypeSuffix = entityType match {

case EntityType.SemanticCore => "semantic\_core"

case EntityType.Hashtag => "hashtag"

case \_ => "unknown"

}

val pathSuffix = s"$reverseIndex$entityTypeSuffix"

EmbeddingUtil.getHdfsPath(isAdhoc, isManhattanKeyVal, modelVersion, pathSuffix)

}

def embeddingsLitePath(modelVersion: ModelVersion, pathSuffix: String): String = {

s"/user/cassowary/processed/entity\_real\_graph/simclusters\_embedding/lite/$modelVersion/$pathSuffix/"

}

}