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.presto\_hdfs\_sources.\_

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

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

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

import com.twitter.simclusters\_v2.scalding.embedding.LocaleEntitySimClustersEmbeddingsJob.\_

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

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

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.Edge

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\_per\_language\_embeddings\_job-adhoc

\*

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

\* $ scalding remote run \

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

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

--user recos-platform \

-- --date 2019-12-17 --model-version 20M\_145K\_updated --entity-type SemanticCore

\*/

object LocaleEntitySimClustersEmbeddingAdhocApp extends AdhocExecutionApp {

// Import implicits

import EntityUtil.\_

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(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

isReverseIndex = false,

isLogFav = false,

jobConfig.modelVersion,

jobConfig.entityType)))

val fromSimClusterEmbeddingExec =

toReverseIndexSimClusterEmbedding(embeddings, jobConfig.topK)

.writeExecution(

AdhocKeyValSources.clusterToEntitiesSource(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

isReverseIndex = true,

isLogFav = false,

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", "2000").toInt

/\*

Can use 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 userEntityMatrix: TypedPipe[(UserId, (Entity, Double))] =

getUserEntityMatrix(

jobConfig,

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

Some(ExternalDataSources.uttEntitiesSource())

).forceToDisk

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

val simClustersSource = jobConfig.modelVersion match {

case ModelVersion.Model20m145kUpdated =>

InterestedInSources.simClustersInterestedInUpdatedSource(dateRange, timeZone)

case modelVersion =>

throw new IllegalArgumentException(

s"SimClusters model version not supported ${modelVersion.name}")

}

val entityPerLanguage = userEntityMatrix.join(ExternalDataSources.userSource).map {

case (userId, ((entity, score), (\_, language))) =>

((entity, language), (userId, score))

}

val normalizedUserEntityMatrix =

getNormalizedTransposeInputMatrix(entityPerLanguage, numReducers = Some(numReducers))

val embeddings = computeEmbeddings[(Entity, String)](

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\_per\_language\_job

\* $ capesospy-v2 update \

--build\_locally \

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

\*/

object LocaleEntitySimClustersEmbeddingScheduledApp extends ScheduledExecutionApp {

// Import implicits

import EmbeddingUtil.\_

import EntityUtil.\_

override val firstTime: RichDate = RichDate("2019-10-22")

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]]

)(

implicit dateRange: DateRange,

timeZone: TimeZone

): Execution[Unit] = {

val thriftSimClustersEmbedding = topKEmbeddings

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

val writeSimClustersEmbeddingKeyValDataset =

thriftSimClustersEmbedding

.map {

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

}

.writeDALVersionedKeyValExecution(

clusterEmbeddingsDataset,

D.Suffix(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

isReverseIndex = false,

isLogFav = false,

jobConfig.modelVersion,

jobConfig.entityType))

)

val writeSimClustersEmbeddingDataset = thriftSimClustersEmbedding

.map {

case (embeddingId, embedding) => SimClustersEmbeddingWithId(embeddingId, embedding)

}

.writeDALSnapshotExecution(

SemanticCorePerLanguageSimclustersEmbeddingsPrestoScalaDataset,

D.Daily,

D.Suffix(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = false,

isReverseIndex = false,

isLogFav = false,

jobConfig.modelVersion,

jobConfig.entityType)),

D.EBLzo(),

dateRange.end

)

val thriftReversedSimclustersEmbeddings =

toReverseIndexSimClusterEmbedding(embeddings, jobConfig.topK)

val writeReverseSimClustersEmbeddingKeyValDataset =

thriftReversedSimclustersEmbeddings

.map {

case (embeddingId, internalIdsWithScore) =>

KeyVal(embeddingId, internalIdsWithScore)

}

.writeDALVersionedKeyValExecution(

entityEmbeddingsDataset,

D.Suffix(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

isReverseIndex = true,

isLogFav = false,

jobConfig.modelVersion,

jobConfig.entityType))

)

val writeReverseSimClustersEmbeddingDataset =

thriftReversedSimclustersEmbeddings

.map {

case (embeddingId, embedding) => InternalIdEmbeddingWithId(embeddingId, embedding)

}.writeDALSnapshotExecution(

ReverseIndexSemanticCorePerLanguageSimclustersEmbeddingsPrestoScalaDataset,

D.Daily,

D.Suffix(

LocaleEntitySimClustersEmbeddingsJob.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = false,

isReverseIndex = true,

isLogFav = false,

jobConfig.modelVersion,

jobConfig.entityType)),

D.EBLzo(),

dateRange.end

)

Execution

.zip(

writeSimClustersEmbeddingDataset,

writeSimClustersEmbeddingKeyValDataset,

writeReverseSimClustersEmbeddingDataset,

writeReverseSimClustersEmbeddingKeyValDataset

).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),

isEmbeddingsPerLocale = true

)

val reverseIndexEmbeddingsDataset =

EntityEmbeddingsSources.getReverseIndexedEntityEmbeddingsDataset(

jobConfig.entityType,

ModelVersions.toKnownForModelVersion(jobConfig.modelVersion),

isEmbeddingsPerLocale = true

)

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

getUserEntityMatrix(

jobConfig,

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

Some(ExternalDataSources.uttEntitiesSource())

).forceToDisk

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

val simClustersSource = jobConfig.modelVersion match {

case ModelVersion.Model20m145kUpdated =>

InterestedInSources.simClustersInterestedInUpdatedSource(dateRange, timeZone)

case modelVersion =>

throw new IllegalArgumentException(

s"SimClusters model version not supported ${modelVersion.name}")

}

val entityPerLanguage = userEntityMatrix.join(ExternalDataSources.userSource).map {

case (userId, ((entity, score), (\_, language))) =>

((entity, language), (userId, score))

}

val normalizedUserEntityMatrix =

getNormalizedTransposeInputMatrix(entityPerLanguage, numReducers = Some(3000))

val simClustersEmbedding = jobConfig.modelVersion match {

case ModelVersion.Model20m145kUpdated =>

computeEmbeddings(

simClustersSource,

normalizedUserEntityMatrix,

scoreExtractors,

ModelVersion.Model20m145kUpdated,

toSimClustersEmbeddingId(ModelVersion.Model20m145kUpdated),

numReducers = Some(8000)

)

case modelVersion =>

throw new IllegalArgumentException(

s"SimClusters model version not supported ${modelVersion.name}")

}

val topKEmbeddings =

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

writeOutput(

simClustersEmbedding,

topKEmbeddings,

jobConfig,

embeddingsDataset,

reverseIndexEmbeddingsDataset)

}

}

object LocaleEntitySimClustersEmbeddingsJob {

def getUserEntityMatrix(

jobConfig: EntityEmbeddingsJobConfig,

entityRealGraphSource: TypedPipe[Edge],

semanticCoreEntityIdsToKeep: Option[TypedPipe[Long]],

applyLogTransform: Boolean = false

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

jobConfig.entityType match {

case EntityType.SemanticCore =>

semanticCoreEntityIdsToKeep match {

case Some(entityIdsToKeep) =>

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

.map {

case (entity, (userId, score)) =>

entity match {

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

if (applyLogTransform) {

(entityId, (userId, (entity, Math.log(score + 1))))

} else {

(entityId, (userId, (entity, score)))

}

case \_ =>

throw new IllegalArgumentException(

"Job config specified EntityType.SemanticCore, but non-semantic core entity was found.")

}

}.hashJoin(entityIdsToKeep.asKeys).values.map {

case ((userId, (entity, score)), \_) => (userId, (entity, score))

}

case \_ =>

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

.map { case (entity, (userId, score)) => (userId, (entity, score)) }

}

case EntityType.Hashtag =>

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

.map { case (entity, (userId, score)) => (userId, (entity, score)) }

case \_ =>

throw new IllegalArgumentException(

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

}

def toSimClustersEmbeddingId(

modelVersion: ModelVersion

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

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

SimClustersEmbeddingId(

EmbeddingType.FavBasedSematicCoreEntity,

modelVersion,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang)))

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

SimClustersEmbeddingId(

EmbeddingType.FollowBasedSematicCoreEntity,

modelVersion,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang)))

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

SimClustersEmbeddingId(

EmbeddingType.LogFavBasedLocaleSemanticCoreEntity,

modelVersion,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang)))

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\_per\_language/model\_20m\_145k\_updated

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

\*

\*/

def getHdfsPath(

isAdhoc: Boolean,

isManhattanKeyVal: Boolean,

isReverseIndex: Boolean,

isLogFav: Boolean,

modelVersion: ModelVersion,

entityType: EntityType

): String = {

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

val logFav = if (isLogFav) "log\_fav/" else ""

val entityTypeSuffix = entityType match {

case EntityType.SemanticCore => "semantic\_core\_per\_language"

case EntityType.Hashtag => "hashtag\_per\_language"

case \_ => "unknown\_per\_language"

}

val pathSuffix = s"$logFav$reverseIndex$entityTypeSuffix"

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

}

}