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

import com.twitter.bijection.{Bufferable, Injection}

import com.twitter.recos.entities.thriftscala.{Entity, SemanticCoreEntity}

import com.twitter.scalding.{DateRange, Days, Duration, Execution, RichDate, TypedPipe, UniqueID}

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.{AdhocKeyValSources, EntityEmbeddingsSources}

import com.twitter.simclusters\_v2.scalding.common.matrix.{SparseMatrix, SparseRowMatrix}

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

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

EmbeddingUtil,

ExternalDataSources,

SimClustersEmbeddingBaseJob

}

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

EmbeddingType,

InternalId,

InternalIdEmbedding,

InternalIdWithScore,

LocaleEntityId,

ModelVersion,

SimClustersEmbeddingId

}

import com.twitter.wtf.entity\_real\_graph.thriftscala.{Edge, FeatureName}

import com.twitter.wtf.scalding.jobs.common.{AdhocExecutionApp, DataSources, ScheduledExecutionApp}

import java.util.TimeZone

/\*\*

\* Scheduled production job which generates topic embeddings per locale based on Entity Real Graph.

\*

\* V2 Uses the log transform of the ERG favScores and the SimCluster InterestedIn scores.

\*

\* $ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:locale\_entity\_simclusters\_embedding\_v2

\* $ capesospy-v2 update \

--build\_locally \

--start\_cron locale\_entity\_simclusters\_embedding\_v2 src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object LocaleEntitySimClustersEmbeddingV2ScheduledApp

extends LocaleEntitySimClustersEmbeddingV2Job

with ScheduledExecutionApp {

override val firstTime: RichDate = RichDate("2020-04-08")

override val batchIncrement: Duration = Days(1)

override def writeNounToClustersIndex(

output: TypedPipe[(LocaleEntity, Seq[(ClusterId, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

output

.map {

case ((entityId, lang), clustersWithScores) =>

KeyVal(

SimClustersEmbeddingId(

EmbeddingType.LogFavBasedLocaleSemanticCoreEntity,

ModelVersion.Model20m145kUpdated,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang))

),

SimClustersEmbedding(clustersWithScores).toThrift

)

}

.writeDALVersionedKeyValExecution(

EntityEmbeddingsSources.LogFavSemanticCorePerLanguageSimClustersEmbeddingsDataset,

D.Suffix(

EmbeddingUtil.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

ModelVersion.Model20m145kUpdated,

pathSuffix = "log\_fav\_erg\_based\_embeddings"))

)

}

override def writeClusterToNounsIndex(

output: TypedPipe[(ClusterId, Seq[(LocaleEntity, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

output

.map {

case (clusterId, nounsWithScore) =>

KeyVal(

SimClustersEmbeddingId(

EmbeddingType.LogFavBasedLocaleSemanticCoreEntity,

ModelVersion.Model20m145kUpdated,

InternalId.ClusterId(clusterId)

),

InternalIdEmbedding(nounsWithScore.map {

case ((entityId, lang), score) =>

InternalIdWithScore(

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

score)

})

)

}

.writeDALVersionedKeyValExecution(

EntityEmbeddingsSources.LogFavReverseIndexSemanticCorePerLanguageSimClustersEmbeddingsDataset,

D.Suffix(

EmbeddingUtil.getHdfsPath(

isAdhoc = false,

isManhattanKeyVal = true,

ModelVersion.Model20m145kUpdated,

pathSuffix = "reverse\_index\_log\_fav\_erg\_based\_embeddings"))

)

}

}

/\*\*

\* $ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:locale\_entity\_simclusters\_embedding\_v2-adhoc

\*

\* $ scalding remote run \

--main-class com.twitter.simclusters\_v2.scalding.embedding.LocaleEntitySimClustersEmbeddingV2AdhocApp \

--target src/scala/com/twitter/simclusters\_v2/scalding/embedding:locale\_entity\_simclusters\_embedding\_v2-adhoc \

--user recos-platform --reducers 2000\

-- --date 2020-04-06

\*/

object LocaleEntitySimClustersEmbeddingV2AdhocApp

extends LocaleEntitySimClustersEmbeddingV2Job

with AdhocExecutionApp {

override def writeNounToClustersIndex(

output: TypedPipe[(LocaleEntity, Seq[(ClusterId, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

output

.map {

case ((entityId, lang), clustersWithScores) =>

SimClustersEmbeddingId(

EmbeddingType.LogFavBasedLocaleSemanticCoreEntity,

ModelVersion.Model20m145kUpdated,

InternalId.LocaleEntityId(LocaleEntityId(entityId, lang))

) -> SimClustersEmbedding(clustersWithScores).toThrift

}.writeExecution(

AdhocKeyValSources.entityToClustersSource(

EmbeddingUtil.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

ModelVersion.Model20m145kUpdated,

pathSuffix = "log\_fav\_erg\_based\_embeddings")))

}

override def writeClusterToNounsIndex(

output: TypedPipe[(ClusterId, Seq[(LocaleEntity, Double)])]

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

output

.map {

case (clusterId, nounsWithScore) =>

SimClustersEmbeddingId(

EmbeddingType.LogFavBasedLocaleSemanticCoreEntity,

ModelVersion.Model20m145kUpdated,

InternalId.ClusterId(clusterId)

) ->

InternalIdEmbedding(nounsWithScore.map {

case ((entityId, lang), score) =>

InternalIdWithScore(

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

score)

})

}

.writeExecution(

AdhocKeyValSources.clusterToEntitiesSource(

EmbeddingUtil.getHdfsPath(

isAdhoc = true,

isManhattanKeyVal = true,

ModelVersion.Model20m145kUpdated,

pathSuffix = "reverse\_index\_log\_fav\_erg\_based\_embeddings")))

}

}

trait LocaleEntitySimClustersEmbeddingV2Job extends SimClustersEmbeddingBaseJob[LocaleEntity] {

override val numClustersPerNoun = 100

override val numNounsPerClusters = 100

override val thresholdForEmbeddingScores: Double = 0.001

override val numReducersOpt: Option[Int] = Some(8000)

private val DefaultERGHalfLifeInDays = 14

private val MinInterestedInLogFavScore = 0.0

implicit val inj: Injection[LocaleEntity, Array[Byte]] = Bufferable.injectionOf[LocaleEntity]

override def prepareNounToUserMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): SparseMatrix[LocaleEntity, UserId, Double] = {

val erg: TypedPipe[(SemanticCoreEntityId, (UserId, Double))] =

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

case Edge(

userId,

Entity.SemanticCore(SemanticCoreEntity(entityId, \_)),

consumerFeatures,

\_,

\_) if consumerFeatures.exists(\_.exists(\_.featureName == FeatureName.Favorites)) =>

for {

features <- consumerFeatures

favFeatures <- features.find(\_.featureName == FeatureName.Favorites)

ewmaMap <- favFeatures.featureValues.ewmaMap

favScore <- ewmaMap.get(DefaultERGHalfLifeInDays)

} yield (entityId, (userId, Math.log(favScore + 1)))

case \_ => None

}

SparseMatrix[LocaleEntity, UserId, Double](

erg

.hashJoin(ExternalDataSources.uttEntitiesSource().asKeys).map {

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

}.join(ExternalDataSources.userSource).map {

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

((entityId, language), userId, score)

}

)

}

override def prepareUserToClusterMatrix(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): SparseRowMatrix[UserId, ClusterId, Double] = {

SparseRowMatrix(

ExternalDataSources.simClustersInterestInLogFavSource(MinInterestedInLogFavScore),

isSkinnyMatrix = true

)

}

}