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

import com.twitter.onboarding.relevance.candidates.thriftscala.InterestBasedUserRecommendations

import com.twitter.onboarding.relevance.candidates.thriftscala.UTTInterest

import com.twitter.onboarding.relevance.source.UttAccountRecommendationsScalaDataset

import com.twitter.scalding.Args

import com.twitter.scalding.DateRange

import com.twitter.scalding.Days

import com.twitter.scalding.Duration

import com.twitter.scalding.Execution

import com.twitter.scalding.RichDate

import com.twitter.scalding.UniqueID

import com.twitter.scalding.typed.TypedPipe

import com.twitter.scalding.typed.UnsortedGrouped

import com.twitter.scalding\_internal.dalv2.DAL

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

import com.twitter.scalding\_internal.dalv2.remote\_access.ExplicitLocation

import com.twitter.scalding\_internal.dalv2.remote\_access.ProcAtla

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.AdhocKeyValSources

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

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

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

import com.twitter.simclusters\_v2.thriftscala

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

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

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

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

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

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

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

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

import com.twitter.wtf.scalding.jobs.common.StatsUtil.\_

import java.util.TimeZone

/\*

$ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:entity\_embedding\_from\_producer\_embedding-adhoc

$ scalding remote run \

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

--target src/scala/com/twitter/simclusters\_v2/scalding/embedding:entity\_embedding\_from\_producer\_embedding-adhoc \

--user recos-platform \

-- --date 2019-10-23 --model\_version 20M\_145K\_updated

\*/

object EntityEmbeddingFromProducerEmbeddingAdhocJob extends AdhocExecutionApp {

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

// step 1: read in (entity, producer) pairs and remove duplicates

val topK = args.getOrElse("top\_k", "100").toInt

val modelVersion = ModelVersions.toModelVersion(

args.getOrElse("model\_version", ModelVersions.Model20M145KUpdated))

val entityKnownForProducers =

EntityEmbeddingFromProducerEmbeddingJob

.getNormalizedEntityProducerMatrix(dateRange.embiggen(Days(7)))

.count("num unique entity producer pairs").map {

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

}

// step 2: read in producer to simclusters embeddings

val producersEmbeddingsFollowBased =

ProducerEmbeddingSources.producerEmbeddingSourceLegacy(

EmbeddingType.ProducerFollowBasedSemanticCoreEntity,

modelVersion)(dateRange.embiggen(Days(7)))

val producersEmbeddingsFavBased =

ProducerEmbeddingSources.producerEmbeddingSourceLegacy(

EmbeddingType.ProducerFavBasedSemanticCoreEntity,

modelVersion)(dateRange.embiggen(Days(7)))

// step 3: join producer embedding with entity, producer pairs and reformat result into format [SimClustersEmbeddingId, SimClustersEmbedding]

val producerBasedEntityEmbeddingsFollowBased =

EntityEmbeddingFromProducerEmbeddingJob

.computeEmbedding(

producersEmbeddingsFollowBased,

entityKnownForProducers,

topK,

modelVersion,

EmbeddingType.ProducerFollowBasedSemanticCoreEntity).toTypedPipe.count(

"follow\_based\_entity\_count")

val producerBasedEntityEmbeddingsFavBased =

EntityEmbeddingFromProducerEmbeddingJob

.computeEmbedding(

producersEmbeddingsFavBased,

entityKnownForProducers,

topK,

modelVersion,

EmbeddingType.ProducerFavBasedSemanticCoreEntity).toTypedPipe.count(

"fav\_based\_entity\_count")

val producerBasedEntityEmbeddings =

producerBasedEntityEmbeddingsFollowBased ++ producerBasedEntityEmbeddingsFavBased

// step 4 write results to file

producerBasedEntityEmbeddings

.count("total\_count").writeExecution(

AdhocKeyValSources.entityToClustersSource(

getHdfsPath(isAdhoc = true, isManhattanKeyVal = true, modelVersion, "producer")))

}

}

/\*

$ ./bazel bundle src/scala/com/twitter/simclusters\_v2/scalding/embedding:entity\_embedding\_from\_producer\_embedding\_job

$ capesospy-v2 update \

--build\_locally \

--start\_cron entity\_embedding\_from\_producer\_embedding\_job src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object EntityEmbeddingFromProducerEmbeddingScheduledJob extends ScheduledExecutionApp {

override def firstTime: RichDate = RichDate("2019-10-16")

override def batchIncrement: Duration = Days(7)

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

// parse args: modelVersion, topK

val topK = args.getOrElse("top\_k", "100").toInt

// only support dec11 now since updated model is not productionized for producer embedding

val modelVersion =

ModelVersions.toModelVersion(

args.getOrElse("model\_version", ModelVersions.Model20M145KUpdated))

val entityKnownForProducers =

EntityEmbeddingFromProducerEmbeddingJob

.getNormalizedEntityProducerMatrix(dateRange.embiggen(Days(7)))

.count("num unique entity producer pairs").map {

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

}

val favBasedEmbeddings = EntityEmbeddingFromProducerEmbeddingJob

.computeEmbedding(

ProducerEmbeddingSources.producerEmbeddingSourceLegacy(

EmbeddingType.ProducerFavBasedSemanticCoreEntity,

modelVersion)(dateRange.embiggen(Days(7))),

entityKnownForProducers,

topK,

modelVersion,

EmbeddingType.ProducerFavBasedSemanticCoreEntity

).toTypedPipe.count("follow\_based\_entity\_count")

val followBasedEmbeddings = EntityEmbeddingFromProducerEmbeddingJob

.computeEmbedding(

ProducerEmbeddingSources.producerEmbeddingSourceLegacy(

EmbeddingType.ProducerFollowBasedSemanticCoreEntity,

modelVersion)(dateRange.embiggen(Days(7))),

entityKnownForProducers,

topK,

modelVersion,

EmbeddingType.ProducerFollowBasedSemanticCoreEntity

).toTypedPipe.count("fav\_based\_entity\_count")

val embedding = favBasedEmbeddings ++ followBasedEmbeddings

embedding

.count("total\_count")

.map {

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

}.writeDALVersionedKeyValExecution(

SemanticCoreEmbeddingsFromProducerScalaDataset,

D.Suffix(getHdfsPath(isAdhoc = false, isManhattanKeyVal = true, modelVersion, "producer"))

)

}

}

private object EntityEmbeddingFromProducerEmbeddingJob {

def computeEmbedding(

producersEmbeddings: TypedPipe[(Long, TopSimClustersWithScore)],

entityKnownForProducers: TypedPipe[(Long, (Long, Double))],

topK: Int,

modelVersion: ModelVersion,

embeddingType: EmbeddingType

): UnsortedGrouped[SimClustersEmbeddingId, thriftscala.SimClustersEmbedding] = {

producersEmbeddings

.hashJoin(entityKnownForProducers).flatMap {

case (\_, (topSimClustersWithScore, (entityId, producerScore))) => {

val entityEmbedding = topSimClustersWithScore.topClusters

entityEmbedding.map {

case SimClusterWithScore(clusterId, score) =>

(

(

SimClustersEmbeddingId(

embeddingType,

modelVersion,

InternalId.EntityId(entityId)),

clusterId),

score \* producerScore)

}

}

}.sumByKey.map {

case ((embeddingId, clusterId), clusterScore) =>

(embeddingId, (clusterId, clusterScore))

}.group.sortedReverseTake(topK)(Ordering.by(\_.\_2)).mapValues(SimClustersEmbedding

.apply(\_).toThrift)

}

def getNormalizedEntityProducerMatrix(

implicit dateRange: DateRange

): TypedPipe[(Long, Long, Double)] = {

val uttRecs: TypedPipe[(UTTInterest, InterestBasedUserRecommendations)] =

DAL

.readMostRecentSnapshot(UttAccountRecommendationsScalaDataset).withRemoteReadPolicy(

ExplicitLocation(ProcAtla)).toTypedPipe.map {

case KeyVal(interest, candidates) => (interest, candidates)

}

uttRecs

.flatMap {

case (interest, candidates) => {

// current populated features

val top20Producers = candidates.recommendations.sortBy(-\_.score.getOrElse(0.0d)).take(20)

val producerScorePairs = top20Producers.map { producer =>

(producer.candidateUserID, producer.score.getOrElse(0.0))

}

val scoreSum = producerScorePairs.map(\_.\_2).sum

producerScorePairs.map {

case (producerId, score) => (interest.uttID, producerId, score / scoreSum)

}

}

}

}

}