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

import com.twitter.dal.client.dataset.SnapshotDALDatasetBase

import com.twitter.ml.api.DataSetPipe

import com.twitter.ml.api.Feature.Continuous

import com.twitter.ml.api.constant.SharedFeatures

import com.twitter.ml.api.util.SRichDataRecord

import com.twitter.scalding.Execution

import com.twitter.scalding.\_

import com.twitter.scalding.typed.UnsortedGrouped

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

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

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

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

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

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

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

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

import com.twitter.simclusters\_v2.hdfs\_sources.FavTfgTopicEmbeddings2020ScalaDataset

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

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

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

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

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

import com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework.conversion.\_

import com.twitter.timelines.prediction.common.aggregates.TimelinesAggregationConfig

import com.twitter.timelines.prediction.features.common.TimelinesSharedFeatures

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

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

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

import java.util.TimeZone

/\*\*

\* Jobs to generate Fav-based engagement weighted Topic-Follow-Graph (TFG) topic embeddings

\* The job uses fav based TFG embeddings and fav based engagement to produce a new embedding

\*/

/\*\*

\* ./bazel bundle ...

\* scalding workflow upload --jobs src/scala/com/twitter/simclusters\_v2/scalding/embedding/tfg:fav\_weighted\_user\_topic\_tfg\_embeddings\_adhoc\_job --autoplay

\*/

object EngagementWeightedTfgBasedTopicEmbeddingsAdhocJob

extends AdhocExecutionApp

with EngagementWeightedTfgBasedTopicEmbeddingsBaseJob {

override val outputByFav =

"/user/cassowary/adhoc/manhattan\_sequence\_files/simclusters\_v2\_embedding/user\_tfgembedding/by\_fav"

override val parquetOutputByFav =

"/user/cassowary/adhoc/processed/simclusters\_v2\_embedding/user\_tfgembedding/by\_fav/snapshot"

}

/\*\*

\* ./bazel bundle ...

\* scalding workflow upload --jobs src/scala/com/twitter/simclusters\_v2/scalding/embedding/tfg:fav\_weighted\_user\_topic\_tfg\_embeddings\_batch\_job --autoplay

\*/

object EngagementWeightedTfgBasedTopicEmbeddingsScheduleJob

extends ScheduledExecutionApp

with EngagementWeightedTfgBasedTopicEmbeddingsBaseJob {

override val firstTime: RichDate = RichDate("2021-10-03")

override val batchIncrement: Duration = Days(1)

override val outputByFav =

"/user/cassowary/manhattan\_sequence\_files/simclusters\_v2\_embedding/user\_tfgembedding/by\_fav"

override val parquetOutputByFav =

"/user/cassowary/processed/simclusters\_v2\_embedding/user\_tfgembedding/by\_fav/snapshot"

}

trait EngagementWeightedTfgBasedTopicEmbeddingsBaseJob extends DateRangeExecutionApp {

val outputByFav: String

val parquetOutputByFav: String

//root path to read aggregate data

private val aggregateFeatureRootPath =

"/atla/proc2/user/timelines/processed/aggregates\_v2"

private val topKTopicsToKeep = 100

private val favContinuousFeature = new Continuous(

"user\_topic\_aggregate.pair.recap.engagement.is\_favorited.any\_feature.50.days.count")

private val parquetDataSource: SnapshotDALDatasetBase[UserTopicWeightedEmbedding] =

UserTopicWeightedEmbeddingParquetScalaDataset

def sortedTake[K](m: Map[K, Double], keysToKeep: Int): Map[K, Double] = {

m.toSeq.sortBy { case (k, v) => -v }.take(keysToKeep).toMap

}

case class UserTopicEngagement(

userId: Long,

topicId: Long,

language: String,

country: String, //field is not used

favCount: Double) {

val userLanguageGroup: (Long, String) = (userId, language)

}

def prepareUserToTopicEmbedding(

favTfgTopicEmbeddings: TypedPipe[(Long, String, SimClustersEmbedding)],

userTopicEngagementCount: TypedPipe[UserTopicEngagement]

)(

implicit uniqueID: UniqueID

): TypedPipe[((Long, String), Map[Int, Double])] = {

val userTfgEmbeddingsStat = Stat("User Tfg Embeddings Count")

val userTopicTopKEngagementStat = Stat("User Topic Top K engagement count")

val userEngagementStat = Stat("User engagement count")

val tfgEmbeddingsStat = Stat("TFG Embedding Map count")

//get only top K topics

val userTopKTopicEngagementCount: TypedPipe[UserTopicEngagement] = userTopicEngagementCount

.groupBy(\_.userLanguageGroup)

.withReducers(499)

.withDescription("select topK topics")

.sortedReverseTake(topKTopicsToKeep)(Ordering.by(\_.favCount))

.values

.flatten

//(userId, language), totalCount

val userLanguageEngagementCount: UnsortedGrouped[(Long, String), Double] =

userTopKTopicEngagementCount

.collect {

case UserTopicEngagement(userId, topicId, language, country, favCount) =>

userTopicTopKEngagementStat.inc()

((userId, language), favCount)

}.sumByKey

.withReducers(499)

.withDescription("fav count by user")

//(topicId, language), (userId, favWeight)

val topicUserWithNormalizedWeights: TypedPipe[((Long, String), (Long, Double))] =

userTopKTopicEngagementCount

.groupBy(\_.userLanguageGroup)

.join(userLanguageEngagementCount)

.withReducers(499)

.withDescription("join userTopic and user EngagementCount")

.collect {

case ((userId, language), (engagementData, totalCount)) =>

userEngagementStat.inc()

(

(engagementData.topicId, engagementData.language),

(userId, engagementData.favCount / totalCount)

)

}

// (topicId, language), embeddingMap

val tfgEmbeddingsMap: TypedPipe[((Long, String), Map[Int, Double])] = favTfgTopicEmbeddings

.map {

case (topicId, language, embedding) =>

tfgEmbeddingsStat.inc()

((topicId, language), embedding.embedding.map(a => a.clusterId -> a.score).toMap)

}

.withDescription("covert sim cluster embedding to map")

// (userId, language), clusters

val newUserTfgEmbedding = topicUserWithNormalizedWeights

.join(tfgEmbeddingsMap)

.withReducers(799)

.withDescription("join user | topic | favWeight \* embedding")

.collect {

case ((topicId, language), ((userId, favWeight), embeddingMap)) =>

userTfgEmbeddingsStat.inc()

((userId, language), embeddingMap.mapValues(\_ \* favWeight))

}

.sumByKey

.withReducers(799)

.withDescription("aggregate embedding by user")

newUserTfgEmbedding.toTypedPipe

}

def writeOutput(

newUserTfgEmbedding: TypedPipe[((Long, String), Map[Int, Double])],

outputPath: String,

parquetOutputPath: String,

modelVersion: String

)(

implicit uniqueID: UniqueID,

dateRange: DateRange

): Execution[Unit] = {

val outputRecordStat = Stat("output record count")

val output = newUserTfgEmbedding

.map {

//language has been purposely ignored because the entire logic is based on the fact that

//user is mapped to a language. In future if a user is mapped to multiple languages then

//the final output needs to be keyed on (userId, language)

case ((userId, language), embeddingMap) =>

outputRecordStat.inc()

val clusterScores = embeddingMap.map {

case (clusterId, score) =>

clusterId -> UserToInterestedInClusterScores(favScore = Some(score))

}

KeyVal(userId, ClustersUserIsInterestedIn(modelVersion, clusterScores))

}

val keyValExec = output

.withDescription("write output keyval dataset")

.writeDALVersionedKeyValExecution(

UserTopicWeightedEmbeddingScalaDataset,

D.Suffix(outputPath))

val parquetExec = newUserTfgEmbedding

.map {

case ((userId, language), embeddingMap) =>

val clusterScores = embeddingMap.map {

case (clusterId, score) => ClustersScore(clusterId, score)

}

UserTopicWeightedEmbedding(userId, clusterScores.toSeq)

}

.withDescription("write output parquet dataset")

.writeDALSnapshotExecution(

parquetDataSource,

D.Daily,

D.Suffix(parquetOutputPath),

D.Parquet,

dateRange.end

)

Execution.zip(keyValExec, parquetExec).unit

}

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val end = dateRange.start

val start = end - Days(21)

val featureDateRange = DateRange(start, end - Millisecs(1))

val outputPath = args.getOrElse("output\_path", outputByFav)

val parquetOutputPath = args.getOrElse("parquet\_output\_path", parquetOutputByFav)

val modelVersion = ModelVersions.Model20M145K2020

//define stats counter

val favTfgTopicEmbeddingsStat = Stat("FavTfgTopicEmbeddings")

val userTopicEngagementStat = Stat("UserTopicEngagement")

val userTopicsStat = Stat("UserTopics")

val userLangStat = Stat("UserLanguage")

//get fav based tfg embeddings

//topic can have different languages and the clusters will be different

//current logic is to filter based on user language

// topicId, lang, embedding

val favTfgTopicEmbeddings: TypedPipe[(Long, String, SimClustersEmbedding)] = DAL

.readMostRecentSnapshot(FavTfgTopicEmbeddings2020ScalaDataset, featureDateRange)

.withRemoteReadPolicy(AllowCrossClusterSameDC)

.toTypedPipe

.collect {

case KeyVal(

SimClustersEmbeddingId(

embedType,

modelVersion,

InternalId.LocaleEntityId(LocaleEntityId(entityId, language))),

embedding) =>

favTfgTopicEmbeddingsStat.inc()

(entityId, language, embedding)

}

/\*

Ideally, if the timeline aggregate framework provided data with breakdown by language,

it could have been joined with (topic, language) embedding. Since, it is not possible

we fetch the language of the user from other sources.

This returns language for the user so that it could be joined with (topic, language) embedding.

`userSource` returns 1 language per user

`inferredUserConsumedLanguageSource` returns multiple languages with confidence values

\*/

val userLangSource = ExternalDataSources.userSource

.map {

case (userId, (country, language)) =>

userLangStat.inc()

(userId, (language, country))

}

//get userid, topicid, favcount as aggregated dataset

//currently there is no way to get language breakdown from the timeline aggregate framework.

val userTopicEngagementPipe: DataSetPipe = AggregatesV2MostRecentFeatureSource(

rootPath = aggregateFeatureRootPath,

storeName = "user\_topic\_aggregates",

aggregates =

Set(TimelinesAggregationConfig.userTopicAggregates).flatMap(\_.buildTypedAggregateGroups()),

).read

val userTopicEngagementCount = userTopicEngagementPipe.records

.flatMap { record =>

val sRichDataRecord = SRichDataRecord(record)

val userId: Long = sRichDataRecord.getFeatureValue(SharedFeatures.USER\_ID)

val topicId: Long = sRichDataRecord.getFeatureValue(TimelinesSharedFeatures.TOPIC\_ID)

val favCount: Double = sRichDataRecord

.getFeatureValueOpt(favContinuousFeature).map(\_.toDouble).getOrElse(0.0)

userTopicEngagementStat.inc()

if (favCount > 0) {

List((userId, (topicId, favCount)))

} else None

}.join(userLangSource)

.collect {

case (userId, ((topicId, favCount), (language, country))) =>

userTopicsStat.inc()

UserTopicEngagement(userId, topicId, language, country, favCount)

}

.withDescription("User Topic aggregated favcount")

// combine user, topics, topic\_embeddings

// and take weighted aggregate of the tfg embedding

val newUserTfgEmbedding =

prepareUserToTopicEmbedding(favTfgTopicEmbeddings, userTopicEngagementCount)

writeOutput(newUserTfgEmbedding, outputPath, parquetOutputPath, modelVersion)

}

}