package com.twitter.simclusters\_v2.scalding.topic\_recommendations.model\_based\_topic\_recommendations

import com.twitter.algebird.Monoid

import com.twitter.bijection.Injection

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

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.\_

import com.twitter.scalding.TypedPipe

import com.twitter.scalding.\_

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

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

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

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

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

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

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

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

import java.util.TimeZone

import scala.util.Random

import com.twitter.ml.api.util.FDsl.\_

import com.twitter.scalding.source.DailySuffixCsv

import com.twitter.scalding.source.DailySuffixTypedTsv

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

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

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

/\*\*

This job is to obtain the training and test data for the model-based approach to topic recommendations:

Approach:

1. Read FavTfgTopicEmbeddingsScalaDataset - to get topic simclusters embeddings for the followed and not interested in topics

2. Read SimclustersV2InterestedIn20M145KUpdatedScalaDataset - to get user's interestedIn Simclusters embeddings

3. Read UsersourceScalaDataset - to get user's countryCode and language

Use the datasets above to get the features for the model and generate DataRecords.

\*/

/\*

To run:

scalding remote run --target src/scala/com/twitter/simclusters\_v2/scalding/topic\_recommendations/model\_based\_topic\_recommendations:training\_data\_for\_topic\_recommendations-adhoc \

--user cassowary \

--submitter atla-aor-08-sr1 \

--main-class com.twitter.simclusters\_v2.scalding.topic\_recommendations.model\_based\_topic\_recommendations.UserTopicFeatureHydrationAdhocApp \

--submitter-memory 128192.megabyte --hadoop-properties "mapreduce.map.memory.mb=8192 mapreduce.map.java.opts='-Xmx7618M' mapreduce.reduce.memory.mb=8192 mapreduce.reduce.java.opts='-Xmx7618M'" \

-- \

--date 2020-10-14 \

--outputDir "/user/cassowary/adhoc/your\_ldap/user\_topic\_features\_popular\_clusters\_filtered\_oct\_16"

\*/

object UserTopicFeatureHydrationAdhocApp extends AdhocExecutionApp {

import UserTopicModellingJobUtils.\_

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val outputDir = args("outputDir")

val numDataRecordsTraining = Stat("num\_data\_records\_training")

val numDataRecordsTesting = Stat("num\_data\_records\_testing")

val testingRatio = args.double("testingRatio", 0.2)

val (trainingDataSamples, testDataSamples, sortedVocab) = UserTopicModellingJobUtils.run(

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.notInterestedTopicsSource,

ExternalDataSources.userSource,

DataSources.getUserInterestedInData,

DataSources.getPerLanguageTopicEmbeddings,

testingRatio

)

val userTopicAdapter = new UserTopicDataRecordAdapter()

Execution

.zip(

convertTypedPipeToDataSetPipe(

trainingDataSamples.map { train =>

numDataRecordsTraining.inc()

train

},

userTopicAdapter)

.writeExecution(

DailySuffixFeatureSink(outputDir + "/training")

),

convertTypedPipeToDataSetPipe(

testDataSamples.map { test =>

numDataRecordsTesting.inc()

test

},

userTopicAdapter)

.writeExecution(

DailySuffixFeatureSink(outputDir + "/testing")

),

sortedVocab

.map { topicsWithSortedIndexes =>

topicsWithSortedIndexes.map(\_.\_1)

}.flatten.writeExecution(DailySuffixTypedTsv(outputDir + "/vocab"))

).unit

}

}

/\*\*

capesospy-v2 update --build\_locally \

--start\_cron training\_data\_for\_topic\_recommendations \

src/scala/com/twitter/simclusters\_v2/capesos\_config/atla\_proc3.yaml

\*/

object UserTopicFeatureHydrationScheduledApp extends ScheduledExecutionApp {

import UserTopicModellingJobUtils.\_

private val outputPath: String =

"/user/cassowary/processed/user\_topic\_modelling"

override def batchIncrement: Duration = Days(1)

override def firstTime: RichDate = RichDate("2020-10-13")

override def runOnDateRange(

args: Args

)(

implicit dateRange: DateRange,

timeZone: TimeZone,

uniqueID: UniqueID

): Execution[Unit] = {

val testingRatio = args.double("testingRatio", 0.2)

val (trainingDataSamples, testDataSamples, sortedVocab) = UserTopicModellingJobUtils.run(

ExternalDataSources.topicFollowGraphSource,

ExternalDataSources.notInterestedTopicsSource,

ExternalDataSources.userSource,

DataSources.getUserInterestedInData,

DataSources.getPerLanguageTopicEmbeddings,

testingRatio

)

val userTopicAdapter = new UserTopicDataRecordAdapter()

Execution

.zip(

getTrainTestExec(

trainingDataSamples,

testDataSamples,

TopicRecommendationsTrainDatarecordsJavaDataset,

TopicRecommendationsTestDatarecordsJavaDataset,

outputPath,

userTopicAdapter

),

sortedVocab

.map { topicsWithSortedIndexes =>

topicsWithSortedIndexes.map(\_.\_1)

}.flatten.writeExecution(DailySuffixTypedTsv(outputPath + "/vocab"))

).unit

}

}

object UserTopicModellingJobUtils {

/\*\*

\* The main function that produces training and the test data

\*

\* @param topicFollowGraphSource user with followed topics from TFG

\* @param notInterestedTopicsSource user with not interested in topics

\* @param userSource user with country and language

\* @param userInterestedInData user with interestedin simcluster embeddings

\* @param topicPerLanguageEmbeddings topics with simcluster embeddings

\*

\* @return Tuple (trainingDataSamples, testingDataSamples, sortedTopicsVocab)

\*/

def run(

topicFollowGraphSource: TypedPipe[(TopicId, UserId)],

notInterestedTopicsSource: TypedPipe[(TopicId, UserId)],

userSource: TypedPipe[(UserId, (Country, Language))],

userInterestedInData: TypedPipe[(UserId, Map[Int, Double])],

topicPerLanguageEmbeddings: TypedPipe[((TopicId, Language), Map[Int, Double])],

testingRatio: Double

)(

implicit uniqueID: UniqueID,

dateRange: DateRange,

timeZone: TimeZone

): (

TypedPipe[UserTopicTrainingSample],

TypedPipe[UserTopicTrainingSample],

TypedPipe[Seq[(TopicId, Int)]]

) = {

val allFollowableTopics: TypedPipe[TopicId] =

topicFollowGraphSource.map(\_.\_1).distinct

val allFollowableTopicsWithMappedIds: TypedPipe[(TopicId, Int)] =

allFollowableTopics.groupAll.mapGroup {

case (\_, topicIter) =>

topicIter.zipWithIndex.map {

case (topicId, mappedId) =>

(topicId, mappedId)

}

}.values

val sortedVocab: TypedPipe[Seq[(TopicId, Int)]] =

allFollowableTopicsWithMappedIds.map(Seq(\_)).map(\_.sortBy(\_.\_2))

val dataTrainingSamples: TypedPipe[UserTopicTrainingSample] = getDataSamplesFromTrainingData(

topicFollowGraphSource,

notInterestedTopicsSource,

userSource,

userInterestedInData,

topicPerLanguageEmbeddings,

allFollowableTopicsWithMappedIds

)

val (trainSplit, testSplit) = splitByUser(dataTrainingSamples, testingRatio)

(trainSplit, testSplit, sortedVocab)

}

/\*\*

\* Split the data samples based on user\_id into train and test data. This ensures that the same

\* user's data records are not part of both train and test data.

\*/

def splitByUser(

dataTrainingSamples: TypedPipe[UserTopicTrainingSample],

testingRatio: Double

): (TypedPipe[UserTopicTrainingSample], TypedPipe[UserTopicTrainingSample]) = {

val (trainSplit, testSplit) = dataTrainingSamples

.map { currSmple => (currSmple.userId, currSmple) }.groupBy(\_.\_1).partition(\_ =>

Random.nextDouble() > testingRatio)

val trainingData = trainSplit.values.map(\_.\_2)

val testingData = testSplit.values.map(\_.\_2)

(trainingData, testingData)

}

/\*\*

\* To get the target topic for each training data sample for a user from the TopicFollowGraph

\*

\* @param topicFollowSource

\* @return (UserId, Set(allFollowedTopicsExceptTargetTopic), targetTopic)

\*/

def getTargetTopicsFromTFG(

topicFollowSource: TypedPipe[(TopicId, UserId)]

)(

implicit uniqueID: UniqueID

): TypedPipe[(UserId, Set[TopicId], TopicId)] = {

val numTrainingSamples = Stat("num\_positive\_training\_samples")

val userFollowedTopics = topicFollowSource.swap

.map {

case (userId, topicId) => (userId, Set(topicId))

}.sumByKey.toTypedPipe

userFollowedTopics.flatMap {

case (userID, followedTopicsSet) =>

followedTopicsSet.map { currFollowedTopic =>

numTrainingSamples.inc()

val remainingTopics = followedTopicsSet - currFollowedTopic

(userID, remainingTopics, currFollowedTopic)

}

}

}

/\*\*

\* Helper function that does the intermediate join operation between a user's followed,

\* not-interested, interestedIn, country and language typedpipe sources, read from different sources.

\*/

def getFeaturesIntermediateJoin(

topicFollowGraphSource: TypedPipe[(TopicId, UserId)],

notInterestedTopicsSource: TypedPipe[(TopicId, UserId)],

allFollowableTopicsWithMappedIds: TypedPipe[(TopicId, Int)],

userCountryAndLanguage: TypedPipe[(UserId, (Country, Language))],

userInterestedInData: TypedPipe[(UserId, Map[Int, Double])]

)(

implicit uniqueID: UniqueID

): TypedPipe[

(

UserId,

Set[TopicId],

Set[TopicId],

TopicId,

Int,

Country,

Language,

Map[Int, Double]

)

] = {

implicit val l2b: Long => Array[Byte] = Injection.long2BigEndian

val userWithFollowedTargetTopics: TypedPipe[

(UserId, Set[TopicId], TopicId)

] = getTargetTopicsFromTFG(topicFollowGraphSource)

val userWithNotInterestedTopics: TypedPipe[(UserId, Set[TopicId])] =

notInterestedTopicsSource.swap.mapValues(Set(\_)).sumByKey.toTypedPipe

userWithFollowedTargetTopics

.groupBy(\_.\_1).leftJoin(userWithNotInterestedTopics).values.map {

case ((userId, followedTopics, targetFollowedTopic), notInterestedOpt) =>

(

userId,

followedTopics,

targetFollowedTopic,

notInterestedOpt.getOrElse(Set.empty[TopicId]))

}

.map {

case (userId, followedTopics, targetFollowedTopic, notInterestedTopics) =>

(targetFollowedTopic, (userId, followedTopics, notInterestedTopics))

}.join(allFollowableTopicsWithMappedIds).map {

case (targetTopic, ((userId, followedTopics, notInterestedTopics), targetTopicIdx)) =>

(userId, followedTopics, notInterestedTopics, targetTopic, targetTopicIdx)

}

.groupBy(\_.\_1).sketch(4000)

.join(userCountryAndLanguage

.groupBy(\_.\_1)).sketch(4000).leftJoin(userInterestedInData)

.values.map {

case (

(

(userId, followedTopics, notInterestedTopics, targetTopic, targetTopicIdx),

(\_, (userCountry, userLanguage))

),

userIntOpt) =>

(

userId,

followedTopics,

notInterestedTopics,

targetTopic,

targetTopicIdx,

userCountry,

userLanguage,

userIntOpt.getOrElse(Map.empty))

}

}

/\*\*

\* Helper function that aggregates user's followed topics, not-interested topics,

\* country, language with join operations and generates the UserTopicTrainingSample

\* for each DataRecord

\*/

def getDataSamplesFromTrainingData(

topicFollowGraphSource: TypedPipe[(TopicId, UserId)],

notInterestedTopicsSource: TypedPipe[(TopicId, UserId)],

userCountryAndLanguage: TypedPipe[(UserId, (Country, Language))],

userInterestedInData: TypedPipe[(UserId, Map[Int, Double])],

topicPerLanguageEmbeddings: TypedPipe[((TopicId, Language), Map[Int, Double])],

allFollowableTopicsWithMappedIds: TypedPipe[(TopicId, Int)]

)(

implicit uniqueID: UniqueID

): TypedPipe[UserTopicTrainingSample] = {

implicit val l2b: Long => Array[Byte] = Injection.long2BigEndian

val allTopicEmbeddingsMap: ValuePipe[Map[(TopicId, Language), Map[Int, Double]]] =

topicPerLanguageEmbeddings.map {

case (topicWithLang, embedding) =>

Map(topicWithLang -> embedding)

}.sum

val userWithFollowedAndNotInterestedTopics = getFeaturesIntermediateJoin(

topicFollowGraphSource,

notInterestedTopicsSource,

allFollowableTopicsWithMappedIds,

userCountryAndLanguage,

userInterestedInData)

userWithFollowedAndNotInterestedTopics.flatMapWithValue(allTopicEmbeddingsMap) {

case (

(

userId,

followedTopics,

notInterestedTopics,

targetTopic,

targetTopicIdx,

userCountry,

userLanguage,

userInt),

Some(allTopicEmbeddings)) =>

val averageFollowedTopicsSimClusters = Monoid

.sum(followedTopics.toSeq.map { topicId =>

allTopicEmbeddings.getOrElse((topicId, userLanguage), Map.empty)

}).mapValues(v =>

v / followedTopics.size) // average simcluster embedding of the followed topics

val averageNotInterestedTopicsSimClusters = Monoid

.sum(notInterestedTopics.toSeq.map { topicId =>

allTopicEmbeddings.getOrElse((topicId, userLanguage), Map.empty)

}).mapValues(v =>

v / notInterestedTopics.size) // average simcluster embedding of the notInterested topics

Some(

UserTopicTrainingSample(

userId,

followedTopics,

notInterestedTopics,

userCountry,

userLanguage,

targetTopicIdx,

userInt,

averageFollowedTopicsSimClusters,

averageNotInterestedTopicsSimClusters

)

)

case \_ =>

None

}

}

/\*\*

\* Write train and test data

\*/

def getTrainTestExec(

trainingData: TypedPipe[UserTopicTrainingSample],

testingData: TypedPipe[UserTopicTrainingSample],

trainDataset: SnapshotDALDatasetBase[DataRecord],

testDataset: SnapshotDALDatasetBase[DataRecord],

outputPath: String,

adapter: IRecordOneToOneAdapter[UserTopicTrainingSample]

)(

implicit dateRange: DateRange

): Execution[Unit] = {

val trainExec =

convertTypedPipeToDataSetPipe(trainingData, adapter)

.writeDALSnapshotExecution(

trainDataset,

D.Daily,

D.Suffix(s"$outputPath/training"),

D.EBLzo(),

dateRange.end)

val testExec =

convertTypedPipeToDataSetPipe(testingData, adapter)

.writeDALSnapshotExecution(

testDataset,

D.Daily,

D.Suffix(s"$outputPath/testing"),

D.EBLzo(),

dateRange.end)

Execution.zip(trainExec, testExec).unit

}

/\*\*

\* To get the datasetPipe containing datarecords hydrated by datarecordAdapter

\* @param userTrainingSamples

\* @param adapter

\* @return DataSetPipe

\*/

def convertTypedPipeToDataSetPipe(

userTrainingSamples: TypedPipe[UserTopicTrainingSample],

adapter: IRecordOneToOneAdapter[UserTopicTrainingSample]

): DataSetPipe = {

userTrainingSamples.toDataSetPipe(adapter)

}

}