package com.twitter.home\_mixer.product.scored\_tweets.side\_effect

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.mysql.Client

import com.twitter.finagle.mysql.Transactions

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.home\_mixer.model.HomeFeatures.ServedRequestIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.SourceTweetIdFeature

import com.twitter.home\_mixer.param.HomeMixerFlagName.DataRecordMetadataStoreConfigsYmlFlag

import com.twitter.home\_mixer.param.HomeMixerFlagName.ScribeServedCommonFeaturesAndCandidateFeaturesFlag

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.CandidateFeaturesScribeEventPublisher

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.CommonFeaturesScribeEventPublisher

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.MinimumFeaturesScribeEventPublisher

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.non\_ml\_features.NonMLCandidateFeatures

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.non\_ml\_features.NonMLCandidateFeaturesAdapter

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.non\_ml\_features.NonMLCommonFeatures

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.non\_ml\_features.NonMLCommonFeaturesAdapter

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsQuery

import com.twitter.home\_mixer.product.scored\_tweets.model.ScoredTweetsResponse

import com.twitter.home\_mixer.product.scored\_tweets.scorer.CandidateFeaturesDataRecordFeature

import com.twitter.home\_mixer.product.scored\_tweets.scorer.CommonFeaturesDataRecordFeature

import com.twitter.home\_mixer.product.scored\_tweets.scorer.PredictedScoreFeature.PredictedScoreFeatures

import com.twitter.home\_mixer.util.CandidatesUtil.getOriginalAuthorId

import com.twitter.inject.annotations.Flag

import com.twitter.logpipeline.client.common.EventPublisher

import com.twitter.ml.api.DataRecordMerger

import com.twitter.product\_mixer.core.feature.featuremap.datarecord.DataRecordConverter

import com.twitter.product\_mixer.core.feature.featuremap.datarecord.SpecificFeatures

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

import com.twitter.product\_mixer.core.model.common.identifier.SideEffectIdentifier

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.stitch.Stitch

import com.twitter.timelines.ml.cont\_train.common.domain.non\_scalding.CandidateAndCommonFeaturesStreamingUtils

import com.twitter.timelines.ml.pldr.client.MysqlClientUtils

import com.twitter.timelines.ml.pldr.client.VersionedMetadataCacheClient

import com.twitter.timelines.ml.pldr.conversion.VersionIdAndFeatures

import com.twitter.timelines.suggests.common.data\_record\_metadata.{thriftscala => drmd}

import com.twitter.timelines.suggests.common.poly\_data\_record.{thriftjava => pldr}

import com.twitter.timelines.util.stats.OptionObserver

import com.twitter.util.Time

import com.twitter.util.Try

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

/\*\*

\* (1) Scribe common features sent to prediction service + some other features as PLDR format into logs

\* (2) Scribe candidate features sent to prediction service + some other features as PLDR format into another logs

\*/

@Singleton

class ScribeServedCommonFeaturesAndCandidateFeaturesSideEffect @Inject() (

@Flag(DataRecordMetadataStoreConfigsYmlFlag) dataRecordMetadataStoreConfigsYml: String,

@Flag(ScribeServedCommonFeaturesAndCandidateFeaturesFlag) enableScribeServedCommonFeaturesAndCandidateFeatures: Boolean,

@Named(CommonFeaturesScribeEventPublisher) commonFeaturesScribeEventPublisher: EventPublisher[

pldr.PolyDataRecord

],

@Named(CandidateFeaturesScribeEventPublisher) candidateFeaturesScribeEventPublisher: EventPublisher[

pldr.PolyDataRecord

],

@Named(MinimumFeaturesScribeEventPublisher) minimumFeaturesScribeEventPublisher: EventPublisher[

pldr.PolyDataRecord

],

statsReceiver: StatsReceiver,

) extends PipelineResultSideEffect[ScoredTweetsQuery, ScoredTweetsResponse]

with PipelineResultSideEffect.Conditionally[ScoredTweetsQuery, ScoredTweetsResponse]

with Logging {

override val identifier: SideEffectIdentifier =

SideEffectIdentifier("ScribeServedCommonFeaturesAndCandidateFeatures")

private val drMerger = new DataRecordMerger

private val postScoringCandidateFeatures = SpecificFeatures(PredictedScoreFeatures)

private val postScoringCandidateFeaturesDataRecordAdapter =

new DataRecordConverter(postScoringCandidateFeatures)

private val scopedStatsReceiver = statsReceiver.scope(getClass.getSimpleName)

private val metadataFetchFailedCounter = scopedStatsReceiver.counter("metadataFetchFailed")

private val commonFeaturesScribeCounter = scopedStatsReceiver.counter("commonFeaturesScribe")

private val commonFeaturesPLDROptionObserver =

OptionObserver(scopedStatsReceiver.scope("commonFeaturesPLDR"))

private val candidateFeaturesScribeCounter =

scopedStatsReceiver.counter("candidateFeaturesScribe")

private val candidateFeaturesPLDROptionObserver =

OptionObserver(scopedStatsReceiver.scope("candidateFeaturesPLDR"))

private val minimumFeaturesPLDROptionObserver =

OptionObserver(scopedStatsReceiver.scope("minimumFeaturesPLDR"))

private val minimumFeaturesScribeCounter =

scopedStatsReceiver.counter("minimumFeaturesScribe")

lazy private val dataRecordMetadataStoreClient: Option[Client with Transactions] =

Try {

MysqlClientUtils.mysqlClientProvider(

MysqlClientUtils.parseConfigFromYaml(dataRecordMetadataStoreConfigsYml))

}.onFailure { e => info(s"Error building MySQL client: $e") }.toOption

lazy private val versionedMetadataCacheClientOpt: Option[

VersionedMetadataCacheClient[Map[drmd.FeaturesCategory, Option[VersionIdAndFeatures]]]

] =

dataRecordMetadataStoreClient.map { mysqlClient =>

new VersionedMetadataCacheClient[Map[drmd.FeaturesCategory, Option[VersionIdAndFeatures]]](

maximumSize = 1,

expireDurationOpt = None,

mysqlClient = mysqlClient,

transform = CandidateAndCommonFeaturesStreamingUtils.metadataTransformer,

statsReceiver = statsReceiver

)

}

versionedMetadataCacheClientOpt.foreach { versionedMetadataCacheClient =>

versionedMetadataCacheClient

.metadataFetchTimerTask(

CandidateAndCommonFeaturesStreamingUtils.metadataFetchKey,

metadataFetchTimer = DefaultTimer,

metadataFetchInterval = 90.seconds,

metadataFetchFailedCounter = metadataFetchFailedCounter

)

}

override def onlyIf(

query: ScoredTweetsQuery,

selectedCandidates: Seq[CandidateWithDetails],

remainingCandidates: Seq[CandidateWithDetails],

droppedCandidates: Seq[CandidateWithDetails],

response: ScoredTweetsResponse

): Boolean = enableScribeServedCommonFeaturesAndCandidateFeatures

override def apply(

inputs: PipelineResultSideEffect.Inputs[ScoredTweetsQuery, ScoredTweetsResponse]

): Stitch[Unit] = {

Stitch.value {

val servedTimestamp: Long = Time.now.inMilliseconds

val nonMLCommonFeatures = NonMLCommonFeatures(

userId = inputs.query.getRequiredUserId,

predictionRequestId =

inputs.query.features.flatMap(\_.getOrElse(ServedRequestIdFeature, None)),

servedTimestamp = servedTimestamp

)

val nonMLCommonFeaturesDataRecord =

NonMLCommonFeaturesAdapter.adaptToDataRecords(nonMLCommonFeatures).asScala.head

/\*\*

\* Steps of scribing common features

\* (1) fetch common features as data record

\* (2) extract additional feature as data record, e.g. predictionRequestId which is used as join key in downstream jobs

\* (3) merge two data records above and convert the merged data record to pldr

\* (4) publish pldr

\*/

val commonFeaturesDataRecordOpt =

inputs.selectedCandidates.headOption.map(\_.features.get(CommonFeaturesDataRecordFeature))

val commonFeaturesPLDROpt = commonFeaturesDataRecordOpt.flatMap { commonFeaturesDataRecord =>

drMerger.merge(commonFeaturesDataRecord, nonMLCommonFeaturesDataRecord)

CandidateAndCommonFeaturesStreamingUtils.commonFeaturesToPolyDataRecord(

versionedMetadataCacheClientOpt = versionedMetadataCacheClientOpt,

commonFeatures = commonFeaturesDataRecord,

valueFormat = pldr.PolyDataRecord.\_Fields.LITE\_COMPACT\_DATA\_RECORD

)

}

commonFeaturesPLDROptionObserver(commonFeaturesPLDROpt).foreach { pldr =>

commonFeaturesScribeEventPublisher.publish(pldr)

commonFeaturesScribeCounter.incr()

}

/\*\*

\* steps of scribing candidate features

\* (1) fetch candidate features as data record

\* (2) extract additional features (mostly non ML features including predicted scores, predictionRequestId, userId, tweetId)

\* (3) merge data records and convert the merged data record into pldr

\* (4) publish pldr

\*/

inputs.selectedCandidates.foreach { candidate =>

val candidateFeaturesDataRecord = candidate.features.get(CandidateFeaturesDataRecordFeature)

/\*\*

\* extract predicted scores as data record and merge it into original data record

\*/

val postScoringCandidateFeaturesDataRecord =

postScoringCandidateFeaturesDataRecordAdapter.toDataRecord(candidate.features)

drMerger.merge(candidateFeaturesDataRecord, postScoringCandidateFeaturesDataRecord)

/\*\*

\* extract non ML common features as data record and merge it into original data record

\*/

drMerger.merge(candidateFeaturesDataRecord, nonMLCommonFeaturesDataRecord)

/\*\*

\* extract non ML candidate features as data record and merge it into original data record

\*/

val nonMLCandidateFeatures = NonMLCandidateFeatures(

tweetId = candidate.candidateIdLong,

sourceTweetId = candidate.features.getOrElse(SourceTweetIdFeature, None),

originalAuthorId = getOriginalAuthorId(candidate.features)

)

val nonMLCandidateFeaturesDataRecord =

NonMLCandidateFeaturesAdapter.adaptToDataRecords(nonMLCandidateFeatures).asScala.head

drMerger.merge(candidateFeaturesDataRecord, nonMLCandidateFeaturesDataRecord)

val candidateFeaturesPLDROpt =

CandidateAndCommonFeaturesStreamingUtils.candidateFeaturesToPolyDataRecord(

versionedMetadataCacheClientOpt = versionedMetadataCacheClientOpt,

candidateFeatures = candidateFeaturesDataRecord,

valueFormat = pldr.PolyDataRecord.\_Fields.LITE\_COMPACT\_DATA\_RECORD

)

candidateFeaturesPLDROptionObserver(candidateFeaturesPLDROpt).foreach { pldr =>

candidateFeaturesScribeEventPublisher.publish(pldr)

candidateFeaturesScribeCounter.incr()

}

// scribe minimum features which are used to join labels from client events.

val minimumFeaturesPLDROpt = candidateFeaturesPLDROpt

.map(CandidateAndCommonFeaturesStreamingUtils.extractMinimumFeaturesFromPldr)

.map(pldr.PolyDataRecord.dataRecord)

minimumFeaturesPLDROptionObserver(minimumFeaturesPLDROpt).foreach { pldr =>

minimumFeaturesScribeEventPublisher.publish(pldr)

minimumFeaturesScribeCounter.incr()

}

}

}

}

}