package com.twitter.timelines.data\_processing.ad\_hoc.earlybird\_ranking.common

import com.twitter.ml.api.DataRecord

import com.twitter.ml.api.Feature

import com.twitter.ml.api.FeatureContext

import com.twitter.ml.api.ITransform

import com.twitter.ml.api.transform.CascadeTransform

import com.twitter.ml.api.transform.TransformFactory

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

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

import com.twitter.search.common.features.SearchResultFeature

import com.twitter.search.common.features.ExternalTweetFeature

import com.twitter.search.common.features.TweetFeature

import com.twitter.timelines.prediction.features.recap.RecapFeatures

import com.twitter.timelines.prediction.features.request\_context.RequestContextFeatures

import com.twitter.timelines.prediction.features.time\_features.TimeDataRecordFeatures

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

import com.twitter.timelines.prediction.features.real\_graph.RealGraphDataRecordFeatures

import scala.collection.JavaConverters.\_

import java.lang.{Boolean => JBoolean}

case class LabelInfo(name: String, downsampleFraction: Double, importance: Double)

case class LabelInfoWithFeature(info: LabelInfo, feature: Feature[JBoolean])

trait EarlybirdTrainingConfiguration {

protected def labels: Map[String, Feature.Binary]

protected def weights: Map[String, Double] = Map(

"detail\_expanded" -> 0.3,

"favorited" -> 1.0,

"open\_linked" -> 0.1,

"photo\_expanded" -> 0.03,

"profile\_clicked" -> 1.0,

"replied" -> 9.0,

"retweeted" -> 1.0,

"video\_playback50" -> 0.01

)

// we basically should not downsample any of the precious positive data.

// importance are currently set to match the full model's weights.

protected def PositiveSamplingRate: Double = 1.0

private def NegativeSamplingRate: Double = PositiveSamplingRate \* 0.08

// we basically should not downsample any of the precious positive data.

// importance are currently set to match the full model's weights.

final lazy val LabelInfos: List[LabelInfoWithFeature] = {

assert(labels.keySet == weights.keySet)

labels.keySet.map(makeLabelInfoWithFeature).toList

}

def makeLabelInfoWithFeature(labelName: String): LabelInfoWithFeature = {

LabelInfoWithFeature(

LabelInfo(labelName, PositiveSamplingRate, weights(labelName)),

labels(labelName))

}

final lazy val NegativeInfo: LabelInfo = LabelInfo("negative", NegativeSamplingRate, 1.0)

// example of features available in schema based namespace:

protected def featureToSearchResultFeatureMap: Map[Feature[\_], SearchResultFeature] = Map(

RecapFeatures.TEXT\_SCORE -> TweetFeature.TEXT\_SCORE,

RecapFeatures.REPLY\_COUNT -> TweetFeature.REPLY\_COUNT,

RecapFeatures.RETWEET\_COUNT -> TweetFeature.RETWEET\_COUNT,

RecapFeatures.FAV\_COUNT -> TweetFeature.FAVORITE\_COUNT,

RecapFeatures.HAS\_CARD -> TweetFeature.HAS\_CARD\_FLAG,

RecapFeatures.HAS\_CONSUMER\_VIDEO -> TweetFeature.HAS\_CONSUMER\_VIDEO\_FLAG,

RecapFeatures.HAS\_PRO\_VIDEO -> TweetFeature.HAS\_PRO\_VIDEO\_FLAG,

// no corresponding HAS\_NATIVE\_VIDEO feature in TweetFeature

RecapFeatures.HAS\_VINE -> TweetFeature.HAS\_VINE\_FLAG,

RecapFeatures.HAS\_PERISCOPE -> TweetFeature.HAS\_PERISCOPE\_FLAG,

RecapFeatures.HAS\_NATIVE\_IMAGE -> TweetFeature.HAS\_NATIVE\_IMAGE\_FLAG,

RecapFeatures.HAS\_IMAGE -> TweetFeature.HAS\_IMAGE\_URL\_FLAG,

RecapFeatures.HAS\_NEWS -> TweetFeature.HAS\_NEWS\_URL\_FLAG,

RecapFeatures.HAS\_VIDEO -> TweetFeature.HAS\_VIDEO\_URL\_FLAG,

RecapFeatures.HAS\_TREND -> TweetFeature.HAS\_TREND\_FLAG,

RecapFeatures.HAS\_MULTIPLE\_HASHTAGS\_OR\_TRENDS -> TweetFeature.HAS\_MULTIPLE\_HASHTAGS\_OR\_TRENDS\_FLAG,

RecapFeatures.IS\_OFFENSIVE -> TweetFeature.IS\_OFFENSIVE\_FLAG,

RecapFeatures.IS\_REPLY -> TweetFeature.IS\_REPLY\_FLAG,

RecapFeatures.IS\_RETWEET -> TweetFeature.IS\_RETWEET\_FLAG,

RecapFeatures.IS\_AUTHOR\_BOT -> TweetFeature.IS\_USER\_BOT\_FLAG,

RecapFeatures.FROM\_VERIFIED\_ACCOUNT -> TweetFeature.FROM\_VERIFIED\_ACCOUNT\_FLAG,

RecapFeatures.USER\_REP -> TweetFeature.USER\_REPUTATION,

RecapFeatures.EMBEDS\_IMPRESSION\_COUNT -> TweetFeature.EMBEDS\_IMPRESSION\_COUNT,

RecapFeatures.EMBEDS\_URL\_COUNT -> TweetFeature.EMBEDS\_URL\_COUNT,

// RecapFeatures.VIDEO\_VIEW\_COUNT deprecated

RecapFeatures.FAV\_COUNT\_V2 -> TweetFeature.FAVORITE\_COUNT\_V2,

RecapFeatures.RETWEET\_COUNT\_V2 -> TweetFeature.RETWEET\_COUNT\_V2,

RecapFeatures.REPLY\_COUNT\_V2 -> TweetFeature.REPLY\_COUNT\_V2,

RecapFeatures.IS\_SENSITIVE -> TweetFeature.IS\_SENSITIVE\_CONTENT,

RecapFeatures.HAS\_MULTIPLE\_MEDIA -> TweetFeature.HAS\_MULTIPLE\_MEDIA\_FLAG,

RecapFeatures.IS\_AUTHOR\_PROFILE\_EGG -> TweetFeature.PROFILE\_IS\_EGG\_FLAG,

RecapFeatures.IS\_AUTHOR\_NEW -> TweetFeature.IS\_USER\_NEW\_FLAG,

RecapFeatures.NUM\_MENTIONS -> TweetFeature.NUM\_MENTIONS,

RecapFeatures.NUM\_HASHTAGS -> TweetFeature.NUM\_HASHTAGS,

RecapFeatures.HAS\_VISIBLE\_LINK -> TweetFeature.HAS\_VISIBLE\_LINK\_FLAG,

RecapFeatures.HAS\_LINK -> TweetFeature.HAS\_LINK\_FLAG,

//note: DISCRETE features are not supported by the modelInterpreter tool.

// for the following features, we will create separate CONTINUOUS features instead of renaming

//RecapFeatures.LINK\_LANGUAGE

//RecapFeatures.LANGUAGE

TimelinesSharedFeatures.HAS\_QUOTE -> TweetFeature.HAS\_QUOTE\_FLAG,

TimelinesSharedFeatures.QUOTE\_COUNT -> TweetFeature.QUOTE\_COUNT,

TimelinesSharedFeatures.WEIGHTED\_FAV\_COUNT -> TweetFeature.WEIGHTED\_FAVORITE\_COUNT,

TimelinesSharedFeatures.WEIGHTED\_QUOTE\_COUNT -> TweetFeature.WEIGHTED\_QUOTE\_COUNT,

TimelinesSharedFeatures.WEIGHTED\_REPLY\_COUNT -> TweetFeature.WEIGHTED\_REPLY\_COUNT,

TimelinesSharedFeatures.WEIGHTED\_RETWEET\_COUNT -> TweetFeature.WEIGHTED\_RETWEET\_COUNT,

TimelinesSharedFeatures.DECAYED\_FAVORITE\_COUNT -> TweetFeature.DECAYED\_FAVORITE\_COUNT,

TimelinesSharedFeatures.DECAYED\_RETWEET\_COUNT -> TweetFeature.DECAYED\_RETWEET\_COUNT,

TimelinesSharedFeatures.DECAYED\_REPLY\_COUNT -> TweetFeature.DECAYED\_RETWEET\_COUNT,

TimelinesSharedFeatures.DECAYED\_QUOTE\_COUNT -> TweetFeature.DECAYED\_QUOTE\_COUNT,

TimelinesSharedFeatures.FAKE\_FAVORITE\_COUNT -> TweetFeature.FAKE\_FAVORITE\_COUNT,

TimelinesSharedFeatures.FAKE\_RETWEET\_COUNT -> TweetFeature.FAKE\_RETWEET\_COUNT,

TimelinesSharedFeatures.FAKE\_REPLY\_COUNT -> TweetFeature.FAKE\_REPLY\_COUNT,

TimelinesSharedFeatures.FAKE\_QUOTE\_COUNT -> TweetFeature.FAKE\_QUOTE\_COUNT,

TimelinesSharedFeatures.EMBEDS\_IMPRESSION\_COUNT\_V2 -> TweetFeature.EMBEDS\_IMPRESSION\_COUNT\_V2,

TimelinesSharedFeatures.EMBEDS\_URL\_COUNT\_V2 -> TweetFeature.EMBEDS\_URL\_COUNT\_V2,

TimelinesSharedFeatures.LABEL\_ABUSIVE\_FLAG -> TweetFeature.LABEL\_ABUSIVE\_FLAG,

TimelinesSharedFeatures.LABEL\_ABUSIVE\_HI\_RCL\_FLAG -> TweetFeature.LABEL\_ABUSIVE\_HI\_RCL\_FLAG,

TimelinesSharedFeatures.LABEL\_DUP\_CONTENT\_FLAG -> TweetFeature.LABEL\_DUP\_CONTENT\_FLAG,

TimelinesSharedFeatures.LABEL\_NSFW\_HI\_PRC\_FLAG -> TweetFeature.LABEL\_NSFW\_HI\_PRC\_FLAG,

TimelinesSharedFeatures.LABEL\_NSFW\_HI\_RCL\_FLAG -> TweetFeature.LABEL\_NSFW\_HI\_RCL\_FLAG,

TimelinesSharedFeatures.LABEL\_SPAM\_FLAG -> TweetFeature.LABEL\_SPAM\_FLAG,

TimelinesSharedFeatures.LABEL\_SPAM\_HI\_RCL\_FLAG -> TweetFeature.LABEL\_SPAM\_HI\_RCL\_FLAG

)

protected def derivedFeaturesAdder: ITransform =

new ITransform {

private val hasEnglishTweetDiffUiLangFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.HAS\_ENGLISH\_TWEET\_DIFF\_UI\_LANG)

.asInstanceOf[Feature.Binary]

private val hasEnglishUiDiffTweetLangFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.HAS\_ENGLISH\_UI\_DIFF\_TWEET\_LANG)

.asInstanceOf[Feature.Binary]

private val hasDiffLangFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.HAS\_DIFF\_LANG)

.asInstanceOf[Feature.Binary]

private val isSelfTweetFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.IS\_SELF\_TWEET)

.asInstanceOf[Feature.Binary]

private val tweetAgeInSecsFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.TWEET\_AGE\_IN\_SECS)

.asInstanceOf[Feature.Continuous]

private val authorSpecificScoreFeature =

featureInstanceFromSearchResultFeature(ExternalTweetFeature.AUTHOR\_SPECIFIC\_SCORE)

.asInstanceOf[Feature.Continuous]

// see comments above

private val linkLanguageFeature = new Feature.Continuous(TweetFeature.LINK\_LANGUAGE.getName)

private val languageFeature = new Feature.Continuous(TweetFeature.LANGUAGE.getName)

override def transformContext(featureContext: FeatureContext): FeatureContext =

featureContext.addFeatures(

authorSpecificScoreFeature,

// used when training against the full scoreEarlybirdModelEvaluationJob.scala

// TimelinesSharedFeatures.PREDICTED\_SCORE\_LOG,

hasEnglishTweetDiffUiLangFeature,

hasEnglishUiDiffTweetLangFeature,

hasDiffLangFeature,

isSelfTweetFeature,

tweetAgeInSecsFeature,

linkLanguageFeature,

languageFeature

)

override def transform(record: DataRecord): Unit = {

val srecord = SRichDataRecord(record)

srecord.getFeatureValueOpt(RealGraphDataRecordFeatures.WEIGHT).map { realgraphWeight =>

srecord.setFeatureValue(authorSpecificScoreFeature, realgraphWeight)

}

// use this when training against the log of the full score

// srecord.getFeatureValueOpt(TimelinesSharedFeatures.PREDICTED\_SCORE).map { score =>

// if (score > 0.0) {

// srecord.setFeatureValue(TimelinesSharedFeatures.PREDICTED\_SCORE\_LOG, Math.log(score))

// }

// }

if (srecord.hasFeature(RequestContextFeatures.LANGUAGE\_CODE) && srecord.hasFeature(

RecapFeatures.LANGUAGE)) {

val uilangIsEnglish = srecord

.getFeatureValue(RequestContextFeatures.LANGUAGE\_CODE).toString == "en"

val tweetIsEnglish = srecord.getFeatureValue(RecapFeatures.LANGUAGE) == 5

srecord.setFeatureValue(

hasEnglishTweetDiffUiLangFeature,

tweetIsEnglish && !uilangIsEnglish

)

srecord.setFeatureValue(

hasEnglishUiDiffTweetLangFeature,

uilangIsEnglish && !tweetIsEnglish

)

}

srecord.getFeatureValueOpt(RecapFeatures.MATCH\_UI\_LANG).map { match\_ui\_lang =>

srecord.setFeatureValue(

hasDiffLangFeature,

!match\_ui\_lang

)

}

for {

author\_id <- srecord.getFeatureValueOpt(SharedFeatures.AUTHOR\_ID)

user\_id <- srecord.getFeatureValueOpt(SharedFeatures.USER\_ID)

} srecord.setFeatureValue(

isSelfTweetFeature,

author\_id == user\_id

)

srecord.getFeatureValueOpt(TimeDataRecordFeatures.TIME\_SINCE\_TWEET\_CREATION).map {

time\_since\_tweet\_creation =>

srecord.setFeatureValue(

tweetAgeInSecsFeature,

time\_since\_tweet\_creation / 1000.0

)

}

srecord.getFeatureValueOpt(RecapFeatures.LINK\_LANGUAGE).map { link\_language =>

srecord.setFeatureValue(

linkLanguageFeature,

link\_language.toDouble

)

}

srecord.getFeatureValueOpt(RecapFeatures.LANGUAGE).map { language =>

srecord.setFeatureValue(

languageFeature,

language.toDouble

)

}

}

}

protected def featureInstanceFromSearchResultFeature(

tweetFeature: SearchResultFeature

): Feature[\_] = {

val featureType = tweetFeature.getType

val featureName = tweetFeature.getName

require(

!tweetFeature.isDiscrete && (

featureType == com.twitter.search.common.features.thrift.ThriftSearchFeatureType.BOOLEAN\_VALUE ||

featureType == com.twitter.search.common.features.thrift.ThriftSearchFeatureType.DOUBLE\_VALUE ||

featureType == com.twitter.search.common.features.thrift.ThriftSearchFeatureType.INT32\_VALUE

)

)

if (featureType == com.twitter.search.common.features.thrift.ThriftSearchFeatureType.BOOLEAN\_VALUE)

new Feature.Binary(featureName)

else

new Feature.Continuous(featureName)

}

lazy val EarlybirdFeatureRenamer: ITransform = {

val earlybirdFeatureRenameMap: Map[Feature[\_], Feature[\_]] =

featureToSearchResultFeatureMap.map {

case (originalFeature, tweetFeature) =>

originalFeature -> featureInstanceFromSearchResultFeature(tweetFeature)

}.toMap

new CascadeTransform(

List(

derivedFeaturesAdder,

TransformFactory.produceTransform(

TransformFactory.produceFeatureRenameTransformSpec(

earlybirdFeatureRenameMap.asJava

)

)

).asJava

)

}

}