package com.twitter.home\_mixer.product.for\_you

import com.twitter.tweetconvosvc.tweet\_ancestor.{thriftscala => ta}

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

import com.twitter.mediaservices.commons.tweetmedia.{thriftscala => mt}

import com.twitter.product\_mixer.component\_library.candidate\_source.timeline\_scorer.ScoredTweetCandidateWithFocalTweet

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidateFeatureTransformer

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

import com.twitter.product\_mixer.core.model.marshalling.response.urt.metadata.BasicTopicContextFunctionalityType

import com.twitter.product\_mixer.core.model.marshalling.response.urt.metadata.RecWithEducationTopicContextFunctionalityType

import com.twitter.product\_mixer.core.model.marshalling.response.urt.metadata.RecommendationTopicContextFunctionalityType

import com.twitter.search.common.constants.thriftjava.ThriftLanguage

import com.twitter.search.common.util.lang.ThriftLanguageUtil

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.timelinemixer.injection.model.candidate.AudioSpaceMetaData

import com.twitter.timelines.conversation\_features.{thriftscala => cvt}

import com.twitter.timelinescorer.common.scoredtweetcandidate.{thriftscala => stc}

import com.twitter.timelineservice.suggests.{thriftscala => tls}

object ForYouTimelineScorerResponseFeatureTransformer

extends CandidateFeatureTransformer[ScoredTweetCandidateWithFocalTweet] {

override val identifier: TransformerIdentifier =

TransformerIdentifier("ForYouTimelineScorerResponse")

override val features: Set[Feature[\_, \_]] = Set(

AncestorsFeature,

AudioSpaceMetaDataFeature,

AuthorIdFeature,

AuthorIsBlueVerifiedFeature,

AuthorIsCreatorFeature,

AuthorIsGoldVerifiedFeature,

AuthorIsGrayVerifiedFeature,

AuthorIsLegacyVerifiedFeature,

AuthoredByContextualUserFeature,

CandidateSourceIdFeature,

ConversationFeature,

ConversationModuleFocalTweetIdFeature,

ConversationModuleIdFeature,

DirectedAtUserIdFeature,

EarlybirdFeature,

EntityTokenFeature,

ExclusiveConversationAuthorIdFeature,

FavoritedByUserIdsFeature,

FollowedByUserIdsFeature,

TopicIdSocialContextFeature,

TopicContextFunctionalityTypeFeature,

FromInNetworkSourceFeature,

FullScoringSucceededFeature,

HasDisplayedTextFeature,

InNetworkFeature,

InReplyToTweetIdFeature,

IsAncestorCandidateFeature,

IsExtendedReplyFeature,

IsRandomTweetFeature,

IsReadFromCacheFeature,

IsRetweetFeature,

IsRetweetedReplyFeature,

NonSelfFavoritedByUserIdsFeature,

NumImagesFeature,

OriginalTweetCreationTimeFromSnowflakeFeature,

PredictionRequestIdFeature,

QuotedTweetIdFeature,

ScoreFeature,

SimclustersTweetTopKClustersWithScoresFeature,

SourceTweetIdFeature,

SourceUserIdFeature,

StreamToKafkaFeature,

SuggestTypeFeature,

TweetLanguageFeature,

VideoDurationMsFeature,

)

// Convert language code to ISO 639-3 format

private def getLanguageISOFormatByValue(languageCodeValue: Int): String =

ThriftLanguageUtil.getLanguageCodeOf(ThriftLanguage.findByValue(languageCodeValue))

override def transform(

candidateWithFocalTweet: ScoredTweetCandidateWithFocalTweet

): FeatureMap = {

val candidate: stc.v1.ScoredTweetCandidate = candidateWithFocalTweet.candidate

val focalTweetId = candidateWithFocalTweet.focalTweetIdOpt

val originalTweetId = candidate.sourceTweetId.getOrElse(candidate.tweetId)

val tweetFeatures = candidate.tweetFeaturesMap.flatMap(\_.get(originalTweetId))

val earlybirdFeatures = tweetFeatures.flatMap(\_.recapFeatures.flatMap(\_.tweetFeatures))

val directedAtUserIsInFirstDegree =

earlybirdFeatures.flatMap(\_.directedAtUserIdIsInFirstDegree)

val isReply = candidate.inReplyToTweetId.nonEmpty

val isRetweet = candidate.isRetweet.getOrElse(false)

val isInNetwork = candidate.isInNetwork.getOrElse(true)

val conversationFeatures = candidate.conversationFeatures.flatMap {

case cvt.ConversationFeatures.V1(candidate) => Some(candidate)

case \_ => None

}

val numImages = candidate.mediaMetaData

.map(

\_.count(mediaEntity =>

mediaEntity.mediaInfo.exists(\_.isInstanceOf[mt.MediaInfo.ImageInfo]) ||

mediaEntity.mediaInfo.isEmpty))

val hasImage = earlybirdFeatures.exists(\_.hasImage)

val hasVideo = earlybirdFeatures.exists(\_.hasVideo)

val hasCard = earlybirdFeatures.exists(\_.hasCard)

val hasQuote = earlybirdFeatures.exists(\_.hasQuote.contains(true))

val hasDisplayedText = earlybirdFeatures.exists(\_.tweetLength.exists(length => {

val numMedia = Seq(hasVideo, (hasImage || hasCard), hasQuote).count(b => b)

val tcoLengthsPlusSpaces = 23 \* numMedia + (if (numMedia > 0) numMedia - 1 else 0)

length > tcoLengthsPlusSpaces

}))

val suggestType = candidate.overrideSuggestType.orElse(Some(tls.SuggestType.Undefined))

val topicSocialProofMetadataOpt = candidate.entityData.flatMap(\_.topicSocialProofMetadata)

val topicIdSocialContextOpt = topicSocialProofMetadataOpt.map(\_.topicId)

val topicContextFunctionalityTypeOpt =

topicSocialProofMetadataOpt.map(\_.topicContextFunctionalityType).collect {

case stc.v1.TopicContextFunctionalityType.Basic => BasicTopicContextFunctionalityType

case stc.v1.TopicContextFunctionalityType.Recommendation =>

RecommendationTopicContextFunctionalityType

case stc.v1.TopicContextFunctionalityType.RecWithEducation =>

RecWithEducationTopicContextFunctionalityType

}

FeatureMapBuilder()

.add(

AncestorsFeature,

candidate.ancestors

.getOrElse(Seq.empty)

.map(ancestor => ta.TweetAncestor(ancestor.tweetId, ancestor.userId.getOrElse(0L))))

.add(

AudioSpaceMetaDataFeature,

candidate.audioSpaceMetaDatalist.map(\_.head).map(AudioSpaceMetaData.fromThrift))

.add(AuthorIdFeature, Some(candidate.authorId))

.add(AuthorIsBlueVerifiedFeature, candidate.authorIsBlueVerified.getOrElse(false))

.add(

AuthorIsCreatorFeature,

candidate.authorIsCreator.getOrElse(false)

)

.add(AuthorIsGoldVerifiedFeature, candidate.authorIsGoldVerified.getOrElse(false))

.add(AuthorIsGrayVerifiedFeature, candidate.authorIsGrayVerified.getOrElse(false))

.add(AuthorIsLegacyVerifiedFeature, candidate.authorIsLegacyVerified.getOrElse(false))

.add(

AuthoredByContextualUserFeature,

candidate.viewerId.contains(candidate.authorId) ||

candidate.viewerId.exists(candidate.sourceUserId.contains))

.add(CandidateSourceIdFeature, candidate.candidateTweetSourceId)

.add(ConversationFeature, conversationFeatures)

.add(ConversationModuleIdFeature, candidate.conversationId)

.add(ConversationModuleFocalTweetIdFeature, focalTweetId)

.add(DirectedAtUserIdFeature, candidate.directedAtUserId)

.add(EarlybirdFeature, earlybirdFeatures)

// This is temporary, will need to be updated with the encoded string.

.add(EntityTokenFeature, Some("test\_EntityTokenForYou"))

.add(ExclusiveConversationAuthorIdFeature, candidate.exclusiveConversationAuthorId)

.add(FavoritedByUserIdsFeature, candidate.favoritedByUserIds.getOrElse(Seq.empty))

.add(FollowedByUserIdsFeature, candidate.followedByUserIds.getOrElse(Seq.empty))

.add(TopicIdSocialContextFeature, topicIdSocialContextOpt)

.add(TopicContextFunctionalityTypeFeature, topicContextFunctionalityTypeOpt)

.add(FullScoringSucceededFeature, candidate.fullScoringSucceeded.getOrElse(false))

.add(HasDisplayedTextFeature, hasDisplayedText)

.add(InNetworkFeature, candidate.isInNetwork.getOrElse(true))

.add(InReplyToTweetIdFeature, candidate.inReplyToTweetId)

.add(IsAncestorCandidateFeature, candidate.isAncestorCandidate.getOrElse(false))

.add(

IsExtendedReplyFeature,

isInNetwork && isReply && !isRetweet && directedAtUserIsInFirstDegree.contains(false))

.add(FromInNetworkSourceFeature, candidate.isInNetwork.getOrElse(true))

.add(IsRandomTweetFeature, candidate.isRandomTweet.getOrElse(false))

.add(IsReadFromCacheFeature, candidate.isReadFromCache.getOrElse(false))

.add(IsRetweetFeature, candidate.isRetweet.getOrElse(false))

.add(IsRetweetedReplyFeature, isReply && isRetweet)

.add(

NonSelfFavoritedByUserIdsFeature,

candidate.favoritedByUserIds.getOrElse(Seq.empty).filterNot(\_ == candidate.authorId))

.add(NumImagesFeature, numImages)

.add(

OriginalTweetCreationTimeFromSnowflakeFeature,

SnowflakeId.timeFromIdOpt(originalTweetId))

.add(PredictionRequestIdFeature, candidate.predictionRequestId)

.add(ScoreFeature, Some(candidate.score))

.add(

SimclustersTweetTopKClustersWithScoresFeature,

candidate.simclustersTweetTopKClustersWithScores.map(\_.toMap).getOrElse(Map.empty))

.add(

StreamToKafkaFeature,

candidate.predictionRequestId.nonEmpty && candidate.fullScoringSucceeded.getOrElse(false))

.add(SourceTweetIdFeature, candidate.sourceTweetId)

.add(SourceUserIdFeature, candidate.sourceUserId)

.add(SuggestTypeFeature, suggestType)

.add(QuotedTweetIdFeature, candidate.quotedTweetId)

.add(

TweetLanguageFeature,

earlybirdFeatures.flatMap(\_.language.map(\_.value)).map(getLanguageISOFormatByValue))

.add(VideoDurationMsFeature, earlybirdFeatures.flatMap(\_.videoDurationMs))

.build()

}

}