package com.twitter.search.common.converter.earlybird;

import java.io.IOException;

import java.util.HashSet;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Optional;

import java.util.Set;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import java.util.stream.Collectors;

import com.google.common.base.Joiner;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.apache.commons.lang.StringUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.text.token.TokenizedCharSequence;

import com.twitter.common.text.token.TokenizedCharSequenceStream;

import com.twitter.common.text.util.TokenStreamSerializer;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.search.common.indexing.thriftjava.Place;

import com.twitter.search.common.indexing.thriftjava.PotentialLocation;

import com.twitter.search.common.indexing.thriftjava.ProfileGeoEnrichment;

import com.twitter.search.common.indexing.thriftjava.ThriftExpandedUrl;

import com.twitter.search.common.indexing.thriftjava.VersionedTweetFeatures;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.relevance.entities.PotentialLocationObject;

import com.twitter.search.common.relevance.entities.TwitterMessage;

import com.twitter.search.common.relevance.features.FeatureSink;

import com.twitter.search.common.relevance.features.MutableFeatureNormalizers;

import com.twitter.search.common.relevance.features.RelevanceSignalConstants;

import com.twitter.search.common.relevance.features.TweetTextFeatures;

import com.twitter.search.common.relevance.features.TweetTextQuality;

import com.twitter.search.common.relevance.features.TweetUserFeatures;

import com.twitter.search.common.schema.base.FeatureConfiguration;

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

import com.twitter.search.common.schema.earlybird.EarlybirdEncodedFeatures;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

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

import com.twitter.search.common.util.text.LanguageIdentifierHelper;

import com.twitter.search.common.util.text.NormalizerHelper;

import com.twitter.search.common.util.text.SourceNormalizer;

import com.twitter.search.common.util.text.TokenizerHelper;

import com.twitter.search.common.util.text.TokenizerResult;

import com.twitter.search.common.util.text.TweetTokenStreamSerializer;

import com.twitter.search.common.util.url.LinkVisibilityUtils;

import com.twitter.search.common.util.url.NativeVideoClassificationUtils;

import com.twitter.search.ingester.model.VisibleTokenRatioUtil;

/\*\*

\* EncodedFeatureBuilder helps to build encoded features for TwitterMessage.

\*

\* This is stateful so should only be used one tweet at a time

\*/

public class EncodedFeatureBuilder {

private static final Logger LOG = LoggerFactory.getLogger(EncodedFeatureBuilder.class);

private static final SearchCounter NUM\_TWEETS\_WITH\_INVALID\_TWEET\_ID\_IN\_PHOTO\_URL =

SearchCounter.export("tweets\_with\_invalid\_tweet\_id\_in\_photo\_url");

// TwitterTokenStream for converting TokenizedCharSequence into a stream for serialization

// This is stateful so should only be used one tweet at a time

private final TokenizedCharSequenceStream tokenSeqStream = new TokenizedCharSequenceStream();

// SUPPRESS CHECKSTYLE:OFF LineLength

private static final Pattern TWITTER\_PHOTO\_PERMA\_LINK\_PATTERN =

Pattern.compile("(?i:^(?:(?:https?\\:\\/\\/)?(?:www\\.)?)?twitter\\.com\\/(?:\\?[^#]+)?(?:#!?\\/?)?\\w{1,20}\\/status\\/(\\d+)\\/photo\\/\\d\*$)");

private static final Pattern TWITTER\_PHOTO\_COPY\_PASTE\_LINK\_PATTERN =

Pattern.compile("(?i:^(?:(?:https?\\:\\/\\/)?(?:www\\.)?)?twitter\\.com\\/(?:#!?\\/)?\\w{1,20}\\/status\\/(\\d+)\\/photo\\/\\d\*$)");

// SUPPRESS CHECKSTYLE:ON LineLength

private static final VisibleTokenRatioUtil VISIBLE\_TOKEN\_RATIO = new VisibleTokenRatioUtil();

private static final Map<PenguinVersion, SearchCounter> SERIALIZE\_FAILURE\_COUNTERS\_MAP =

Maps.newEnumMap(PenguinVersion.class);

static {

for (PenguinVersion penguinVersion : PenguinVersion.values()) {

SERIALIZE\_FAILURE\_COUNTERS\_MAP.put(

penguinVersion,

SearchCounter.export(

"tokenstream\_serialization\_failure\_" + penguinVersion.name().toLowerCase()));

}

}

public static class TweetFeatureWithEncodeFeatures {

public final VersionedTweetFeatures versionedFeatures;

public final EarlybirdEncodedFeatures encodedFeatures;

public final EarlybirdEncodedFeatures extendedEncodedFeatures;

public TweetFeatureWithEncodeFeatures(

VersionedTweetFeatures versionedFeatures,

EarlybirdEncodedFeatures encodedFeatures,

EarlybirdEncodedFeatures extendedEncodedFeatures) {

this.versionedFeatures = versionedFeatures;

this.encodedFeatures = encodedFeatures;

this.extendedEncodedFeatures = extendedEncodedFeatures;

}

}

/\*\*

\* Create tweet text features and the encoded features.

\*

\* @param message the tweet message

\* @param penguinVersion the based penguin version to create the features

\* @param schemaSnapshot the schema associated with the features

\* @return the text features and the encoded features

\*/

public TweetFeatureWithEncodeFeatures createTweetFeaturesFromTwitterMessage(

TwitterMessage message,

PenguinVersion penguinVersion,

ImmutableSchemaInterface schemaSnapshot) {

VersionedTweetFeatures versionedTweetFeatures = new VersionedTweetFeatures();

// Write extendedPackedFeatures.

EarlybirdEncodedFeatures extendedEncodedFeatures =

createExtendedEncodedFeaturesFromTwitterMessage(message, penguinVersion, schemaSnapshot);

if (extendedEncodedFeatures != null) {

extendedEncodedFeatures

.writeExtendedFeaturesToVersionedTweetFeatures(versionedTweetFeatures);

}

setSourceAndNormalizedSource(

message.getStrippedSource(), versionedTweetFeatures, penguinVersion);

TweetTextFeatures textFeatures = message.getTweetTextFeatures(penguinVersion);

///////////////////////////////

// Add hashtags and mentions

textFeatures.getHashtags().forEach(versionedTweetFeatures::addToHashtags);

textFeatures.getMentions().forEach(versionedTweetFeatures::addToMentions);

///////////////////////////////

// Extract some extra information from the message text.

// Index stock symbols with $ prepended

textFeatures.getStocks().stream()

.filter(stock -> stock != null)

.forEach(stock -> versionedTweetFeatures.addToStocks(stock.toLowerCase()));

// Question marks

versionedTweetFeatures.setHasQuestionMark(textFeatures.hasQuestionMark());

// Smileys

versionedTweetFeatures.setHasPositiveSmiley(textFeatures.hasPositiveSmiley());

versionedTweetFeatures.setHasNegativeSmiley(textFeatures.hasNegativeSmiley());

TokenStreamSerializer streamSerializer =

TweetTokenStreamSerializer.getTweetTokenStreamSerializer();

TokenizedCharSequence tokenSeq = textFeatures.getTokenSequence();

tokenSeqStream.reset(tokenSeq);

int tokenPercent = VISIBLE\_TOKEN\_RATIO.extractAndNormalizeTokenPercentage(tokenSeqStream);

tokenSeqStream.reset(tokenSeq);

// Write packedFeatures.

EarlybirdEncodedFeatures encodedFeatures = createEncodedFeaturesFromTwitterMessage(

message, penguinVersion, schemaSnapshot, tokenPercent);

encodedFeatures.writeFeaturesToVersionedTweetFeatures(versionedTweetFeatures);

try {

versionedTweetFeatures.setTweetTokenStream(streamSerializer.serialize(tokenSeqStream));

versionedTweetFeatures.setTweetTokenStreamText(tokenSeq.toString());

} catch (IOException e) {

LOG.error("TwitterTokenStream serialization error! Could not serialize: "

+ tokenSeq.toString());

SERIALIZE\_FAILURE\_COUNTERS\_MAP.get(penguinVersion).increment();

versionedTweetFeatures.unsetTweetTokenStream();

versionedTweetFeatures.unsetTweetTokenStreamText();

}

// User name features

if (message.getFromUserDisplayName().isPresent()) {

Locale locale = LanguageIdentifierHelper

.identifyLanguage(message.getFromUserDisplayName().get());

String normalizedDisplayName = NormalizerHelper.normalize(

message.getFromUserDisplayName().get(), locale, penguinVersion);

TokenizerResult result = TokenizerHelper

.tokenizeTweet(normalizedDisplayName, locale, penguinVersion);

tokenSeqStream.reset(result.tokenSequence);

try {

versionedTweetFeatures.setUserDisplayNameTokenStream(

streamSerializer.serialize(tokenSeqStream));

versionedTweetFeatures.setUserDisplayNameTokenStreamText(result.tokenSequence.toString());

} catch (IOException e) {

LOG.error("TwitterTokenStream serialization error! Could not serialize: "

+ message.getFromUserDisplayName().get());

SERIALIZE\_FAILURE\_COUNTERS\_MAP.get(penguinVersion).increment();

versionedTweetFeatures.unsetUserDisplayNameTokenStream();

versionedTweetFeatures.unsetUserDisplayNameTokenStreamText();

}

}

String resolvedUrlsText = Joiner.on(" ").skipNulls().join(textFeatures.getResolvedUrlTokens());

versionedTweetFeatures.setNormalizedResolvedUrlText(resolvedUrlsText);

addPlace(message, versionedTweetFeatures, penguinVersion);

addProfileGeoEnrichment(message, versionedTweetFeatures, penguinVersion);

versionedTweetFeatures.setTweetSignature(message.getTweetSignature(penguinVersion));

return new TweetFeatureWithEncodeFeatures(

versionedTweetFeatures, encodedFeatures, extendedEncodedFeatures);

}

protected static void setSourceAndNormalizedSource(

String strippedSource,

VersionedTweetFeatures versionedTweetFeatures,

PenguinVersion penguinVersion) {

if (strippedSource != null && !strippedSource.isEmpty()) {

// normalize source for searchable field - replaces whitespace with underscores (???).

versionedTweetFeatures.setNormalizedSource(

SourceNormalizer.normalize(strippedSource, penguinVersion));

// source facet has simpler normalization.

Locale locale = LanguageIdentifierHelper.identifyLanguage(strippedSource);

versionedTweetFeatures.setSource(NormalizerHelper.normalizeKeepCase(

strippedSource, locale, penguinVersion));

}

}

/\*\*

\* Adds the given photo url to the thrift status if it is a twitter photo permalink.

\* Returns true, if this was indeed a twitter photo, false otherwise.

\*/

public static boolean addPhotoUrl(TwitterMessage message, String photoPermalink) {

Matcher matcher = TWITTER\_PHOTO\_COPY\_PASTE\_LINK\_PATTERN.matcher(photoPermalink);

if (!matcher.matches() || matcher.groupCount() < 1) {

matcher = TWITTER\_PHOTO\_PERMA\_LINK\_PATTERN.matcher(photoPermalink);

}

if (matcher.matches() && matcher.groupCount() == 1) {

// this is a native photo url which we need to store in a separate field

String idStr = matcher.group(1);

if (idStr != null) {

// idStr should be a valid tweet ID (and therefore, should fit into a Long), but we have

// tweets for which idStr is a long sequence of digits that does not fit into a Long.

try {

long photoStatusId = Long.parseLong(idStr);

message.addPhotoUrl(photoStatusId, null);

} catch (NumberFormatException e) {

LOG.warn("Found a tweet with a photo URL with an invalid tweet ID: " + message);

NUM\_TWEETS\_WITH\_INVALID\_TWEET\_ID\_IN\_PHOTO\_URL.increment();

}

}

return true;

}

return false;

}

private void addPlace(TwitterMessage message,

VersionedTweetFeatures versionedTweetFeatures,

PenguinVersion penguinVersion) {

String placeId = message.getPlaceId();

if (placeId == null) {

return;

}

// Tweet.Place.id and Tweet.Place.full\_name are both required fields.

String placeFullName = message.getPlaceFullName();

Preconditions.checkNotNull(placeFullName, "Tweet.Place without full\_name.");

Locale placeFullNameLocale = LanguageIdentifierHelper.identifyLanguage(placeFullName);

String normalizedPlaceFullName =

NormalizerHelper.normalize(placeFullName, placeFullNameLocale, penguinVersion);

String tokenizedPlaceFullName = StringUtils.join(

TokenizerHelper.tokenizeQuery(normalizedPlaceFullName, placeFullNameLocale, penguinVersion),

" ");

Place place = new Place(placeId, tokenizedPlaceFullName);

String placeCountryCode = message.getPlaceCountryCode();

if (placeCountryCode != null) {

Locale placeCountryCodeLocale = LanguageIdentifierHelper.identifyLanguage(placeCountryCode);

place.setCountryCode(

NormalizerHelper.normalize(placeCountryCode, placeCountryCodeLocale, penguinVersion));

}

versionedTweetFeatures.setTokenizedPlace(place);

}

private void addProfileGeoEnrichment(TwitterMessage message,

VersionedTweetFeatures versionedTweetFeatures,

PenguinVersion penguinVersion) {

List<PotentialLocationObject> potentialLocations = message.getPotentialLocations();

if (potentialLocations.isEmpty()) {

return;

}

List<PotentialLocation> thriftPotentialLocations = Lists.newArrayList();

for (PotentialLocationObject potentialLocation : potentialLocations) {

thriftPotentialLocations.add(potentialLocation.toThriftPotentialLocation(penguinVersion));

}

versionedTweetFeatures.setTokenizedProfileGeoEnrichment(

new ProfileGeoEnrichment(thriftPotentialLocations));

}

/\*\* Returns the encoded features. \*/

public static EarlybirdEncodedFeatures createEncodedFeaturesFromTwitterMessage(

TwitterMessage message,

PenguinVersion penguinVersion,

ImmutableSchemaInterface schema,

int normalizedTokenPercentBucket) {

FeatureSink sink = new FeatureSink(schema);

// Static features

sink.setBooleanValue(EarlybirdFieldConstant.IS\_RETWEET\_FLAG, message.isRetweet())

.setBooleanValue(EarlybirdFieldConstant.IS\_REPLY\_FLAG, message.isReply())

.setBooleanValue(

EarlybirdFieldConstant.FROM\_VERIFIED\_ACCOUNT\_FLAG, message.isUserVerified())

.setBooleanValue(

EarlybirdFieldConstant.FROM\_BLUE\_VERIFIED\_ACCOUNT\_FLAG, message.isUserBlueVerified())

.setBooleanValue(EarlybirdFieldConstant.IS\_SENSITIVE\_CONTENT, message.isSensitiveContent());

TweetTextFeatures textFeatures = message.getTweetTextFeatures(penguinVersion);

if (textFeatures != null) {

final FeatureConfiguration featureConfigNumHashtags = schema.getFeatureConfigurationByName(

EarlybirdFieldConstant.NUM\_HASHTAGS.getFieldName());

final FeatureConfiguration featureConfigNumMentions = schema.getFeatureConfigurationByName(

EarlybirdFieldConstant.NUM\_MENTIONS.getFieldName());

sink.setNumericValue(

EarlybirdFieldConstant.NUM\_HASHTAGS,

Math.min(textFeatures.getHashtagsSize(), featureConfigNumHashtags.getMaxValue()))

.setNumericValue(

EarlybirdFieldConstant.NUM\_MENTIONS,

Math.min(textFeatures.getMentionsSize(), featureConfigNumMentions.getMaxValue()))

.setBooleanValue(

EarlybirdFieldConstant.HAS\_MULTIPLE\_HASHTAGS\_OR\_TRENDS\_FLAG,

TwitterMessage.hasMultipleHashtagsOrTrends(textFeatures))

.setBooleanValue(

EarlybirdFieldConstant.HAS\_TREND\_FLAG,

textFeatures.getTrendingTermsSize() > 0);

}

TweetTextQuality textQuality = message.getTweetTextQuality(penguinVersion);

if (textQuality != null) {

sink.setNumericValue(EarlybirdFieldConstant.TEXT\_SCORE, textQuality.getTextScore());

sink.setBooleanValue(

EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG,

textQuality.hasBoolQuality(TweetTextQuality.BooleanQualityType.OFFENSIVE)

|| textQuality.hasBoolQuality(TweetTextQuality.BooleanQualityType.OFFENSIVE\_USER)

// Note: if json message "possibly\_sensitive" flag is set, we consider the tweet

// sensitive and is currently filtered out in safe search mode via a hacky setup:

// earlybird does not create \_filter\_sensitive\_content field, only

// \_is\_offensive field is created, and used in filter:safe operator

|| textQuality.hasBoolQuality(TweetTextQuality.BooleanQualityType.SENSITIVE));

if (textQuality.hasBoolQuality(TweetTextQuality.BooleanQualityType.SENSITIVE)) {

sink.setBooleanValue(EarlybirdFieldConstant.IS\_SENSITIVE\_CONTENT, true);

}

} else {

// we don't have text score, for whatever reason, set to sentinel value so we won't be

// skipped by scoring function

sink.setNumericValue(EarlybirdFieldConstant.TEXT\_SCORE,

RelevanceSignalConstants.UNSET\_TEXT\_SCORE\_SENTINEL);

}

if (message.isSetLocale()) {

sink.setNumericValue(EarlybirdFieldConstant.LANGUAGE,

ThriftLanguageUtil.getThriftLanguageOf(message.getLocale()).getValue());

}

// User features

TweetUserFeatures userFeatures = message.getTweetUserFeatures(penguinVersion);

if (userFeatures != null) {

sink.setBooleanValue(EarlybirdFieldConstant.IS\_USER\_SPAM\_FLAG, userFeatures.isSpam())

.setBooleanValue(EarlybirdFieldConstant.IS\_USER\_NSFW\_FLAG, userFeatures.isNsfw())

.setBooleanValue(EarlybirdFieldConstant.IS\_USER\_BOT\_FLAG, userFeatures.isBot());

}

if (message.getUserReputation() != TwitterMessage.DOUBLE\_FIELD\_NOT\_PRESENT) {

sink.setNumericValue(EarlybirdFieldConstant.USER\_REPUTATION,

(byte) message.getUserReputation());

} else {

sink.setNumericValue(EarlybirdFieldConstant.USER\_REPUTATION,

RelevanceSignalConstants.UNSET\_REPUTATION\_SENTINEL);

}

sink.setBooleanValue(EarlybirdFieldConstant.IS\_NULLCAST\_FLAG, message.getNullcast());

// Realtime Ingestion does not write engagement features. Updater does that.

if (message.getNumFavorites() > 0) {

sink.setNumericValue(EarlybirdFieldConstant.FAVORITE\_COUNT,

MutableFeatureNormalizers.BYTE\_NORMALIZER.normalize(message.getNumFavorites()));

}

if (message.getNumRetweets() > 0) {

sink.setNumericValue(EarlybirdFieldConstant.RETWEET\_COUNT,

MutableFeatureNormalizers.BYTE\_NORMALIZER.normalize(message.getNumRetweets()));

}

if (message.getNumReplies() > 0) {

sink.setNumericValue(EarlybirdFieldConstant.REPLY\_COUNT,

MutableFeatureNormalizers.BYTE\_NORMALIZER.normalize(message.getNumReplies()));

}

sink.setNumericValue(EarlybirdFieldConstant.VISIBLE\_TOKEN\_RATIO, normalizedTokenPercentBucket);

EarlybirdEncodedFeatures encodedFeatures =

(EarlybirdEncodedFeatures) sink.getFeaturesForBaseField(

EarlybirdFieldConstant.ENCODED\_TWEET\_FEATURES\_FIELD.getFieldName());

updateLinkEncodedFeatures(encodedFeatures, message);

return encodedFeatures;

}

/\*\*

\* Returns the extended encoded features.

\*/

public static EarlybirdEncodedFeatures createExtendedEncodedFeaturesFromTwitterMessage(

TwitterMessage message,

PenguinVersion penguinVersion,

ImmutableSchemaInterface schema) {

FeatureSink sink = new FeatureSink(schema);

TweetTextFeatures textFeatures = message.getTweetTextFeatures(penguinVersion);

if (textFeatures != null) {

setExtendedEncodedFeatureIntValue(sink, schema,

EarlybirdFieldConstant.NUM\_HASHTAGS\_V2, textFeatures.getHashtagsSize());

setExtendedEncodedFeatureIntValue(sink, schema,

EarlybirdFieldConstant.NUM\_MENTIONS\_V2, textFeatures.getMentionsSize());

setExtendedEncodedFeatureIntValue(sink, schema,

EarlybirdFieldConstant.NUM\_STOCKS, textFeatures.getStocksSize());

}

Optional<Long> referenceAuthorId = message.getReferenceAuthorId();

if (referenceAuthorId.isPresent()) {

setEncodedReferenceAuthorId(sink, referenceAuthorId.get());

}

return (EarlybirdEncodedFeatures) sink.getFeaturesForBaseField(

EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD.getFieldName());

}

/\*\*

\* Updates all URL-related features, based on the values stored in the given message.

\*

\* @param encodedFeatures The features to be updated.

\* @param message The message.

\*/

public static void updateLinkEncodedFeatures(

EarlybirdEncodedFeatures encodedFeatures, TwitterMessage message) {

if (message.getLinkLocale() != null) {

encodedFeatures.setFeatureValue(

EarlybirdFieldConstant.LINK\_LANGUAGE,

ThriftLanguageUtil.getThriftLanguageOf(message.getLinkLocale()).getValue());

}

if (message.hasCard()) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_CARD\_FLAG);

}

// Set HAS\_IMAGE HAS\_NEWS HAS\_VIDEO etc. flags for expanded urls.

if (message.getExpandedUrlMapSize() > 0) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_LINK\_FLAG);

for (ThriftExpandedUrl url : message.getExpandedUrlMap().values()) {

if (url.isSetMediaType()) {

switch (url.getMediaType()) {

case NATIVE\_IMAGE:

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG);

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_NATIVE\_IMAGE\_FLAG);

break;

case IMAGE:

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG);

break;

case VIDEO:

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_VIDEO\_URL\_FLAG);

break;

case NEWS:

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_NEWS\_URL\_FLAG);

break;

case UNKNOWN:

break;

default:

throw new IllegalStateException("Unexpected enum value: " + url.getMediaType());

}

}

}

}

Set<String> canonicalLastHopUrlsStrings = message.getCanonicalLastHopUrls();

Set<String> expandedUrlsStrings = message.getExpandedUrls()

.stream()

.map(ThriftExpandedUrl::getExpandedUrl)

.collect(Collectors.toSet());

Set<String> expandedAndLastHopUrlsStrings = new HashSet<>();

expandedAndLastHopUrlsStrings.addAll(expandedUrlsStrings);

expandedAndLastHopUrlsStrings.addAll(canonicalLastHopUrlsStrings);

// Check both expanded and last hop url for consumer videos as consumer video urls are

// sometimes redirected to the url of the tweets containing the videos (SEARCH-42612).

if (NativeVideoClassificationUtils.hasConsumerVideo(expandedAndLastHopUrlsStrings)) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_CONSUMER\_VIDEO\_FLAG);

}

if (NativeVideoClassificationUtils.hasProVideo(canonicalLastHopUrlsStrings)) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_PRO\_VIDEO\_FLAG);

}

if (NativeVideoClassificationUtils.hasVine(canonicalLastHopUrlsStrings)) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_VINE\_FLAG);

}

if (NativeVideoClassificationUtils.hasPeriscope(canonicalLastHopUrlsStrings)) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_PERISCOPE\_FLAG);

}

if (LinkVisibilityUtils.hasVisibleLink(message.getExpandedUrls())) {

encodedFeatures.setFlag(EarlybirdFieldConstant.HAS\_VISIBLE\_LINK\_FLAG);

}

}

private static void setExtendedEncodedFeatureIntValue(

FeatureSink sink,

ImmutableSchemaInterface schema,

EarlybirdFieldConstant field,

int value) {

boolean fieldInSchema = schema.hasField(field.getFieldName());

if (fieldInSchema) {

FeatureConfiguration featureConfig =

schema.getFeatureConfigurationByName(field.getFieldName());

sink.setNumericValue(field, Math.min(value, featureConfig.getMaxValue()));

}

}

private static void setEncodedReferenceAuthorId(FeatureSink sink, long referenceAuthorId) {

LongIntConverter.IntegerRepresentation ints =

LongIntConverter.convertOneLongToTwoInt(referenceAuthorId);

sink.setNumericValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_LEAST\_SIGNIFICANT\_INT, ints.leastSignificantInt);

sink.setNumericValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_MOST\_SIGNIFICANT\_INT, ints.mostSignificantInt);

}

}