package com.twitter.search.common.relevance.classifiers;

import java.io.IOException;

import java.util.Set;

import com.google.common.base.Preconditions;

import com.twitter.common.text.transformer.RegexTransformer;

import com.twitter.common.text.transformer.RtRemovalTransformer;

import com.twitter.common.text.transformer.Transformer;

import com.twitter.common.text.transformer.TransformerChain;

import com.twitter.common\_internal.text.duplicate.RandomSubstringExtractor;

import com.twitter.common\_internal.text.duplicate.SignatureGenerator;

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

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

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

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

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

import com.twitter.twittertext.Regex;

/\*\*

\* Given a tweet text, extract useful text features.

\*/

public class TweetQualityFeatureExtractor {

private static final Transformer STATUS\_TEXT\_CLEANER =

TransformerChain.of(

// remove @reply as defined in twitter-text

new RegexTransformer.Builder()

.setRegexPattern(Regex.VALID\_REPLY)

.setReplaceString("")

.setTriggeringChar('@')

.build(),

// remove the old style retweet, eg RT: @mention or via @mention

new RtRemovalTransformer()

);

// for signature generation

private static final int MIN\_NUM\_FEATURES = 2;

private final SignatureGenerator signatureGenerator = new SignatureGenerator(

new RandomSubstringExtractor(

TweetIntegerShingleSignature.NUM\_SHINGLES, // number of signatures

MIN\_NUM\_FEATURES, // each signature is generated by taking this number of features/tokens

// from text

false, // do not consider full tweet text as a feature

false)); // do not do early termination

/\*\*

\* Given TwitterMessage, extract all interesting tweet text features and store in

\* the returned TweetTextFeatures object.

\*

\* @param tweet TwitterMessage to extract features from

\* @throws IOException

\*/

public void extractTweetTextFeatures(final TwitterMessage tweet) {

Preconditions.checkNotNull(tweet);

for (PenguinVersion penguinVersion : tweet.getSupportedPenguinVersions()) {

// Get basic features.

TweetTextFeatures textFeatures = tweet.getTweetTextFeatures(penguinVersion);

extractCharLength(textFeatures);

// Signature that hashes on text with resolved urls, aggressively remove RT tags, which

// accounts for more than 50% of neardups, also remove @mentions.

// we use resolved urls for signature since they are what matters.

CharSequence strippedText = tweet.getTextReplacedWithResolvedURLs();

strippedText = strippedText == null ? "" : strippedText;

strippedText = STATUS\_TEXT\_CLEANER.transform(strippedText);

// Generate the signature.

// will lower case, use penguin

String normalizedSignatureText =

NormalizerHelper.normalize(strippedText, tweet.getLocale(), penguinVersion);

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

Set<byte[]> rawSignature =

signatureGenerator.generateSignatureByteArray(normalizedSignatureText);

textFeatures.setSignature((new TweetIntegerShingleSignature(rawSignature)).serialize());

}

}

}

/\*\*

\* Compute number of letters in stripped tweet text, also records unsupported char counts.

\*

\* @param textFeatures TweetTextFeatures object to store letter length, unsupported chars, etc.

\*/

private static void extractCharLength(final TweetTextFeatures textFeatures) {

Preconditions.checkNotNull(textFeatures);

int length = 0;

int caps = 0;

String strippedText = textFeatures.getNormalizedStrippedText();

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

for (char c : strippedText.toCharArray()) {

if (Character.isLetter(c)) {

length++;

if (Character.isUpperCase(c)) {

caps++;

}

}

}

}

textFeatures.setLength(length);

textFeatures.setCaps(caps);

}

}