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

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Locale;

import java.util.Set;

import com.google.common.base.Joiner;

import com.google.common.collect.Sets;

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

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

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

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

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

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

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

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

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

/\*\*

\* A parser to extract very basic information from a tweet.

\*/

public class TweetParser {

private static final boolean DO\_NOT\_REMOVE\_WWW = false;

/\*\* Parses the given TwitterMessage. \*/

public void parseTweet(TwitterMessage message) {

parseTweet(message, false, true);

}

/\*\* Parses the given TwitterMessage. \*/

public void parseTweet(TwitterMessage message,

boolean useEntitiesFromTweetText,

boolean parseUrls) {

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

parseTweet(message, useEntitiesFromTweetText, parseUrls, penguinVersion);

}

}

/\*\* Parses the given TwitterMessage. \*/

public void parseTweet(TwitterMessage message,

boolean useEntitiesFromTweetText,

boolean parseUrls,

PenguinVersion penguinVersion) {

TweetTextFeatures textFeatures = message.getTweetTextFeatures(penguinVersion);

String rawText = message.getText();

Locale locale = message.getLocale();

// don't lower case first.

String normalizedText = NormalizerHelper.normalizeKeepCase(rawText, locale, penguinVersion);

String lowercasedNormalizedText =

CharSequenceUtils.toLowerCase(normalizedText, locale).toString();

textFeatures.setNormalizedText(lowercasedNormalizedText);

TokenizerResult result = TokenizerHelper.tokenizeTweet(normalizedText, locale, penguinVersion);

List<String> tokens = new ArrayList<>(result.tokens);

textFeatures.setTokens(tokens);

textFeatures.setTokenSequence(result.tokenSequence);

if (parseUrls) {

parseUrls(message, textFeatures);

}

textFeatures.setStrippedTokens(result.strippedDownTokens);

textFeatures.setNormalizedStrippedText(Joiner.on(" ").skipNulls()

.join(result.strippedDownTokens));

// Sanity checks, make sure there is no null token list.

if (textFeatures.getTokens() == null) {

textFeatures.setTokens(Collections.<String>emptyList());

}

if (textFeatures.getResolvedUrlTokens() == null) {

textFeatures.setResolvedUrlTokens(Collections.<String>emptyList());

}

if (textFeatures.getStrippedTokens() == null) {

textFeatures.setStrippedTokens(Collections.<String>emptyList());

}

setHashtagsAndMentions(message, textFeatures, penguinVersion);

textFeatures.setStocks(sanitizeTokenizerResults(result.stocks, '$'));

textFeatures.setHasQuestionMark(findQuestionMark(textFeatures));

// Set smiley polarities.

textFeatures.setSmileys(result.smileys);

for (String smiley : textFeatures.getSmileys()) {

if (Smileys.isValidSmiley(smiley)) {

boolean polarity = Smileys.getPolarity(smiley);

if (polarity) {

textFeatures.setHasPositiveSmiley(true);

} else {

textFeatures.setHasNegativeSmiley(true);

}

}

}

message.setTokenizedCharSequence(penguinVersion, result.rawSequence);

if (useEntitiesFromTweetText) {

takeEntities(message, textFeatures, result, penguinVersion);

}

}

/\*\* Parse the URLs in the given TwitterMessage. \*/

public void parseUrls(TwitterMessage message) {

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

parseUrls(message, message.getTweetTextFeatures(penguinVersion));

}

}

/\*\* Parse the URLs in the given TwitterMessage. \*/

public void parseUrls(TwitterMessage message, TweetTextFeatures textFeatures) {

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

Set<String> urlsToTokenize = Sets.newLinkedHashSet();

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

if (url.isSetExpandedUrl()) {

urlsToTokenize.add(url.getExpandedUrl());

}

if (url.isSetCanonicalLastHopUrl()) {

urlsToTokenize.add(url.getCanonicalLastHopUrl());

}

}

TokenizerResult resolvedUrlResult =

TokenizerHelper.tokenizeUrls(urlsToTokenize, message.getLocale(), DO\_NOT\_REMOVE\_WWW);

List<String> urlTokens = new ArrayList<>(resolvedUrlResult.tokens);

textFeatures.setResolvedUrlTokens(urlTokens);

}

}

private void takeEntities(TwitterMessage message,

TweetTextFeatures textFeatures,

TokenizerResult result,

PenguinVersion penguinVersion) {

if (message.getHashtags().isEmpty()) {

// add hashtags to TwitterMessage if it doens't already have them, from

// JSON entities, this happens when we do offline indexing

for (String hashtag : sanitizeTokenizerResults(result.hashtags, '#')) {

message.addHashtag(hashtag);

}

}

if (message.getMentions().isEmpty()) {

// add mentions to TwitterMessage if it doens't already have them, from

// JSON entities, this happens when we do offline indexing

for (String mention : sanitizeTokenizerResults(result.mentions, '@')) {

message.addMention(mention);

}

}

setHashtagsAndMentions(message, textFeatures, penguinVersion);

}

private void setHashtagsAndMentions(TwitterMessage message,

TweetTextFeatures textFeatures,

PenguinVersion penguinVersion) {

textFeatures.setHashtags(message.getNormalizedHashtags(penguinVersion));

textFeatures.setMentions(message.getLowercasedMentions());

}

// The strings in the mentions, hashtags and stocks lists in TokenizerResult should already have

// the leading characters ('@', '#' and '$') stripped. So in most cases, this sanitization is not

// needed. However, sometimes Penguin tokenizes hashtags, cashtags and mentions incorrectly

// (for example, when using the Korean tokenizer for tokens like ~@mention or ?#hashtag -- see

// SEARCHQUAL-11924 for more details). So we're doing this extra sanitization here to try to work

// around these tokenization issues.

private List<String> sanitizeTokenizerResults(List<String> tokens, char tokenSymbol) {

List<String> sanitizedTokens = new ArrayList<String>();

for (String token : tokens) {

int indexOfTokenSymbol = token.indexOf(tokenSymbol);

if (indexOfTokenSymbol < 0) {

sanitizedTokens.add(token);

} else {

String sanitizedToken = token.substring(indexOfTokenSymbol + 1);

if (!sanitizedToken.isEmpty()) {

sanitizedTokens.add(sanitizedToken);

}

}

}

return sanitizedTokens;

}

/\*\* Determines if the normalized text of the given features contain a question mark. \*/

public static boolean findQuestionMark(TweetTextFeatures textFeatures) {

// t.co links don't contain ?'s, so it's not necessary to subtract ?'s occurring in Urls

// the tweet text always contains t.co, even if the display url is different

// all links on twitter are now wrapped into t.co

return textFeatures.getNormalizedText().contains("?");

}

}