package com.twitter.timelineranker.util

import com.twitter.common.text.tagger.UniversalPOS

import com.twitter.common.text.token.attribute.TokenType

import com.twitter.common\_internal.text.pipeline.TwitterTextNormalizer

import com.twitter.common\_internal.text.pipeline.TwitterTextTokenizer

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

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

import com.twitter.search.common.util.text.PhraseExtractor

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

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

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.tweetypie.{thriftscala => tweetypie}

import com.twitter.util.Try

import java.util.Locale

import scala.collection.JavaConversions.\_

object TweetTextFeaturesExtractor {

private[this] val threadLocaltokenizer = new ThreadLocal[Option[TwitterTextTokenizer]] {

override protected def initialValue(): Option[TwitterTextTokenizer] =

Try {

val normalizer = new TwitterTextNormalizer.Builder(penguinVersion).build

TokenizerHelper

.getTokenizerBuilder(penguinVersion)

.enablePOSTagger

.enableStopwordFilterWithNormalizer(normalizer)

.setStopwordResourcePath("com/twitter/ml/feature/generator/stopwords\_extended\_{LANG}.txt")

.enableStemmer

.build

}.toOption

}

val penguinVersion: PenguinVersion = PenguinVersion.PENGUIN\_6

def addTextFeaturesFromTweet(

inputFeatures: ContentFeatures,

tweet: tweetypie.Tweet,

hydratePenguinTextFeatures: Boolean,

hydrateTokens: Boolean,

hydrateTweetText: Boolean

): ContentFeatures = {

tweet.coreData

.map { coreData =>

val tweetText = coreData.text

val hasQuestion = hasQuestionCharacter(tweetText)

val length = getLength(tweetText).toShort

val numCaps = getCaps(tweetText).toShort

val numWhiteSpaces = getSpaces(tweetText).toShort

val numNewlines = Some(getNumNewlines(tweetText))

val tweetTextOpt = getTweetText(tweetText, hydrateTweetText)

if (hydratePenguinTextFeatures) {

val locale = getLocale(tweetText)

val tokenizerOpt = threadLocaltokenizer.get

val tokenizerResult = tokenizerOpt.flatMap { tokenizer =>

tokenizeWithPosTagger(tokenizer, locale, tweetText)

}

val normalizedTokensOpt = if (hydrateTokens) {

tokenizerOpt.flatMap { tokenizer =>

tokenizedStringsWithNormalizerAndStemmer(tokenizer, locale, tweetText)

}

} else None

val emoticonTokensOpt = tokenizerResult.map(getEmoticons)

val emojiTokensOpt = tokenizerResult.map(getEmojis)

val posUnigramsOpt = tokenizerResult.map(getPosUnigrams)

val posBigramsOpt = posUnigramsOpt.map(getPosBigrams)

val tokensOpt = normalizedTokensOpt

inputFeatures.copy(

emojiTokens = emojiTokensOpt,

emoticonTokens = emoticonTokensOpt,

hasQuestion = hasQuestion,

length = length,

numCaps = numCaps,

numWhiteSpaces = numWhiteSpaces,

numNewlines = numNewlines,

posUnigrams = posUnigramsOpt.map(\_.toSet),

posBigrams = posBigramsOpt.map(\_.toSet),

tokens = tokensOpt.map(\_.toSeq),

tweetText = tweetTextOpt

)

} else {

inputFeatures.copy(

hasQuestion = hasQuestion,

length = length,

numCaps = numCaps,

numWhiteSpaces = numWhiteSpaces,

numNewlines = numNewlines,

tweetText = tweetTextOpt

)

}

}

.getOrElse(inputFeatures)

}

private def tokenizeWithPosTagger(

tokenizer: TwitterTextTokenizer,

locale: Locale,

text: String

): Option[TokenizerResult] = {

tokenizer.enableStemmer(false)

tokenizer.enableStopwordFilter(false)

Try { TokenizerHelper.tokenizeTweet(tokenizer, text, locale) }.toOption

}

private def tokenizedStringsWithNormalizerAndStemmer(

tokenizer: TwitterTextTokenizer,

locale: Locale,

text: String

): Option[Seq[String]] = {

tokenizer.enableStemmer(true)

tokenizer.enableStopwordFilter(true)

Try { tokenizer.tokenizeToStrings(text, locale).toSeq }.toOption

}

def getLocale(text: String): Locale = LanguageIdentifierHelper.identifyLanguage(text)

def getTokens(tokenizerResult: TokenizerResult): List[String] =

tokenizerResult.rawSequence.getTokenStrings().toList

def getEmoticons(tokenizerResult: TokenizerResult): Set[String] =

tokenizerResult.smileys.toSet

def getEmojis(tokenizerResult: TokenizerResult): Set[String] =

tokenizerResult.rawSequence.getTokenStringsOf(TokenType.EMOJI).toSet

def getPhrases(tokenizerResult: TokenizerResult, locale: Locale): Set[String] = {

PhraseExtractor.getPhrases(tokenizerResult.rawSequence, locale).map(\_.toString).toSet

}

def getPosUnigrams(tokenizerResult: TokenizerResult): List[String] =

tokenizerResult.tokenSequence.getTokens.toList

.map { token =>

Option(token.getPartOfSpeech)

.map(\_.toString)

.getOrElse(UniversalPOS.X.toString) // UniversalPOS.X is unknown POS tag

}

def getPosBigrams(tagsList: List[String]): List[String] = {

if (tagsList.nonEmpty) {

tagsList

.zip(tagsList.tail)

.map(tagPair => Seq(tagPair.\_1, tagPair.\_2).mkString(" "))

} else {

tagsList

}

}

def getLength(text: String): Int =

text.codePointCount(0, text.length())

def getCaps(text: String): Int = text.count(Character.isUpperCase)

def getSpaces(text: String): Int = text.count(Character.isWhitespace)

def hasQuestionCharacter(text: String): Boolean = {

// List based on https://unicode-search.net/unicode-namesearch.pl?term=question

val QUESTION\_MARK\_CHARS = Seq(

"\u003F",

"\u00BF",

"\u037E",

"\u055E",

"\u061F",

"\u1367",

"\u1945",

"\u2047",

"\u2048",

"\u2049",

"\u2753",

"\u2754",

"\u2CFA",

"\u2CFB",

"\u2E2E",

"\uA60F",

"\uA6F7",

"\uFE16",

"\uFE56",

"\uFF1F",

"\u1114",

"\u1E95"

)

QUESTION\_MARK\_CHARS.exists(text.contains)

}

def getNumNewlines(text: String): Short = {

val newlineRegex = "\r\n|\r|\n".r

newlineRegex.findAllIn(text).length.toShort

}

private[this] def getTweetText(tweetText: String, hydrateTweetText: Boolean): Option[String] = {

if (hydrateTweetText) Some(tweetText) else None

}

}