package com.twitter.tweetypie.matching

import com.twitter.common.text.language.LocaleUtil

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.concurrent.Once

import com.twitter.io.StreamIO

import java.util.Locale

import scala.collection.JavaConverters.\_

/\*\*

\* Extract a sequence of normalized tokens from the input text. The

\* normalization and tokenization are properly configured for keyword

\* matching between texts.

\*/

trait Tokenizer {

def tokenize(input: String): TokenSequence

}

object Tokenizer {

/\*\*

\* When a Penguin version is not explicitly specified, use this

\* version of Penguin to perform normalization and tokenization. If

\* you cache tokenized text, be sure to store the version as well, to

\* avoid comparing text that was normalized with different algorithms.

\*/

val DefaultPenguinVersion: PenguinVersion = PenguinVersion.PENGUIN\_6

/\*\*

\* If you already know the locale of the text that is being tokenized,

\* use this method to get a tokenizer that is much more efficient than

\* the Tweet or Query tokenizer, since it does not have to perform

\* language detection.

\*/

def forLocale(locale: Locale): Tokenizer = get(locale, DefaultPenguinVersion)

/\*\*

\* Obtain a `Tokenizer` that will tokenize the text for the given

\* locale and version of the Penguin library.

\*/

def get(locale: Locale, version: PenguinVersion): Tokenizer =

TokenizerFactories(version).forLocale(locale)

/\*\*

\* Encapsulates the configuration and use of [[TwitterTextTokenizer]]

\* and [[TwitterTextNormalizer]].

\*/

private[this] class TokenizerFactory(version: PenguinVersion) {

// The normalizer is thread-safe, so share one instance.

private[this] val normalizer =

(new TwitterTextNormalizer.Builder(version)).build()

// The TwitterTextTokenizer is relatively expensive to build,

// and is not thread safe, so keep instances of it in a

// ThreadLocal.

private[this] val local =

new ThreadLocal[TwitterTextTokenizer] {

override def initialValue: TwitterTextTokenizer =

(new TwitterTextTokenizer.Builder(version)).build()

}

/\*\*

\* Obtain a [[Tokenizer]] for this combination of [[PenguinVersion]]

\* and [[Locale]].

\*/

def forLocale(locale: Locale): Tokenizer =

new Tokenizer {

override def tokenize(input: String): TokenSequence = {

val stream = local.get.getTwitterTokenStreamFor(locale)

stream.reset(normalizer.normalize(input, locale))

val builder = IndexedSeq.newBuilder[CharSequence]

while (stream.incrementToken) builder += stream.term()

TokenSequence(builder.result())

}

}

}

/\*\*

\* Since there are a small number of Penguin versions, eagerly

\* initialize a TokenizerFactory for each version, to avoid managing

\* mutable state.

\*/

private[this] val TokenizerFactories: PenguinVersion => TokenizerFactory =

PenguinVersion.values.map(v => v -> new TokenizerFactory(v)).toMap

/\*\*

\* The set of locales used in warmup. These locales are mentioned in

\* the logic of TwitterTextTokenizer and TwitterTextNormalizer.

\*/

private[this] val WarmUpLocales: Seq[Locale] =

Seq

.concat(

Seq(

Locale.JAPANESE,

Locale.KOREAN,

LocaleUtil.UNKNOWN,

LocaleUtil.THAI,

LocaleUtil.ARABIC,

LocaleUtil.SWEDISH

),

LocaleUtil.CHINESE\_JAPANESE\_LOCALES.asScala,

LocaleUtil.CJK\_LOCALES.asScala

)

.toSet

.toArray

.toSeq

/\*\*

\* Load the default inputs that are used for warming up this library.

\*/

def warmUpCorpus(): Seq[String] = {

val stream = getClass.getResourceAsStream("warmup-text.txt")

val bytes =

try StreamIO.buffer(stream)

finally stream.close()

bytes.toString("UTF-8").linesIterator.toArray.toSeq

}

/\*\*

\* Exercise the functionality of this library on the specified

\* strings. In general, prefer [[warmUp]] to this method.

\*/

def warmUpWith(ver: PenguinVersion, texts: Iterable[String]): Unit =

texts.foreach { txt =>

// Exercise each locale

WarmUpLocales.foreach { loc =>

Tokenizer.get(loc, ver).tokenize(txt)

UserMutes.builder().withPenguinVersion(ver).withLocale(loc).validate(txt)

}

// Exercise language detection

TweetTokenizer.get(ver).tokenize(txt)

UserMutes.builder().withPenguinVersion(ver).validate(txt)

}

private[this] val warmUpOnce = Once(warmUpWith(DefaultPenguinVersion, warmUpCorpus()))

/\*\*

\* The creation of the first TwitterTextTokenizer is relatively

\* expensive, and tokenizing some texts may cause significant

\* initialization.

\*

\* This method exercises the functionality of this library

\* with a range of texts in order to perform as much initialization as

\* possible before the library is used in a latency-sensitive way.

\*

\* The warmup routine will only run once. Subsequent invocations of

\* `warmUp` will no do additional work, and will return once warmup is

\* complete.

\*

\* The warmup will take on the order of seconds.

\*/

def warmUp(): Unit = warmUpOnce()

}