package com.twitter.tweetypie.matching

import com.twitter.common.text.pipeline.TwitterLanguageIdentifier

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

import java.util.Locale

import scala.collection.JavaConversions.asScalaBuffer

object UserMutesBuilder {

private[matching] val Default =

new UserMutesBuilder(Tokenizer.DefaultPenguinVersion, None)

private val queryLangIdentifier =

(new TwitterLanguageIdentifier.Builder).buildForQuery()

}

class UserMutesBuilder private (penguinVersion: PenguinVersion, localeOpt: Option[Locale]) {

/\*\*

\* Use the specified Penguin version when tokenizing a keyword mute

\* string. In general, use the default version, unless you need to

\* specify a particular version for compatibility with another system

\* that is using that version.

\*/

def withPenguinVersion(ver: PenguinVersion): UserMutesBuilder =

if (ver == penguinVersion) this

else new UserMutesBuilder(ver, localeOpt)

/\*\*

\* Use the specified locale when tokenizing a keyword mute string.

\*/

def withLocale(locale: Locale): UserMutesBuilder =

if (localeOpt.contains(locale)) this

else new UserMutesBuilder(penguinVersion, Some(locale))

/\*\*

\* When tokenizing a user mute list, detect the language of the

\* text. This is significantly more expensive than using a predefined

\* locale, but is appropriate when the locale is not yet known.

\*/

def detectLocale(): UserMutesBuilder =

if (localeOpt.isEmpty) this

else new UserMutesBuilder(penguinVersion, localeOpt)

private[this] lazy val tokenizer =

localeOpt match {

case None =>

// No locale was specified, so use a Tokenizer that performs

// language detection before tokenizing.

new Tokenizer {

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

val locale = UserMutesBuilder.queryLangIdentifier.identify(text).getLocale

Tokenizer.get(locale, penguinVersion).tokenize(text)

}

}

case Some(locale) =>

Tokenizer.get(locale, penguinVersion)

}

/\*\*

\* Given a list of the user's raw keyword mutes, return a preprocessed

\* set of mutes suitable for matching against tweet text. If the input

\* contains any phrases that fail validation, then they will be

\* dropped.

\*/

def build(rawInput: Seq[String]): UserMutes =

UserMutes(rawInput.flatMap(validate(\_).right.toOption))

/\*\*

\* Java-friendly API for processing a user's list of raw keyword mutes

\* into a preprocessed form suitable for matching against text.

\*/

def fromJavaList(rawInput: java.util.List[String]): UserMutes =

build(asScalaBuffer(rawInput).toSeq)

/\*\*

\* Validate the raw user input muted phrase. Currently, the only

\* inputs that are not valid for keyword muting are those inputs that

\* do not contain any keywords, because those inputs would match all

\* tweets.

\*/

def validate(mutedPhrase: String): Either[UserMutes.ValidationError, TokenSequence] = {

val keywords = tokenizer.tokenize(mutedPhrase)

if (keywords.isEmpty) UserMutes.EmptyPhraseError else Right(keywords)

}

}

object UserMutes {

sealed trait ValidationError

/\*\*

\* The phrase's tokenization did not produce any tokens

\*/

case object EmptyPhrase extends ValidationError

private[matching] val EmptyPhraseError = Left(EmptyPhrase)

/\*\*

\* Get a [[UserMutesBuilder]] that uses the default Penguin version and

\* performs language identification to choose a locale.

\*/

def builder(): UserMutesBuilder = UserMutesBuilder.Default

}

/\*\*

\* A user's muted keyword list, preprocessed into token sequences.

\*/

case class UserMutes private[matching] (toSeq: Seq[TokenSequence]) {

/\*\*

\* Do any of the users' muted keyword sequences occur within the

\* supplied text?

\*/

def matches(text: TokenSequence): Boolean =

toSeq.exists(text.containsKeywordSequence)

/\*\*

\* Find all positions of matching muted keyword from the user's

\* muted keyword list

\*/

def find(text: TokenSequence): Seq[Int] =

toSeq.zipWithIndex.collect {

case (token, index) if text.containsKeywordSequence(token) => index

}

def isEmpty: Boolean = toSeq.isEmpty

def nonEmpty: Boolean = toSeq.nonEmpty

}