package com.twitter.home\_mixer.util

import com.twitter.search.common.constants.{thriftscala => scc}

import com.twitter.search.common.util.lang.ThriftLanguageUtil

import com.twitter.service.metastore.gen.{thriftscala => smg}

object LanguageUtil {

private val DefaultMinProducedLanguageRatio = 0.05

private val DefaultMinConsumedLanguageConfidence = 0.8

/\*\*

\* Computes a list of languages based on UserLanguages information retrieved from Metastore.

\*

\* The list is sorted in descending order of confidence score associated with each language.

\* That is, language with highest confidence value is in index 0.

\*/

def computeLanguages(

userLanguages: smg.UserLanguages,

minProducedLanguageRatio: Double = DefaultMinProducedLanguageRatio,

minConsumedLanguageConfidence: Double = DefaultMinConsumedLanguageConfidence

): Seq[scc.ThriftLanguage] = {

val languageConfidenceMap = computeLanguageConfidenceMap(

userLanguages,

minProducedLanguageRatio,

minConsumedLanguageConfidence

)

languageConfidenceMap.toSeq.sortBy(-\_.\_2).map(\_.\_1) // \_1 = language, \_2 = score

}

/\*\*

\* Computes confidence map based on UserLanguages information retrieved from Metastore.

\* where,

\* key = language code

\* value = level of confidence that the language is applicable to a user.

\*/

private def computeLanguageConfidenceMap(

userLanguages: smg.UserLanguages,

minProducedLanguageRatio: Double,

minConsumedLanguageConfidence: Double

): Map[scc.ThriftLanguage, Double] = {

val producedLanguages = getLanguageMap(userLanguages.produced)

val consumedLanguages = getLanguageMap(userLanguages.consumed)

val languages = (producedLanguages.keys ++ consumedLanguages.keys).toSet

var maxConfidence = 0.0

val confidenceMap = languages.map { language =>

val produceRatio = producedLanguages

.get(language)

.map { score => if (score < minProducedLanguageRatio) 0.0 else score }

.getOrElse(0.0)

val consumeConfidence = consumedLanguages

.get(language)

.map { score => if (score < minConsumedLanguageConfidence) 0.0 else score }

.getOrElse(0.0)

val overallConfidence = (0.3 + 4 \* produceRatio) \* (0.1 + consumeConfidence)

maxConfidence = Math.max(maxConfidence, overallConfidence)

(language -> overallConfidence)

}.toMap

val normalizedConfidenceMap = if (maxConfidence > 0) {

confidenceMap.map {

case (language, confidenceScore) =>

val normalizedScore = (confidenceScore / maxConfidence \* 0.9) + 0.1

(language -> normalizedScore)

}

} else {

confidenceMap

}

normalizedConfidenceMap

}

private def getLanguageMap(

scoredLanguages: Seq[smg.ScoredString]

): Map[scc.ThriftLanguage, Double] = {

scoredLanguages.flatMap { scoredLanguage =>

getThriftLanguage(scoredLanguage.item).map { language => (language -> scoredLanguage.weight) }

}.toMap

}

private def getThriftLanguage(languageName: String): Option[scc.ThriftLanguage] = {

val languageOrdinal = ThriftLanguageUtil.getThriftLanguageOf(languageName).ordinal

val language = scc.ThriftLanguage(languageOrdinal)

language match {

case scc.ThriftLanguage.Unknown => None

case \_ => Some(language)

}

}

}