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

import java.util.List;

import java.util.Map;

import java.util.function.Function;

import java.util.stream.Collectors;

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

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

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

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

/\*\*

\* Calculates entropy of tweet text based on tokens.

\*/

public class TweetTextEvaluator extends TweetEvaluator {

@Override

public void evaluate(final TwitterMessage tweet) {

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

TweetTextFeatures textFeatures = tweet.getTweetTextFeatures(penguinVersion);

TweetTextQuality textQuality = tweet.getTweetTextQuality(penguinVersion);

double readability = 0;

int numKeptWords = textFeatures.getStrippedTokensSize();

for (String token : textFeatures.getStrippedTokens()) {

readability += token.length();

}

if (numKeptWords > 0) {

readability = readability \* Math.log(numKeptWords) / numKeptWords;

}

textQuality.setReadability(readability);

textQuality.setEntropy(entropy(textFeatures.getStrippedTokens()));

textQuality.setShout(textFeatures.getCaps() / Math.max(textFeatures.getLength(), 1.0d));

}

}

private static double entropy(List<String> tokens) {

Map<String, Long> tokenCounts =

tokens.stream().collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));

int numItems = tokens.size();

double entropy = 0;

for (long count : tokenCounts.values()) {

double prob = (double) count / numItems;

entropy -= prob \* log2(prob);

}

return entropy;

}

private static double log2(double n) {

return Math.log(n) / Math.log(2);

}

}