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

import java.util.List;

import java.util.concurrent.TimeUnit;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableMap;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import com.twitter.finagle.mtls.authentication.ServiceIdentifier;

import com.twitter.search.common.metrics.RelevanceStats;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.relevance.NGramCache;

import com.twitter.search.common.relevance.TrendsThriftDataServiceManager;

import com.twitter.search.common.relevance.config.TweetProcessingConfig;

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

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

import com.twitter.util.Duration;

/\*\*

\* Determines if tweets contains trending terms.

\* Sets corresponding bits and fields to TweetTextFeatures.

\*/

public class TweetTrendsExtractor {

// The amount of time before filling the trends cache for the first time.

private static final long INIT\_TRENDS\_CACHE\_DELAY = 0;

private static final Logger LOG = LoggerFactory.getLogger(TweetTrendsExtractor.class.getName());

private static final int LOGGING\_INTERVAL = 100000;

// Singleton trends data service. This is the default service used unless a different

// instance is injected in the constructor.

private static volatile TrendsThriftDataServiceManager trendsDataServiceSingleton;

// trends cache used for extracting trends from tweets

private static volatile ImmutableMap<PenguinVersion, NGramCache> trendsCaches;

private static synchronized void initTrendsDataServiceInstance(

ServiceIdentifier serviceIdentifier,

List<PenguinVersion> supportedPenguinVersions) {

if (trendsDataServiceSingleton == null) {

TweetProcessingConfig.init();

if (trendsCaches == null) {

ImmutableMap.Builder<PenguinVersion, NGramCache> trendsCachesBuilder =

ImmutableMap.builder();

for (PenguinVersion penguinVersion : supportedPenguinVersions) {

NGramCache cache = NGramCache.builder()

.maxCacheSize(

TweetProcessingConfig.getInt("trends\_extractor\_num\_trends\_to\_cache", 5000))

.penguinVersion(penguinVersion)

.build();

trendsCachesBuilder.put(penguinVersion, cache);

}

trendsCaches = trendsCachesBuilder.build();

}

long rawTimeout = TweetProcessingConfig.getLong("trends\_extractor\_timeout\_msec", 200);

long rawInterval =

TweetProcessingConfig.getLong("trends\_extractor\_reload\_interval\_sec", 600L);

trendsDataServiceSingleton =

TrendsThriftDataServiceManager.newInstance(

serviceIdentifier,

TweetProcessingConfig.getInt("trends\_extractor\_retry", 2),

Duration.apply(rawTimeout, TimeUnit.MILLISECONDS),

Duration.apply(INIT\_TRENDS\_CACHE\_DELAY, TimeUnit.SECONDS),

Duration.apply(rawInterval, TimeUnit.SECONDS),

trendsCaches.values().asList()

);

trendsDataServiceSingleton.startAutoRefresh();

LOG.info("Started trend extractor.");

}

}

public TweetTrendsExtractor(

ServiceIdentifier serviceIdentifier,

List<PenguinVersion> supportedPenguinVersions) {

initTrendsDataServiceInstance(serviceIdentifier, supportedPenguinVersions);

}

/\*\*

\* Extract trending terms from the specified tweet.

\* @param tweet the specified tweet

\*/

public void extractTrends(TwitterMessage tweet) {

extractTrends(ImmutableList.of(tweet));

}

/\*\*

\* Extract trending terms from the specified list of tweets.

\* @param tweets a list of tweets

\*/

public void extractTrends(Iterable<TwitterMessage> tweets) {

Preconditions.checkNotNull(tweets);

for (TwitterMessage tweet : tweets) {

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

NGramCache trendsCache = trendsCaches.get(penguinVersion);

if (trendsCache == null) {

LOG.info("Trends cache for Penguin version " + penguinVersion + " is null.");

continue;

} else if (trendsCache.numTrendingTerms() == 0) {

LOG.info("Trends cache for Penguin version " + penguinVersion + " is empty.");

continue;

}

List<String> trendsInTweet = trendsCache.extractTrendsFrom(

tweet.getTokenizedCharSequence(penguinVersion), tweet.getLocale());

TweetTextFeatures textFeatures = tweet.getTweetTextFeatures(penguinVersion);

if (textFeatures == null || textFeatures.getTokens() == null) {

continue;

}

textFeatures.getTrendingTerms().addAll(trendsInTweet);

updateTrendsStats(

tweet,

textFeatures,

penguinVersion,

RelevanceStats.exportLong(

"trends\_extractor\_has\_trends\_" + penguinVersion.name().toLowerCase()),

RelevanceStats.exportLong(

"trends\_extractor\_no\_trends\_" + penguinVersion.name().toLowerCase()),

RelevanceStats.exportLong(

"trends\_extractor\_too\_many\_trends\_" + penguinVersion.name().toLowerCase()));

}

}

}

private void updateTrendsStats(TwitterMessage tweet,

TweetTextFeatures textFeatures,

PenguinVersion penguinVersion,

SearchCounter hasTrendsCounterToUpdate,

SearchCounter noTrendsCounterToUpdate,

SearchCounter tooManyTrendsCounterToUpdate) {

int numTrendingTerms = textFeatures.getTrendingTerms().size();

if (numTrendingTerms == 0) {

noTrendsCounterToUpdate.increment();

} else {

if (numTrendingTerms > 1) {

tooManyTrendsCounterToUpdate.increment();

}

hasTrendsCounterToUpdate.increment();

}

long counter = noTrendsCounterToUpdate.get();

if (counter % LOGGING\_INTERVAL == 0) {

long hasTrends = hasTrendsCounterToUpdate.get();

long noTrends = noTrendsCounterToUpdate.get();

long tooManyTrends = tooManyTrendsCounterToUpdate.get();

double ratio = 100.0d \* hasTrends / (hasTrends + noTrends + 1);

double tooManyTrendsRatio = 100.0d \* tooManyTrends / (hasTrends + 1);

LOG.info(String.format(

"Has trends %d, no trends %d, ratio %.2f, too many trends %.2f,"

+ " sample tweet id [%d] matching terms [%s] penguin version [%s]",

hasTrends, noTrends, ratio, tooManyTrendsRatio, tweet.getId(),

textFeatures.getTrendingTerms(), penguinVersion));

}

}

}