package com.twitter.search.earlybird.search.facets;

import java.io.IOException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

import com.twitter.search.common.ranking.thriftjava.ThriftFacetEarlybirdSortingMode;

import com.twitter.search.common.ranking.thriftjava.ThriftFacetRankingOptions;

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

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

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

import com.twitter.search.core.earlybird.facets.FacetAccumulator;

import com.twitter.search.core.earlybird.facets.FacetCountIterator;

import com.twitter.search.core.earlybird.facets.FacetLabelProvider;

import com.twitter.search.core.earlybird.index.EarlybirdIndexSegmentAtomicReader;

import com.twitter.search.earlybird.search.AntiGamingFilter;

import com.twitter.search.earlybird.search.facets.FacetResultsCollector.Accumulator;

import com.twitter.search.earlybird.thrift.ThriftSearchQuery;

public class DefaultFacetScorer extends FacetScorer {

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

private static final double DEFAULT\_FEATURE\_WEIGHT = 0.0;

private static final byte DEFAULT\_PENALTY = 1;

private static final byte DEFAULT\_REPUTATION\_MIN = 45;

private final AntiGamingFilter antiGamingFilter;

// tweepcreds below this value will not be counted at all

private final byte reputationMinFilterThresholdVal;

// tweepcreds between reputationMinFilterThresholdVal and this value will be counted

// with a score of 1

private final byte reputationMinScoreVal;

private final double userRepWeight;

private final double favoritesWeight;

private final double parusWeight;

private final double parusBase;

private final double queryIndependentPenaltyWeight;

private final ThriftLanguage uiLang;

private final double langEnglishUIBoost;

private final double langEnglishFacetBoost;

private final double langDefaultBoost;

private final int antigamingPenalty;

private final int offensiveTweetPenalty;

private final int multipleHashtagsOrTrendsPenalty;

private final int maxScorePerTweet;

private final ThriftFacetEarlybirdSortingMode sortingMode;

private EarlybirdIndexSegmentAtomicReader reader;

private EarlybirdDocumentFeatures features;

/\*\*

\* Creates a new facet scorer.

\*/

public DefaultFacetScorer(ThriftSearchQuery searchQuery,

ThriftFacetRankingOptions rankingOptions,

AntiGamingFilter antiGamingFilter,

ThriftFacetEarlybirdSortingMode sortingMode) {

this.sortingMode = sortingMode;

this.antiGamingFilter = antiGamingFilter;

maxScorePerTweet =

rankingOptions.isSetMaxScorePerTweet()

? rankingOptions.getMaxScorePerTweet()

: Integer.MAX\_VALUE;

// filters

reputationMinFilterThresholdVal =

rankingOptions.isSetMinTweepcredFilterThreshold()

? (byte) (rankingOptions.getMinTweepcredFilterThreshold() & 0xFF)

: DEFAULT\_REPUTATION\_MIN;

// weights

// reputationMinScoreVal must be >= reputationMinFilterThresholdVal

reputationMinScoreVal =

(byte) Math.max(rankingOptions.isSetReputationParams()

? (byte) rankingOptions.getReputationParams().getMin()

: DEFAULT\_REPUTATION\_MIN, reputationMinFilterThresholdVal);

parusWeight =

rankingOptions.isSetParusScoreParams() && rankingOptions.getParusScoreParams().isSetWeight()

? rankingOptions.getParusScoreParams().getWeight()

: DEFAULT\_FEATURE\_WEIGHT;

// compute this once so that base \*\* parusScore is backwards-compatible

parusBase = Math.sqrt(1 + parusWeight);

userRepWeight =

rankingOptions.isSetReputationParams() && rankingOptions.getReputationParams().isSetWeight()

? rankingOptions.getReputationParams().getWeight()

: DEFAULT\_FEATURE\_WEIGHT;

favoritesWeight =

rankingOptions.isSetFavoritesParams() && rankingOptions.getFavoritesParams().isSetWeight()

? rankingOptions.getFavoritesParams().getWeight()

: DEFAULT\_FEATURE\_WEIGHT;

queryIndependentPenaltyWeight =

rankingOptions.isSetQueryIndependentPenaltyWeight()

? rankingOptions.getQueryIndependentPenaltyWeight()

: DEFAULT\_FEATURE\_WEIGHT;

// penalty increment

antigamingPenalty =

rankingOptions.isSetAntigamingPenalty()

? rankingOptions.getAntigamingPenalty()

: DEFAULT\_PENALTY;

offensiveTweetPenalty =

rankingOptions.isSetOffensiveTweetPenalty()

? rankingOptions.getOffensiveTweetPenalty()

: DEFAULT\_PENALTY;

multipleHashtagsOrTrendsPenalty =

rankingOptions.isSetMultipleHashtagsOrTrendsPenalty()

? rankingOptions.getMultipleHashtagsOrTrendsPenalty()

: DEFAULT\_PENALTY;

// query information

if (!searchQuery.isSetUiLang() || searchQuery.getUiLang().isEmpty()) {

uiLang = ThriftLanguage.UNKNOWN;

} else {

uiLang = ThriftLanguageUtil.getThriftLanguageOf(searchQuery.getUiLang());

}

langEnglishUIBoost = rankingOptions.getLangEnglishUIBoost();

langEnglishFacetBoost = rankingOptions.getLangEnglishFacetBoost();

langDefaultBoost = rankingOptions.getLangDefaultBoost();

}

@Override

protected void startSegment(EarlybirdIndexSegmentAtomicReader segmentReader) throws IOException {

reader = segmentReader;

features = new EarlybirdDocumentFeatures(reader);

if (antiGamingFilter != null) {

antiGamingFilter.startSegment(reader);

}

}

@Override

public void incrementCounts(Accumulator accumulator, int internalDocID) throws IOException {

FacetCountIterator.IncrementData data = accumulator.accessor.incrementData;

data.accumulators = accumulator.accumulators;

features.advance(internalDocID);

// Also keep track of the tweet language of tweet themselves.

data.languageId = (int) features.getFeatureValue(EarlybirdFieldConstant.LANGUAGE);

if (antigamingPenalty > 0

&& antiGamingFilter != null

&& !antiGamingFilter.accept(internalDocID)) {

data.weightedCountIncrement = 0;

data.penaltyIncrement = antigamingPenalty;

data.tweepCred = 0;

accumulator.accessor.collect(internalDocID);

return;

}

if (offensiveTweetPenalty > 0 && features.isFlagSet(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG)) {

data.weightedCountIncrement = 0;

data.penaltyIncrement = offensiveTweetPenalty;

data.tweepCred = 0;

accumulator.accessor.collect(internalDocID);

return;

}

byte userRep = (byte) features.getFeatureValue(EarlybirdFieldConstant.USER\_REPUTATION);

if (userRep < reputationMinFilterThresholdVal) {

// don't penalize

data.weightedCountIncrement = 0;

data.penaltyIncrement = 0;

data.tweepCred = 0;

accumulator.accessor.collect(internalDocID);

return;

}

// Other non-terminating penalties

int penalty = 0;

if (multipleHashtagsOrTrendsPenalty > 0

&& features.isFlagSet(EarlybirdFieldConstant.HAS\_MULTIPLE\_HASHTAGS\_OR\_TRENDS\_FLAG)) {

penalty += multipleHashtagsOrTrendsPenalty;

}

double parus = 0xFF & (byte) features.getFeatureValue(EarlybirdFieldConstant.PARUS\_SCORE);

double score = Math.pow(1 + userRepWeight, Math.max(0, userRep - reputationMinScoreVal));

if (parus > 0) {

score += Math.pow(parusBase, parus);

}

int favoriteCount =

(int) features.getUnnormalizedFeatureValue(EarlybirdFieldConstant.FAVORITE\_COUNT);

if (favoriteCount > 0) {

score += favoriteCount \* favoritesWeight;

}

// Language preferences

int tweetLinkLangId = (int) features.getFeatureValue(EarlybirdFieldConstant.LINK\_LANGUAGE);

if (tweetLinkLangId == ThriftLanguage.UNKNOWN.getValue()) {

// fall back to use the tweet language itself.

tweetLinkLangId = (int) features.getFeatureValue(EarlybirdFieldConstant.LANGUAGE);

}

if (uiLang != ThriftLanguage.UNKNOWN && uiLang.getValue() != tweetLinkLangId) {

if (uiLang == ThriftLanguage.ENGLISH) {

score \*= langEnglishUIBoost;

} else if (tweetLinkLangId == ThriftLanguage.ENGLISH.getValue()) {

score \*= langEnglishFacetBoost;

} else {

score \*= langDefaultBoost;

}

}

// make sure a single tweet can't contribute too high a score

if (score > maxScorePerTweet) {

score = maxScorePerTweet;

}

data.weightedCountIncrement = (int) score;

data.penaltyIncrement = penalty;

data.tweepCred = userRep & 0xFF;

accumulator.accessor.collect(internalDocID);

}

@Override

public FacetAccumulator getFacetAccumulator(FacetLabelProvider labelProvider) {

return new HashingAndPruningFacetAccumulator(labelProvider, queryIndependentPenaltyWeight,

HashingAndPruningFacetAccumulator.getComparator(sortingMode));

}

}