package com.twitter.search.earlybird.search.relevance.scoring;

import java.io.IOException;

import org.apache.lucene.search.Explanation;

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

import com.twitter.search.common.schema.base.ImmutableSchemaInterface;

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

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

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

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

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

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

/\*\*

\* A toptweets query cache index selection scoring function that is based purely on retweet counts.

\* The goal of this scoring functon is to deprecate itweet score in entirety.

\*

\* Once all legacy itweet scores are drained from existing earlybird index, new parus score replaces

\* existing itweet score position, then this class will be deprecated, a new scoring function

\* using parus score shall replace this.

\*

\* this scoring function is only used in Query Cache for marking top tweets

\* in the background. When searched, those tweets are still ranked with linear or model-based

\* scoring function.

\*

\*/

public class RetweetBasedTopTweetsScoringFunction extends ScoringFunction {

private static final double DEFAULT\_RECENCY\_SCORE\_FRACTION = 0.1;

private static final double DEFAULT\_SIGMOID\_APLHA = 0.008;

private static final int DEFAULT\_RECENCY\_CENTER\_MINUTES = 1080;

// if you update the default cut off, make sure you update the query cache filter in

// querycache.yml

//

// we know currently each time slice, each partition has about 10K entries in toptweets query

// cache. These are unique tweets. Looking at retweet updates, each time slice, each partition has

// about 650K unique tweets that received retweet. To create roughly similar number of entries in

// query cache, we need top 2% of such tweets, and that sets to min retweet count to 4.

// In this linear scoring function, we will rescale retweet count to [0, 1] range,

// with an input range of [0, 20]. Given the realtime factor's weight of 0.1, that give our

// minimal retweet score threshold to: 4/20 \* 0.9 = 0.18.

// Testing on prod showed much higher volume due to the generous setting of max value of 20,

// (highest we have seen is 14). Adjusted to 0.21 which gave us similar volume.

private static final double DEFAULT\_CUT\_OFF\_SCORE = 0.21;

// Normalize retweet counts from [0, 20] range to [0, 1] range

private static final double MAX\_RETWEET\_COUNT = 20.0;

private static final double MIN\_USER\_REPUTATION = 40.0; // matches itweet system threshold

/\*\*

\* The scores for the retweet based top tweets have to be in the [0, 1] interval. So we can't use

\* SKIP\_HIT as the lowest possible score, and instead have to use Float.MIN\_VALUE.

\*

\* It's OK to use different values for these constants, because they do not interfere with each

\* other. This constant is only used in RetweetBasedTopTweetsScoringFunction, which is only used

\* to filter the hits for the [score\_filter retweets minScore maxScore] operator. So the scores

\* returned by RetweetBasedTopTweetsScoringFunction.score() do not have any impact on the final

\* hit score.

\*

\* See EarlybirdLuceneQueryVisitor.visitScoredFilterOperator() and ScoreFilterQuery for more details.

\*/

private static final float RETWEET\_BASED\_TOP\_TWEETS\_LOWEST\_SCORE = Float.MIN\_VALUE;

private final double recencyScoreFraction;

private final double sigmoidAlpha;

private final double cutOffScore;

private final int recencyCenterMinutes;

private final double maxRecency;

private final int currentTimeSeconds;

private ThriftSearchResultMetadata metadata = null;

private double score;

private double retweetCount;

public RetweetBasedTopTweetsScoringFunction(ImmutableSchemaInterface schema) {

this(schema, DEFAULT\_RECENCY\_SCORE\_FRACTION,

DEFAULT\_SIGMOID\_APLHA,

DEFAULT\_CUT\_OFF\_SCORE,

DEFAULT\_RECENCY\_CENTER\_MINUTES);

}

/\*\*

\* Creates a no decay scoring function (used by top archive).

\* Otherwise same as default constructor.

\* @param nodecay If no decay is set to true. Alpha is set to 0.0.

\*/

public RetweetBasedTopTweetsScoringFunction(ImmutableSchemaInterface schema, boolean nodecay) {

this(schema, DEFAULT\_RECENCY\_SCORE\_FRACTION,

nodecay ? 0.0 : DEFAULT\_SIGMOID\_APLHA,

DEFAULT\_CUT\_OFF\_SCORE,

DEFAULT\_RECENCY\_CENTER\_MINUTES);

}

public RetweetBasedTopTweetsScoringFunction(ImmutableSchemaInterface schema,

double recencyScoreFraction, double sigmoidAlpha,

double cutOffScore, int recencyCenterMinutes) {

super(schema);

this.recencyScoreFraction = recencyScoreFraction;

this.sigmoidAlpha = sigmoidAlpha;

this.cutOffScore = cutOffScore;

this.recencyCenterMinutes = recencyCenterMinutes;

this.maxRecency = computeSigmoid(0);

this.currentTimeSeconds = (int) (System.currentTimeMillis() / 1000);

}

@Override

protected float score(float luceneQueryScore) throws IOException {

// Reset the data for each tweet!!!

metadata = null;

if (documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG)

|| (documentFeatures.getFeatureValue(EarlybirdFieldConstant.USER\_REPUTATION)

< MIN\_USER\_REPUTATION)) {

score = RETWEET\_BASED\_TOP\_TWEETS\_LOWEST\_SCORE;

} else {

// Note that here we want the post log2 value, as the MAX\_RETWEET\_COUNT was actually

// set up for that.

retweetCount = MutableFeatureNormalizers.BYTE\_NORMALIZER.unnormAndLog2(

(byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.RETWEET\_COUNT));

final double recencyScore = computeTopTweetRecencyScore();

score = (retweetCount / MAX\_RETWEET\_COUNT) \* (1 - recencyScoreFraction)

+ recencyScoreFraction \* recencyScore;

if (score < this.cutOffScore) {

score = RETWEET\_BASED\_TOP\_TWEETS\_LOWEST\_SCORE;

}

}

return (float) score;

}

private double computeSigmoid(double x) {

return 1.0f / (1.0f + Math.exp(sigmoidAlpha \* (x - recencyCenterMinutes)));

}

private double computeTopTweetRecencyScore() {

double diffMinutes =

Math.max(0, currentTimeSeconds - timeMapper.getTime(getCurrentDocID())) / 60.0;

return computeSigmoid(diffMinutes) / maxRecency;

}

@Override

protected Explanation doExplain(float luceneScore) {

return null;

}

@Override

public ThriftSearchResultMetadata getResultMetadata(ThriftSearchResultMetadataOptions options) {

if (metadata == null) {

metadata = new ThriftSearchResultMetadata()

.setResultType(ThriftSearchResultType.POPULAR)

.setPenguinVersion(EarlybirdConfig.getPenguinVersionByte());

metadata.setRetweetCount((int) retweetCount);

metadata.setScore(score);

}

return metadata;

}

@Override

public void updateRelevanceStats(ThriftSearchResultsRelevanceStats relevanceStats) {

}

}