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

import java.io.IOException;

import java.util.List;

import com.google.common.collect.Lists;

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.userupdates.UserTable;

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

import com.twitter.search.earlybird.search.relevance.LinearScoringData;

import com.twitter.search.earlybird.search.relevance.LinearScoringParams;

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

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

/\*\*

\* Scoring function that uses the weights and boosts provided in the scoring parameters from the

\* request.

\*/

public class LinearScoringFunction extends FeatureBasedScoringFunction {

private static final double BASE\_SCORE = 0.0001;

public LinearScoringFunction(

ImmutableSchemaInterface schema,

ThriftSearchQuery searchQuery,

AntiGamingFilter antiGamingFilter,

ThriftSearchResultType searchResultType,

UserTable userTable) throws IOException {

super("LinearScoringFunction", schema, searchQuery, antiGamingFilter, searchResultType,

userTable);

}

@Override

protected double computeScore(LinearScoringData data, boolean forExplanation) throws IOException {

double score = BASE\_SCORE;

data.luceneContrib = params.useLuceneScoreAsBoost

? 0.0 : params.luceneWeight \* data.luceneScore;

data.reputationContrib = params.reputationWeight \* data.userRep;

data.textScoreContrib = params.textScoreWeight \* data.textScore;

data.parusContrib = params.parusWeight \* data.parusScore;

// contributions from engagement counters. Note that we have "true" argument for all getters,

// which means all values will get scaled down for scoring, they were unbounded in raw form.

data.retweetContrib = params.retweetWeight \* data.retweetCountPostLog2;

data.favContrib = params.favWeight \* data.favCountPostLog2;

data.replyContrib = params.replyWeight \* data.replyCountPostLog2;

data.embedsImpressionContrib =

params.embedsImpressionWeight \* data.getEmbedsImpressionCount(true);

data.embedsUrlContrib =

params.embedsUrlWeight \* data.getEmbedsUrlCount(true);

data.videoViewContrib =

params.videoViewWeight \* data.getVideoViewCount(true);

data.quotedContrib =

params.quotedCountWeight \* data.quotedCount;

for (int i = 0; i < LinearScoringData.MAX\_OFFLINE\_EXPERIMENTAL\_FIELDS; i++) {

data.offlineExpFeatureContributions[i] =

params.rankingOfflineExpWeights[i] \* data.offlineExpFeatureValues[i];

}

data.hasUrlContrib = params.urlWeight \* (data.hasUrl ? 1.0 : 0.0);

data.isReplyContrib = params.isReplyWeight \* (data.isReply ? 1.0 : 0.0);

data.isFollowRetweetContrib =

params.followRetweetWeight \* (data.isRetweet && data.isFollow ? 1.0 : 0.0);

data.isTrustedRetweetContrib =

params.trustedRetweetWeight \* (data.isRetweet && data.isTrusted ? 1.0 : 0.0);

double replyCountOriginal = getUnscaledReplyCountFeatureValue();

data.multipleReplyContrib = params.multipleReplyWeight

\* (replyCountOriginal < params.multipleReplyMinVal ? 0.0 : replyCountOriginal);

// We directly the query specific score as the contribution below as it doesn't need a weight

// for contribution computation.

score += data.luceneContrib

+ data.reputationContrib

+ data.textScoreContrib

+ data.replyContrib

+ data.multipleReplyContrib

+ data.retweetContrib

+ data.favContrib

+ data.parusContrib

+ data.embedsImpressionContrib

+ data.embedsUrlContrib

+ data.videoViewContrib

+ data.quotedContrib

+ data.hasUrlContrib

+ data.isReplyContrib

+ data.isFollowRetweetContrib

+ data.isTrustedRetweetContrib

+ data.querySpecificScore

+ data.authorSpecificScore;

for (int i = 0; i < LinearScoringData.MAX\_OFFLINE\_EXPERIMENTAL\_FIELDS; i++) {

score += data.offlineExpFeatureContributions[i];

}

return score;

}

/\*\*

\* Generates the explanation for the linear score.

\*/

@Override

protected void generateExplanationForScoring(

LinearScoringData scoringData, boolean isHit, List<Explanation> details) throws IOException {

// 1. Linear components

final List<Explanation> linearDetails = Lists.newArrayList();

addLinearElementExplanation(

linearDetails, "[LuceneQueryScore]",

params.luceneWeight, scoringData.luceneScore, scoringData.luceneContrib);

if (scoringData.hasCard) {

if (scoringData.cardAuthorMatchBoostApplied) {

linearDetails.add(Explanation.match(

(float) params.cardAuthorMatchBoosts[scoringData.cardType],

"[x] card author match boost"));

}

if (scoringData.cardDescriptionMatchBoostApplied) {

linearDetails.add(Explanation.match(

(float) params.cardDescriptionMatchBoosts[scoringData.cardType],

"[x] card description match boost"));

}

if (scoringData.cardDomainMatchBoostApplied) {

linearDetails.add(Explanation.match(

(float) params.cardDomainMatchBoosts[scoringData.cardType],

"[x] card domain match boost"));

}

if (scoringData.cardTitleMatchBoostApplied) {

linearDetails.add(Explanation.match(

(float) params.cardTitleMatchBoosts[scoringData.cardType],

"[x] card title match boost"));

}

}

addLinearElementExplanation(

linearDetails, "reputation",

params.reputationWeight, scoringData.userRep, scoringData.reputationContrib);

addLinearElementExplanation(

linearDetails, "text score",

params.textScoreWeight, scoringData.textScore, scoringData.textScoreContrib);

addLinearElementExplanation(

linearDetails, "reply count (log2)",

params.replyWeight, scoringData.replyCountPostLog2, scoringData.replyContrib);

addLinearElementExplanation(

linearDetails, "multi reply",

params.multipleReplyWeight,

getUnscaledReplyCountFeatureValue() > params.multipleReplyMinVal ? 1 : 0,

scoringData.multipleReplyContrib);

addLinearElementExplanation(

linearDetails, "retweet count (log2)",

params.retweetWeight, scoringData.retweetCountPostLog2, scoringData.retweetContrib);

addLinearElementExplanation(

linearDetails, "fav count (log2)",

params.favWeight, scoringData.favCountPostLog2, scoringData.favContrib);

addLinearElementExplanation(

linearDetails, "parus score",

params.parusWeight, scoringData.parusScore, scoringData.parusContrib);

for (int i = 0; i < LinearScoringData.MAX\_OFFLINE\_EXPERIMENTAL\_FIELDS; i++) {

if (params.rankingOfflineExpWeights[i] != LinearScoringParams.DEFAULT\_FEATURE\_WEIGHT) {

addLinearElementExplanation(linearDetails,

"ranking exp score offline experimental #" + i,

params.rankingOfflineExpWeights[i], scoringData.offlineExpFeatureValues[i],

scoringData.offlineExpFeatureContributions[i]);

}

}

addLinearElementExplanation(linearDetails,

"embedded tweet impression count",

params.embedsImpressionWeight, scoringData.getEmbedsImpressionCount(false),

scoringData.embedsImpressionContrib);

addLinearElementExplanation(linearDetails,

"embedded tweet url count",

params.embedsUrlWeight, scoringData.getEmbedsUrlCount(false),

scoringData.embedsUrlContrib);

addLinearElementExplanation(linearDetails,

"video view count",

params.videoViewWeight, scoringData.getVideoViewCount(false),

scoringData.videoViewContrib);

addLinearElementExplanation(linearDetails,

"quoted count",

params.quotedCountWeight, scoringData.quotedCount, scoringData.quotedContrib);

addLinearElementExplanation(

linearDetails, "has url", params.urlWeight, scoringData.hasUrl ? 1.0 : 0.0,

scoringData.hasUrlContrib);

addLinearElementExplanation(

linearDetails, "is reply", params.isReplyWeight,

scoringData.isReply ? 1.0 : 0.0, scoringData.isReplyContrib);

addLinearElementExplanation(

linearDetails, "is follow retweet", params.followRetweetWeight,

scoringData.isRetweet && scoringData.isFollow ? 1.0 : 0.0,

scoringData.isFollowRetweetContrib);

addLinearElementExplanation(

linearDetails, "is trusted retweet", params.trustedRetweetWeight,

scoringData.isRetweet && scoringData.isTrusted ? 1.0 : 0.0,

scoringData.isTrustedRetweetContrib);

if (scoringData.querySpecificScore != 0.0) {

linearDetails.add(Explanation.match((float) scoringData.querySpecificScore,

"[+] query specific score adjustment"));

}

if (scoringData.authorSpecificScore != 0.0) {

linearDetails.add(Explanation.match((float) scoringData.authorSpecificScore,

"[+] author specific score adjustment"));

}

Explanation linearCombo = isHit

? Explanation.match((float) scoringData.scoreBeforeBoost,

"(MATCH) Linear components, sum of:", linearDetails)

: Explanation.noMatch("Linear components, sum of:", linearDetails);

details.add(linearCombo);

}

private void addLinearElementExplanation(List<Explanation> explanation,

String name,

double weight,

double componentValue,

double contrib) {

if (contrib == 0.0) {

return;

}

explanation.add(

Explanation.match((float) contrib,

String.format("[+] %s=%.3f weight=%.3f", name, componentValue, weight)));

}

private double getUnscaledReplyCountFeatureValue() throws IOException {

byte featureValue = (byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.REPLY\_COUNT);

return MutableFeatureNormalizers.BYTE\_NORMALIZER.unnormLowerBound(featureValue);

}

}