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

import java.io.IOException;

import java.util.EnumSet;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableSet;

import com.google.common.collect.Iterables;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.google.common.primitives.Ints;

import com.google.common.primitives.Longs;

import org.apache.lucene.search.Explanation;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common\_internal.bloomfilter.BloomFilter;

import com.twitter.search.common.constants.SearchCardType;

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

import com.twitter.search.common.database.DatabaseConfig;

import com.twitter.search.common.features.ExternalTweetFeature;

import com.twitter.search.common.features.FeatureHandler;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchemaEntry;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureType;

import com.twitter.search.common.features.thrift.ThriftSearchResultFeatures;

import com.twitter.search.common.query.QueryCommonFieldHitsVisitor;

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

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

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

import com.twitter.search.common.relevance.text.VisibleTokenRatioNormalizer;

import com.twitter.search.common.results.thriftjava.FieldHitList;

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

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

import com.twitter.search.common.util.LongIntConverter;

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

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

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.LinearScoringData.SkipReason;

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

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

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

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;

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

/\*\*

\* Base class for scoring functions that rely on the extracted features stored in LinearScoringData.

\*

\* Extensions of this class must implement 2 methods:

\*

\* - computeScore

\* - generateExplanationForScoring

\*

\* They are called for scoring and generating the debug information of the document that it's

\* currently being evaluated. The field 'data' holds the features of the document.

\*/

public abstract class FeatureBasedScoringFunction extends ScoringFunction {

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

// A multiplier that's applied to all scores to avoid scores too low.

public static final float SCORE\_ADJUSTER = 100.0f;

private static final VisibleTokenRatioNormalizer VISIBLE\_TOKEN\_RATIO\_NORMALIZER =

VisibleTokenRatioNormalizer.createInstance();

// Allow default values only for numeric types.

private static final Set<ThriftSearchFeatureType> ALLOWED\_TYPES\_FOR\_DEFAULT\_FEATURE\_VALUES =

EnumSet.of(ThriftSearchFeatureType.INT32\_VALUE,

ThriftSearchFeatureType.LONG\_VALUE,

ThriftSearchFeatureType.DOUBLE\_VALUE);

private static final Set<Integer> NUMERIC\_FEATURES\_FOR\_WHICH\_DEFAULTS\_SHOULD\_NOT\_BE\_SET =

ImmutableSet.of(EarlybirdFieldConstant.TWEET\_SIGNATURE.getFieldId(),

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_LEAST\_SIGNIFICANT\_INT.getFieldId(),

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_MOST\_SIGNIFICANT\_INT.getFieldId());

// Name of the scoring function. Used for generating explanations.

private final String functionName;

private final BloomFilter trustedFilter;

private final BloomFilter followFilter;

// Current timestamp in seconds. Overridable by unit test or by timestamp set in search query.

private int now;

private final AntiGamingFilter antiGamingFilter;

@Nullable

private final AgeDecay ageDecay;

protected final LinearScoringParams params; // Parameters and query-dependent values.

// In order for the API calls to retrieve the correct `LinearScoringData`

// for the passed `docId`, we need to maintain a map of `docId` -> `LinearScoringData`

// NOTE: THIS CAN ONLY BE REFERENCED AT HIT COLLECTION TIME, SINCE DOC IDS ARE NOT UNIQUE

// ACROSS SEGMENTS. IT'S NOT USABLE DURING BATCH SCORING.

private final Map<Integer, LinearScoringData> docIdToScoringData;

private final ThriftSearchResultType searchResultType;

private final UserTable userTable;

@VisibleForTesting

void setNow(int fakeNow) {

now = fakeNow;

}

public FeatureBasedScoringFunction(

String functionName,

ImmutableSchemaInterface schema,

ThriftSearchQuery searchQuery,

AntiGamingFilter antiGamingFilter,

ThriftSearchResultType searchResultType,

UserTable userTable) throws IOException {

super(schema);

this.functionName = functionName;

this.searchResultType = searchResultType;

this.userTable = userTable;

Preconditions.checkNotNull(searchQuery.getRelevanceOptions());

ThriftRankingParams rankingParams = searchQuery.getRelevanceOptions().getRankingParams();

Preconditions.checkNotNull(rankingParams);

params = new LinearScoringParams(searchQuery, rankingParams);

docIdToScoringData = new HashMap<>();

long timestamp = searchQuery.isSetTimestampMsecs() && searchQuery.getTimestampMsecs() > 0

? searchQuery.getTimestampMsecs() : System.currentTimeMillis();

now = Ints.checkedCast(TimeUnit.MILLISECONDS.toSeconds(timestamp));

this.antiGamingFilter = antiGamingFilter;

this.ageDecay = params.useAgeDecay

? new AgeDecay(params.ageDecayBase, params.ageDecayHalflife, params.ageDecaySlope)

: null;

if (searchQuery.isSetTrustedFilter()) {

trustedFilter = new BloomFilter(searchQuery.getTrustedFilter());

} else {

trustedFilter = null;

}

if (searchQuery.isSetDirectFollowFilter()) {

followFilter = new BloomFilter(searchQuery.getDirectFollowFilter());

} else {

followFilter = null;

}

}

@VisibleForTesting

final LinearScoringParams getScoringParams() {

return params;

}

/\*\*

\* Returns the LinearScoringData instance associated with the current doc ID. If it doesn't exist,

\* an empty LinearScoringData is created.

\*/

@Override

public LinearScoringData getScoringDataForCurrentDocument() {

LinearScoringData data = docIdToScoringData.get(getCurrentDocID());

if (data == null) {

data = new LinearScoringData();

docIdToScoringData.put(getCurrentDocID(), data);

}

return data;

}

@Override

public void setDebugMode(int debugMode) {

super.setDebugMode(debugMode);

}

/\*\*

\* Normal the lucene score, which was unbounded, to a range of [1.0, maxLuceneScoreBoost].

\* The normalized value increases almost linearly in the lucene score range 2.0 ~ 7.0, where

\* most queries fall in. For rare long tail queries, like some hashtags, they have high idf and

\* thus high lucene score, the normalized value won't have much difference between tweets.

\* The normalization function is:

\* ls = luceneScore

\* norm = min(max, 1 + (max - 1.0) / 2.4 \* ln(1 + ls)

\*/

static float normalizeLuceneScore(float luceneScore, float maxBoost) {

return (float) Math.min(maxBoost, 1.0 + (maxBoost - 1.0) / 2.4 \* Math.log1p(luceneScore));

}

@Override

protected float score(float luceneQueryScore) throws IOException {

return scoreInternal(luceneQueryScore, null);

}

protected LinearScoringData updateLinearScoringData(float luceneQueryScore) throws IOException {

// Reset the data for each tweet!!!

LinearScoringData data = new LinearScoringData();

docIdToScoringData.put(getCurrentDocID(), data);

// Set proper version for engagement counters for this request.

data.skipReason = SkipReason.NOT\_SKIPPED;

data.luceneScore = luceneQueryScore;

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

if (antiGamingFilter != null && !antiGamingFilter.accept(getCurrentDocID())) {

data.skipReason = SkipReason.ANTIGAMING;

return data;

}

data.textScore = (byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.TEXT\_SCORE);

data.tokenAt140DividedByNumTokensBucket = VISIBLE\_TOKEN\_RATIO\_NORMALIZER.denormalize(

(byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.VISIBLE\_TOKEN\_RATIO));

data.fromUserId = documentFeatures.getFeatureValue(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF);

data.isFollow = followFilter != null

&& followFilter.contains(Longs.toByteArray(data.fromUserId));

data.isTrusted = trustedFilter != null

&& trustedFilter.contains(Longs.toByteArray(data.fromUserId));

data.isFromVerifiedAccount = documentFeatures.isFlagSet(

EarlybirdFieldConstant.FROM\_VERIFIED\_ACCOUNT\_FLAG);

data.isFromBlueVerifiedAccount = documentFeatures.isFlagSet(

EarlybirdFieldConstant.FROM\_BLUE\_VERIFIED\_ACCOUNT\_FLAG);

data.isSelfTweet = data.fromUserId == params.searcherId;

// v1 engagement counters, note that the first three values are post-log2 version

// of the original unnormalized values.

data.retweetCountPostLog2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.RETWEET\_COUNT);

data.replyCountPostLog2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.REPLY\_COUNT);

data.favCountPostLog2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.FAVORITE\_COUNT);

data.embedsImpressionCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EMBEDS\_IMPRESSION\_COUNT);

data.embedsUrlCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EMBEDS\_URL\_COUNT);

data.videoViewCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.VIDEO\_VIEW\_COUNT);

// v2 engagement counters

data.retweetCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.RETWEET\_COUNT\_V2);

data.replyCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.REPLY\_COUNT\_V2);

data.favCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.FAVORITE\_COUNT\_V2);

// other v2 engagement counters

data.embedsImpressionCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EMBEDS\_IMPRESSION\_COUNT\_V2);

data.embedsUrlCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EMBEDS\_URL\_COUNT\_V2);

data.videoViewCountV2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.VIDEO\_VIEW\_COUNT\_V2);

// pure v2 engagement counters without v1 counterpart

data.quotedCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.QUOTE\_COUNT);

data.weightedRetweetCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.WEIGHTED\_RETWEET\_COUNT);

data.weightedReplyCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.WEIGHTED\_REPLY\_COUNT);

data.weightedFavCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.WEIGHTED\_FAVORITE\_COUNT);

data.weightedQuoteCount = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.WEIGHTED\_QUOTE\_COUNT);

Double querySpecificScoreAdjustment = params.querySpecificScoreAdjustments == null ? null

: params.querySpecificScoreAdjustments.get(tweetIDMapper.getTweetID(getCurrentDocID()));

data.querySpecificScore =

querySpecificScoreAdjustment == null ? 0.0 : querySpecificScoreAdjustment;

data.authorSpecificScore = params.authorSpecificScoreAdjustments == null

? 0.0

: params.authorSpecificScoreAdjustments.getOrDefault(data.fromUserId, 0.0);

// respect social filter type

if (params.socialFilterType != null && !data.isSelfTweet) {

if ((params.socialFilterType == ThriftSocialFilterType.ALL

&& !data.isFollow && !data.isTrusted)

|| (params.socialFilterType == ThriftSocialFilterType.TRUSTED && !data.isTrusted)

|| (params.socialFilterType == ThriftSocialFilterType.FOLLOWS && !data.isFollow)) {

// we can skip this hit as we only want social results in this mode.

data.skipReason = SkipReason.SOCIAL\_FILTER;

return data;

}

}

// 1. first apply all the filters to only non-follow tweets and non-verified accounts,

// but be tender to sentinel values

// unless you specifically asked to apply filters regardless

if (params.applyFiltersAlways

|| (!data.isSelfTweet && !data.isFollow && !data.isFromVerifiedAccount

&& !data.isFromBlueVerifiedAccount)) {

if (data.userRep < params.reputationMinVal

// don't filter unset userreps, we give them the benefit of doubt and let it

// continue to scoring. userrep is unset when either user just signed up or

// during ingestion time we had trouble getting userrep from reputation service.

&& data.userRep != RelevanceSignalConstants.UNSET\_REPUTATION\_SENTINEL) {

data.skipReason = SkipReason.LOW\_REPUTATION;

return data;

} else if (data.textScore < params.textScoreMinVal

// don't filter unset text scores, use goodwill value

&& data.textScore != RelevanceSignalConstants.UNSET\_TEXT\_SCORE\_SENTINEL) {

data.skipReason = SkipReason.LOW\_TEXT\_SCORE;

return data;

} else if (data.retweetCountPostLog2 != LinearScoringData.UNSET\_SIGNAL\_VALUE

&& data.retweetCountPostLog2 < params.retweetMinVal) {

data.skipReason = SkipReason.LOW\_RETWEET\_COUNT;

return data;

} else if (data.favCountPostLog2 != LinearScoringData.UNSET\_SIGNAL\_VALUE

&& data.favCountPostLog2 < params.favMinVal) {

data.skipReason = SkipReason.LOW\_FAV\_COUNT;

return data;

}

}

// if sentinel value is set, assume goodwill score and let scoring continue.

if (data.textScore == RelevanceSignalConstants.UNSET\_TEXT\_SCORE\_SENTINEL) {

data.textScore = RelevanceSignalConstants.GOODWILL\_TEXT\_SCORE;

}

if (data.userRep == RelevanceSignalConstants.UNSET\_REPUTATION\_SENTINEL) {

data.userRep = RelevanceSignalConstants.GOODWILL\_REPUTATION;

}

data.tweetAgeInSeconds = now - timeMapper.getTime(getCurrentDocID());

if (data.tweetAgeInSeconds < 0) {

data.tweetAgeInSeconds = 0; // Age cannot be negative

}

// The PARUS\_SCORE feature should be read as is.

data.parusScore = documentFeatures.getFeatureValue(EarlybirdFieldConstant.PARUS\_SCORE);

data.isNullcast = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_NULLCAST\_FLAG);

data.hasUrl = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_LINK\_FLAG);

data.hasImageUrl = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_IMAGE\_URL\_FLAG);

data.hasVideoUrl = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_VIDEO\_URL\_FLAG);

data.hasNewsUrl = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_NEWS\_URL\_FLAG);

data.isReply = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_REPLY\_FLAG);

data.isRetweet = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_RETWEET\_FLAG);

data.isOffensive = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG);

data.hasTrend = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_TREND\_FLAG);

data.hasMultipleHashtagsOrTrends =

documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_MULTIPLE\_HASHTAGS\_OR\_TRENDS\_FLAG);

data.isUserSpam = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_USER\_SPAM\_FLAG);

data.isUserNSFW = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_USER\_NSFW\_FLAG)

|| userTable.isSet(data.fromUserId, UserTable.NSFW\_BIT);

data.isUserAntiSocial =

userTable.isSet(data.fromUserId, UserTable.ANTISOCIAL\_BIT);

data.isUserBot = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_USER\_BOT\_FLAG);

data.hasCard = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_CARD\_FLAG);

data.cardType = SearchCardType.UNKNOWN.getByteValue();

if (data.hasCard) {

data.cardType =

(byte) documentFeatures.getFeatureValue(EarlybirdFieldConstant.CARD\_TYPE\_CSF\_FIELD);

}

data.hasVisibleLink = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_VISIBLE\_LINK\_FLAG);

data.hasConsumerVideo =

documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_CONSUMER\_VIDEO\_FLAG);

data.hasProVideo = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_PRO\_VIDEO\_FLAG);

data.hasVine = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_VINE\_FLAG);

data.hasPeriscope = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_PERISCOPE\_FLAG);

data.hasNativeImage = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_NATIVE\_IMAGE\_FLAG);

data.hasQuote = documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_QUOTE\_FLAG);

data.isComposerSourceCamera =

documentFeatures.isFlagSet(EarlybirdFieldConstant.COMPOSER\_SOURCE\_IS\_CAMERA\_FLAG);

// Only read the shared status if the isRetweet or isReply bit is true (minor optimization).

if (data.isRetweet || (params.getInReplyToStatusId && data.isReply)) {

data.sharedStatusId =

documentFeatures.getFeatureValue(EarlybirdFieldConstant.SHARED\_STATUS\_ID\_CSF);

}

// Only read the reference tweet author ID if the isRetweet or isReply bit

// is true (minor optimization).

if (data.isRetweet || data.isReply) {

// the REFERENCE\_AUTHOR\_ID\_CSF stores the source tweet author id for all retweets

long referenceAuthorId =

documentFeatures.getFeatureValue(EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_CSF);

if (referenceAuthorId > 0) {

data.referenceAuthorId = referenceAuthorId;

} else {

// we also store the reference author id for retweets, directed at tweets, and self threaded

// tweets separately on Realtime/Protected Earlybirds. This data will be moved to the

// REFERENCE\_AUTHOR\_ID\_CSF and these fields will be deprecated in SEARCH-34958.

referenceAuthorId = LongIntConverter.convertTwoIntToOneLong(

(int) documentFeatures.getFeatureValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_MOST\_SIGNIFICANT\_INT),

(int) documentFeatures.getFeatureValue(

EarlybirdFieldConstant.REFERENCE\_AUTHOR\_ID\_LEAST\_SIGNIFICANT\_INT));

if (referenceAuthorId > 0) {

data.referenceAuthorId = referenceAuthorId;

}

}

}

// Convert language to a thrift language and then back to an int in order to

// ensure a value compatible with our current ThriftLanguage definition.

ThriftLanguage tweetLang = ThriftLanguageUtil.safeFindByValue(

(int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.LANGUAGE));

data.tweetLangId = tweetLang.getValue();

// Set the language-related features here so that they can be later used in promotion/demotion

// and also be transferred to ThriftSearchResultMetadata

data.userLangMult = computeUserLangMultiplier(data, params);

data.hasDifferentLang = params.uiLangId != ThriftLanguage.UNKNOWN.getValue()

&& params.uiLangId != data.tweetLangId;

data.hasEnglishTweetAndDifferentUILang = data.hasDifferentLang

&& data.tweetLangId == ThriftLanguage.ENGLISH.getValue();

data.hasEnglishUIAndDifferentTweetLang = data.hasDifferentLang

&& params.uiLangId == ThriftLanguage.ENGLISH.getValue();

// Exposed all these features for the clients.

data.isSensitiveContent =

documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_SENSITIVE\_CONTENT);

data.hasMultipleMediaFlag =

documentFeatures.isFlagSet(EarlybirdFieldConstant.HAS\_MULTIPLE\_MEDIA\_FLAG);

data.profileIsEggFlag = documentFeatures.isFlagSet(EarlybirdFieldConstant.PROFILE\_IS\_EGG\_FLAG);

data.isUserNewFlag = documentFeatures.isFlagSet(EarlybirdFieldConstant.IS\_USER\_NEW\_FLAG);

data.numMentions = (int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.NUM\_MENTIONS);

data.numHashtags = (int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.NUM\_HASHTAGS);

data.linkLanguage =

(int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.LINK\_LANGUAGE);

data.prevUserTweetEngagement =

(int) documentFeatures.getFeatureValue(EarlybirdFieldConstant.PREV\_USER\_TWEET\_ENGAGEMENT);

// health model scores by HML

data.toxicityScore = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.TOXICITY\_SCORE);

data.pBlockScore = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.PBLOCK\_SCORE);

data.pSpammyTweetScore = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.P\_SPAMMY\_TWEET\_SCORE);

data.pReportedTweetScore = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.P\_REPORTED\_TWEET\_SCORE);

data.spammyTweetContentScore = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.SPAMMY\_TWEET\_CONTENT\_SCORE

);

data.experimentalHealthModelScore1 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EXPERIMENTAL\_HEALTH\_MODEL\_SCORE\_1);

data.experimentalHealthModelScore2 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EXPERIMENTAL\_HEALTH\_MODEL\_SCORE\_2);

data.experimentalHealthModelScore3 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EXPERIMENTAL\_HEALTH\_MODEL\_SCORE\_3);

data.experimentalHealthModelScore4 = documentFeatures.getUnnormalizedFeatureValue(

EarlybirdFieldConstant.EXPERIMENTAL\_HEALTH\_MODEL\_SCORE\_4);

return data;

}

protected float scoreInternal(

float luceneQueryScore, ExplanationWrapper explanation) throws IOException {

LinearScoringData data = updateLinearScoringData(luceneQueryScore);

if (data.skipReason != null && data.skipReason != SkipReason.NOT\_SKIPPED) {

return finalizeScore(data, explanation, SKIP\_HIT);

}

double score = computeScore(data, explanation != null);

return postScoreComputation(data, score, true, explanation);

}

protected float postScoreComputation(

LinearScoringData data,

double score,

boolean boostScoreWithHitAttribution,

ExplanationWrapper explanation) throws IOException {

double modifiedScore = score;

data.scoreBeforeBoost = modifiedScore;

if (params.applyBoosts) {

modifiedScore =

applyBoosts(data, modifiedScore, boostScoreWithHitAttribution, explanation != null);

}

// Final adjustment to avoid too-low scores.

modifiedScore \*= SCORE\_ADJUSTER;

data.scoreAfterBoost = modifiedScore;

// 3. final score filter

data.scoreFinal = modifiedScore;

if ((params.applyFiltersAlways || (!data.isSelfTweet && !data.isFollow))

&& modifiedScore < params.minScore) {

data.skipReason = SkipReason.LOW\_FINAL\_SCORE;

modifiedScore = SKIP\_HIT;

}

// clear field hits

this.fieldHitAttribution = null;

return finalizeScore(data, explanation, modifiedScore);

}

/\*\*

\* Applying promotion/demotion to the scores generated by feature-based scoring functions

\*

\* @param data Original LinearScoringData (to be modified with boosts here)

\* @param score Score generated by the feature-based scoring function

\* @param withHitAttribution Determines if hit attribution data should be included.

\* @param forExplanation Indicates if the score will be computed for generating the explanation.

\* @return Score after applying promotion/demotion

\*/

private double applyBoosts(

LinearScoringData data,

double score,

boolean withHitAttribution,

boolean forExplanation) {

double boostedScore = score;

if (params.useLuceneScoreAsBoost) {

data.normalizedLuceneScore = normalizeLuceneScore(

(float) data.luceneScore, (float) params.maxLuceneScoreBoost);

boostedScore \*= data.normalizedLuceneScore;

}

if (data.isOffensive) {

boostedScore \*= params.offensiveDamping;

}

if (data.isUserSpam && params.spamUserDamping != LinearScoringData.NO\_BOOST\_VALUE) {

data.spamUserDampApplied = true;

boostedScore \*= params.spamUserDamping;

}

if (data.isUserNSFW && params.nsfwUserDamping != LinearScoringData.NO\_BOOST\_VALUE) {

data.nsfwUserDampApplied = true;

boostedScore \*= params.nsfwUserDamping;

}

if (data.isUserBot && params.botUserDamping != LinearScoringData.NO\_BOOST\_VALUE) {

data.botUserDampApplied = true;

boostedScore \*= params.botUserDamping;

}

// cards

if (data.hasCard && params.hasCardBoosts[data.cardType] != LinearScoringData.NO\_BOOST\_VALUE) {

boostedScore \*= params.hasCardBoosts[data.cardType];

data.hasCardBoostApplied = true;

}

// trends

if (data.hasMultipleHashtagsOrTrends) {

boostedScore \*= params.multipleHashtagsOrTrendsDamping;

} else if (data.hasTrend) {

data.tweetHasTrendsBoostApplied = true;

boostedScore \*= params.tweetHasTrendBoost;

}

// Media/News url boosts.

if (data.hasImageUrl || data.hasVideoUrl) {

data.hasMedialUrlBoostApplied = true;

boostedScore \*= params.tweetHasMediaUrlBoost;

}

if (data.hasNewsUrl) {

data.hasNewsUrlBoostApplied = true;

boostedScore \*= params.tweetHasNewsUrlBoost;

}

if (data.isFromVerifiedAccount) {

data.tweetFromVerifiedAccountBoostApplied = true;

boostedScore \*= params.tweetFromVerifiedAccountBoost;

}

if (data.isFromBlueVerifiedAccount) {

data.tweetFromBlueVerifiedAccountBoostApplied = true;

boostedScore \*= params.tweetFromBlueVerifiedAccountBoost;

}

if (data.isFollow) {

// direct follow, so boost both replies and non-replies.

data.directFollowBoostApplied = true;

boostedScore \*= params.directFollowBoost;

} else if (data.isTrusted) {

// trusted circle

if (!data.isReply) {

// non-at-reply, in trusted network

data.trustedCircleBoostApplied = true;

boostedScore \*= params.trustedCircleBoost;

}

} else if (data.isReply) {

// at-reply out of my network

data.outOfNetworkReplyPenaltyApplied = true;

boostedScore -= params.outOfNetworkReplyPenalty;

}

if (data.isSelfTweet) {

data.selfTweetBoostApplied = true;

data.selfTweetMult = params.selfTweetBoost;

boostedScore \*= params.selfTweetBoost;

}

// Language Demotion

// User language based demotion

// The data.userLangMult is set in scoreInternal(), and this setting step is always before

// the applying boosts step

if (params.useUserLanguageInfo) {

boostedScore \*= data.userLangMult;

}

// UI language based demotion

if (params.uiLangId != ThriftLanguage.UNKNOWN.getValue()

&& params.uiLangId != data.tweetLangId) {

if (data.tweetLangId == ThriftLanguage.ENGLISH.getValue()) {

data.uiLangMult = params.langEnglishTweetDemote;

} else if (params.uiLangId == ThriftLanguage.ENGLISH.getValue()) {

data.uiLangMult = params.langEnglishUIDemote;

} else {

data.uiLangMult = params.langDefaultDemote;

}

} else {

data.uiLangMult = LinearScoringData.NO\_BOOST\_VALUE;

}

boostedScore \*= data.uiLangMult;

if (params.useAgeDecay) {

// shallow sigmoid with an inflection point at ageDecayHalflife

data.ageDecayMult = ageDecay.getAgeDecayMultiplier(data.tweetAgeInSeconds);

boostedScore \*= data.ageDecayMult;

}

// Hit Attribute Demotion

// Scoring is currently based on tokenized user name, text, and url in the tweet

// If hit attribute collection is enabled, we demote score based on these fields

if (hitAttributeHelper != null && params.enableHitDemotion) {

Map<Integer, List<String>> hitMap;

if (forExplanation && fieldHitAttribution != null) {

// if this scoring call is for generating an explanation,

// we'll use the fieldHitAttribution found in the search result's metadata because

// collectors are not called during the debug workflow

hitMap = Maps.transformValues(fieldHitAttribution.getHitMap(), FieldHitList::getHitFields);

} else if (withHitAttribution) {

hitMap = hitAttributeHelper.getHitAttribution(getCurrentDocID());

} else {

hitMap = Maps.newHashMap();

}

Set<String> uniqueFieldHits = ImmutableSet.copyOf(Iterables.concat(hitMap.values()));

data.hitFields.addAll(uniqueFieldHits);

// there should always be fields that are hit

// if there aren't, we assume this is a call from 'explain' in debug mode

// do not override hit attribute data if in debug mode

if (!uniqueFieldHits.isEmpty()) {

// demotions based strictly on field hits

if (uniqueFieldHits.size() == 1) {

if (uniqueFieldHits.contains(

EarlybirdFieldConstant.RESOLVED\_LINKS\_TEXT\_FIELD.getFieldName())) {

// if url was the only field that was hit, demote

data.hasUrlOnlyHitDemotionApplied = true;

boostedScore \*= params.urlOnlyHitDemotion;

} else if (uniqueFieldHits.contains(

EarlybirdFieldConstant.TOKENIZED\_FROM\_USER\_FIELD.getFieldName())) {

// if name was the only field that was hit, demote

data.hasNameOnlyHitDemotionApplied = true;

boostedScore \*= params.nameOnlyHitDemotion;

}

} else if (!uniqueFieldHits.contains(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName())

&& !uniqueFieldHits.contains(EarlybirdFieldConstant.MENTIONS\_FIELD.getFieldName())

&& !uniqueFieldHits.contains(EarlybirdFieldConstant.HASHTAGS\_FIELD.getFieldName())

&& !uniqueFieldHits.contains(EarlybirdFieldConstant.STOCKS\_FIELD.getFieldName())) {

// if text or special text was never hit, demote

data.hasNoTextHitDemotionApplied = true;

boostedScore \*= params.noTextHitDemotion;

} else if (uniqueFieldHits.size() == 2) {

// demotions based on field hit combinations

// want to demote if we only hit two of the fields (one being text)

// but with separate terms

Set<String> fieldIntersections = QueryCommonFieldHitsVisitor.findIntersection(

hitAttributeHelper.getNodeToRankMap(),

hitMap,

query);

if (fieldIntersections.isEmpty()) {

if (uniqueFieldHits.contains(

EarlybirdFieldConstant.TOKENIZED\_FROM\_USER\_FIELD.getFieldName())) {

// if name is hit but has no hits in common with text, demote

// want to demote cases where we hit part of the person's name

// and tweet text separately

data.hasSeparateTextAndNameHitDemotionApplied = true;

boostedScore \*= params.separateTextAndNameHitDemotion;

} else if (uniqueFieldHits.contains(

EarlybirdFieldConstant.RESOLVED\_LINKS\_TEXT\_FIELD.getFieldName())) {

// if url is hit but has no hits in common with text, demote

// want to demote cases where we hit a potential domain keyword

// and tweet text separately

data.hasSeparateTextAndUrlHitDemotionApplied = true;

boostedScore \*= params.separateTextAndUrlHitDemotion;

}

}

}

}

}

return boostedScore;

}

/\*\*

\* Compute the user language based demotion multiplier

\*/

private static double computeUserLangMultiplier(

LinearScoringData data, LinearScoringParams params) {

if (data.tweetLangId == params.uiLangId

&& data.tweetLangId != ThriftLanguage.UNKNOWN.getValue()) {

// Effectively the uiLang is considered a language that user knows with 1.0 confidence.

return LinearScoringData.NO\_BOOST\_VALUE;

}

if (params.userLangs[data.tweetLangId] > 0.0) {

return params.userLangs[data.tweetLangId];

}

return params.unknownLanguageBoost;

}

/\*\*

\* Computes the score of the document that it's currently being evaluated.

\*

\* The extracted features from the document are available in the field 'data'.

\*

\* @param data The LinearScoringData instance that will store the document features.

\* @param forExplanation Indicates if the score will be computed for generating the explanation.

\*/

protected abstract double computeScore(

LinearScoringData data, boolean forExplanation) throws IOException;

private float finalizeScore(

LinearScoringData scoringData,

ExplanationWrapper explanation,

double score) throws IOException {

scoringData.scoreReturned = score;

if (explanation != null) {

explanation.explanation = generateExplanation(scoringData);

}

return (float) score;

}

@Override

protected void initializeNextSegment(EarlybirdIndexSegmentAtomicReader reader)

throws IOException {

if (antiGamingFilter != null) {

antiGamingFilter.startSegment(reader);

}

}

/\*

\* Generate the scoring explanation for debug.

\*/

private Explanation generateExplanation(LinearScoringData scoringData) throws IOException {

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

details.add(Explanation.match(0.0f, "[PROPERTIES] "

+ scoringData.getPropertyExplanation()));

// 1. Filters

boolean isHit = scoringData.skipReason == SkipReason.NOT\_SKIPPED;

if (scoringData.skipReason == SkipReason.ANTIGAMING) {

details.add(Explanation.noMatch("SKIPPED for antigaming"));

}

if (scoringData.skipReason == SkipReason.LOW\_REPUTATION) {

details.add(Explanation.noMatch(

String.format("SKIPPED for low reputation: %.3f < %.3f",

scoringData.userRep, params.reputationMinVal)));

}

if (scoringData.skipReason == SkipReason.LOW\_TEXT\_SCORE) {

details.add(Explanation.noMatch(

String.format("SKIPPED for low text score: %.3f < %.3f",

scoringData.textScore, params.textScoreMinVal)));

}

if (scoringData.skipReason == SkipReason.LOW\_RETWEET\_COUNT) {

details.add(Explanation.noMatch(

String.format("SKIPPED for low retweet count: %.3f < %.3f",

scoringData.retweetCountPostLog2, params.retweetMinVal)));

}

if (scoringData.skipReason == SkipReason.LOW\_FAV\_COUNT) {

details.add(Explanation.noMatch(

String.format("SKIPPED for low fav count: %.3f < %.3f",

scoringData.favCountPostLog2, params.favMinVal)));

}

if (scoringData.skipReason == SkipReason.SOCIAL\_FILTER) {

details.add(Explanation.noMatch("SKIPPED for not in the right social circle"));

}

// 2. Explanation depending on the scoring type

generateExplanationForScoring(scoringData, isHit, details);

// 3. Explanation depending on boosts

if (params.applyBoosts) {

generateExplanationForBoosts(scoringData, isHit, details);

}

// 4. Final score filter.

if (scoringData.skipReason == SkipReason.LOW\_FINAL\_SCORE) {

details.add(Explanation.noMatch("SKIPPED for low final score: " + scoringData.scoreFinal));

isHit = false;

}

String hostAndSegment = String.format("%s host = %s segment = %s",

functionName, DatabaseConfig.getLocalHostname(), DatabaseConfig.getDatabase());

if (isHit) {

return Explanation.match((float) scoringData.scoreFinal, hostAndSegment, details);

} else {

return Explanation.noMatch(hostAndSegment, details);

}

}

/\*\*

\* Generates the explanation for the document that is currently being evaluated.

\*

\* Implementations of this method must use the 'details' parameter to collect its output.

\*

\* @param scoringData Scoring components for the document

\* @param isHit Indicates whether the document is not skipped

\* @param details Details of the explanation. Used to collect the output.

\*/

protected abstract void generateExplanationForScoring(

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

/\*\*

\* Generates the boosts part of the explanation for the document that is currently

\* being evaluated.

\*/

private void generateExplanationForBoosts(

LinearScoringData scoringData,

boolean isHit,

List<Explanation> details) {

List<Explanation> boostDetails = Lists.newArrayList();

boostDetails.add(Explanation.match((float) scoringData.scoreBeforeBoost, "Score before boost"));

// Lucene score boost

if (params.useLuceneScoreAsBoost) {

boostDetails.add(Explanation.match(

(float) scoringData.normalizedLuceneScore,

String.format("[x] Lucene score boost, luceneScore=%.3f",

scoringData.luceneScore)));

}

// card boost

if (scoringData.hasCardBoostApplied) {

boostDetails.add(Explanation.match((float) params.hasCardBoosts[scoringData.cardType],

"[x] card boost for type " + SearchCardType.cardTypeFromByteValue(scoringData.cardType)));

}

// Offensive

if (scoringData.isOffensive) {

boostDetails.add(Explanation.match((float) params.offensiveDamping, "[x] Offensive damping"));

} else {

boostDetails.add(Explanation.match(LinearScoringData.NO\_BOOST\_VALUE,

String.format("Not Offensive, damping=%.3f", params.offensiveDamping)));

}

// Spam

if (scoringData.spamUserDampApplied) {

boostDetails.add(Explanation.match((float) params.spamUserDamping, "[x] Spam"));

}

// NSFW

if (scoringData.nsfwUserDampApplied) {

boostDetails.add(Explanation.match((float) params.nsfwUserDamping, "[X] NSFW"));

}

// Bot

if (scoringData.botUserDampApplied) {

boostDetails.add(Explanation.match((float) params.botUserDamping, "[X] Bot"));

}

// Multiple hashtags or trends

if (scoringData.hasMultipleHashtagsOrTrends) {

boostDetails.add(Explanation.match((float) params.multipleHashtagsOrTrendsDamping,

"[x] Multiple hashtags or trends boost"));

} else {

boostDetails.add(Explanation.match(LinearScoringData.NO\_BOOST\_VALUE,

String.format("No multiple hashtags or trends, damping=%.3f",

params.multipleHashtagsOrTrendsDamping)));

}

if (scoringData.tweetHasTrendsBoostApplied) {

boostDetails.add(Explanation.match(

(float) params.tweetHasTrendBoost, "[x] Tweet has trend boost"));

}

if (scoringData.hasMedialUrlBoostApplied) {

boostDetails.add(Explanation.match(

(float) params.tweetHasMediaUrlBoost, "[x] Media url boost"));

}

if (scoringData.hasNewsUrlBoostApplied) {

boostDetails.add(Explanation.match(

(float) params.tweetHasNewsUrlBoost, "[x] News url boost"));

}

boostDetails.add(Explanation.match(0.0f, "[FIELDS HIT] " + scoringData.hitFields));

if (scoringData.hasNoTextHitDemotionApplied) {

boostDetails.add(Explanation.match(

(float) params.noTextHitDemotion, "[x] No text hit demotion"));

}

if (scoringData.hasUrlOnlyHitDemotionApplied) {

boostDetails.add(Explanation.match(

(float) params.urlOnlyHitDemotion, "[x] URL only hit demotion"));

}

if (scoringData.hasNameOnlyHitDemotionApplied) {

boostDetails.add(Explanation.match(

(float) params.nameOnlyHitDemotion, "[x] Name only hit demotion"));

}

if (scoringData.hasSeparateTextAndNameHitDemotionApplied) {

boostDetails.add(Explanation.match((float) params.separateTextAndNameHitDemotion,

"[x] Separate text/name demotion"));

}

if (scoringData.hasSeparateTextAndUrlHitDemotionApplied) {

boostDetails.add(Explanation.match((float) params.separateTextAndUrlHitDemotion,

"[x] Separate text/url demotion"));

}

if (scoringData.tweetFromVerifiedAccountBoostApplied) {

boostDetails.add(Explanation.match((float) params.tweetFromVerifiedAccountBoost,

"[x] Verified account boost"));

}

if (scoringData.tweetFromBlueVerifiedAccountBoostApplied) {

boostDetails.add(Explanation.match((float) params.tweetFromBlueVerifiedAccountBoost,

"[x] Blue-verified account boost"));

}

if (scoringData.selfTweetBoostApplied) {

boostDetails.add(Explanation.match((float) params.selfTweetBoost,

"[x] Self tweet boost"));

}

if (scoringData.skipReason == LinearScoringData.SkipReason.SOCIAL\_FILTER) {

boostDetails.add(Explanation.noMatch("SKIPPED for social filter"));

} else {

if (scoringData.directFollowBoostApplied) {

boostDetails.add(Explanation.match((float) params.directFollowBoost,

"[x] Direct follow boost"));

}

if (scoringData.trustedCircleBoostApplied) {

boostDetails.add(Explanation.match((float) params.trustedCircleBoost,

"[x] Trusted circle boost"));

}

if (scoringData.outOfNetworkReplyPenaltyApplied) {

boostDetails.add(Explanation.match((float) params.outOfNetworkReplyPenalty,

"[-] Out of network reply penalty"));

}

}

// Language demotions

String langDetails = String.format(

"tweetLang=[%s] uiLang=[%s]",

ThriftLanguageUtil.getLocaleOf(

ThriftLanguage.findByValue(scoringData.tweetLangId)).getLanguage(),

ThriftLanguageUtil.getLocaleOf(ThriftLanguage.findByValue(params.uiLangId)).getLanguage());

if (scoringData.uiLangMult == 1.0) {

boostDetails.add(Explanation.match(

LinearScoringData.NO\_BOOST\_VALUE, "No UI Language demotion: " + langDetails));

} else {

boostDetails.add(Explanation.match(

(float) scoringData.uiLangMult, "[x] UI LangMult: " + langDetails));

}

StringBuilder userLangDetails = new StringBuilder();

userLangDetails.append("userLang=[");

for (int i = 0; i < params.userLangs.length; i++) {

if (params.userLangs[i] > 0.0) {

String lang = ThriftLanguageUtil.getLocaleOf(ThriftLanguage.findByValue(i)).getLanguage();

userLangDetails.append(String.format("%s:%.3f,", lang, params.userLangs[i]));

}

}

userLangDetails.append("]");

if (!params.useUserLanguageInfo) {

boostDetails.add(Explanation.noMatch(

"No User Language Demotion: " + userLangDetails.toString()));

} else {

boostDetails.add(Explanation.match(

(float) scoringData.userLangMult,

"[x] User LangMult: " + userLangDetails.toString()));

}

// Age decay

String ageDecayDetails = String.format(

"age=%d seconds, slope=%.3f, base=%.1f, half-life=%.0f",

scoringData.tweetAgeInSeconds, params.ageDecaySlope,

params.ageDecayBase, params.ageDecayHalflife);

if (params.useAgeDecay) {

boostDetails.add(Explanation.match(

(float) scoringData.ageDecayMult, "[x] AgeDecay: " + ageDecayDetails));

} else {

boostDetails.add(Explanation.match(1.0f, "Age decay disabled: " + ageDecayDetails));

}

// Score adjuster

boostDetails.add(Explanation.match(SCORE\_ADJUSTER, "[x] score adjuster"));

Explanation boostCombo = isHit

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

"(MATCH) After Boosts and Demotions:", boostDetails)

: Explanation.noMatch("After Boosts and Demotions:", boostDetails);

details.add(boostCombo);

}

@Override

protected Explanation doExplain(float luceneQueryScore) throws IOException {

// Run the scorer again and get the explanation.

ExplanationWrapper explanation = new ExplanationWrapper();

scoreInternal(luceneQueryScore, explanation);

return explanation.explanation;

}

@Override

public void populateResultMetadataBasedOnScoringData(

ThriftSearchResultMetadataOptions options,

ThriftSearchResultMetadata metadata,

LinearScoringData data) throws IOException {

metadata.setResultType(searchResultType);

metadata.setScore(data.scoreReturned);

metadata.setFromUserId(data.fromUserId);

if (data.isTrusted) {

metadata.setIsTrusted(true);

}

if (data.isFollow) {

metadata.setIsFollow(true);

}

if (data.skipReason != SkipReason.NOT\_SKIPPED) {

metadata.setSkipped(true);

}

if ((data.isRetweet || (params.getInReplyToStatusId && data.isReply))

&& data.sharedStatusId != LinearScoringData.UNSET\_SIGNAL\_VALUE) {

metadata.setSharedStatusId(data.sharedStatusId);

}

if (data.hasCard) {

metadata.setCardType(data.cardType);

}

// Optional features. Note: other optional metadata is populated by

// AbstractRelevanceCollector, not the scoring function.

if (options.isGetLuceneScore()) {

metadata.setLuceneScore(data.luceneScore);

}

if (options.isGetReferencedTweetAuthorId()

&& data.referenceAuthorId != LinearScoringData.UNSET\_SIGNAL\_VALUE) {

metadata.setReferencedTweetAuthorId(data.referenceAuthorId);

}

if (options.isGetMediaBits()) {

metadata.setHasConsumerVideo(data.hasConsumerVideo);

metadata.setHasProVideo(data.hasProVideo);

metadata.setHasVine(data.hasVine);

metadata.setHasPeriscope(data.hasPeriscope);

boolean hasNativeVideo =

data.hasConsumerVideo || data.hasProVideo || data.hasVine || data.hasPeriscope;

metadata.setHasNativeVideo(hasNativeVideo);

metadata.setHasNativeImage(data.hasNativeImage);

}

metadata

.setIsOffensive(data.isOffensive)

.setIsReply(data.isReply)

.setIsRetweet(data.isRetweet)

.setHasLink(data.hasUrl)

.setHasTrend(data.hasTrend)

.setHasMultipleHashtagsOrTrends(data.hasMultipleHashtagsOrTrends)

.setRetweetCount((int) data.retweetCountPostLog2)

.setFavCount((int) data.favCountPostLog2)

.setReplyCount((int) data.replyCountPostLog2)

.setEmbedsImpressionCount((int) data.embedsImpressionCount)

.setEmbedsUrlCount((int) data.embedsUrlCount)

.setVideoViewCount((int) data.videoViewCount)

.setResultType(searchResultType)

.setFromVerifiedAccount(data.isFromVerifiedAccount)

.setIsUserSpam(data.isUserSpam)

.setIsUserNSFW(data.isUserNSFW)

.setIsUserBot(data.isUserBot)

.setHasImage(data.hasImageUrl)

.setHasVideo(data.hasVideoUrl)

.setHasNews(data.hasNewsUrl)

.setHasCard(data.hasCard)

.setHasVisibleLink(data.hasVisibleLink)

.setParusScore(data.parusScore)

.setTextScore(data.textScore)

.setUserRep(data.userRep)

.setTokenAt140DividedByNumTokensBucket(data.tokenAt140DividedByNumTokensBucket);

if (!metadata.isSetExtraMetadata()) {

metadata.setExtraMetadata(new ThriftSearchResultExtraMetadata());

}

ThriftSearchResultExtraMetadata extraMetadata = metadata.getExtraMetadata();

// Promotion/Demotion features

extraMetadata.setUserLangScore(data.userLangMult)

.setHasDifferentLang(data.hasDifferentLang)

.setHasEnglishTweetAndDifferentUILang(data.hasEnglishTweetAndDifferentUILang)

.setHasEnglishUIAndDifferentTweetLang(data.hasEnglishUIAndDifferentTweetLang)

.setHasQuote(data.hasQuote)

.setQuotedCount((int) data.quotedCount)

.setWeightedRetweetCount((int) data.weightedRetweetCount)

.setWeightedReplyCount((int) data.weightedReplyCount)

.setWeightedFavCount((int) data.weightedFavCount)

.setWeightedQuoteCount((int) data.weightedQuoteCount)

.setQuerySpecificScore(data.querySpecificScore)

.setAuthorSpecificScore(data.authorSpecificScore)

.setRetweetCountV2((int) data.retweetCountV2)

.setFavCountV2((int) data.favCountV2)

.setReplyCountV2((int) data.replyCountV2)

.setIsComposerSourceCamera(data.isComposerSourceCamera)

.setFromBlueVerifiedAccount(data.isFromBlueVerifiedAccount);

// Health model scores features

extraMetadata

.setToxicityScore(data.toxicityScore)

.setPBlockScore(data.pBlockScore)

.setPSpammyTweetScore(data.pSpammyTweetScore)

.setPReportedTweetScore(data.pReportedTweetScore)

.setSpammyTweetContentScore(data.spammyTweetContentScore)

.setExperimentalHealthModelScore1(data.experimentalHealthModelScore1)

.setExperimentalHealthModelScore2(data.experimentalHealthModelScore2)

.setExperimentalHealthModelScore3(data.experimentalHealthModelScore3)

.setExperimentalHealthModelScore4(data.experimentalHealthModelScore4);

// Return all extra features for clients to consume.

if (options.isGetAllFeatures()) {

extraMetadata.setIsSensitiveContent(data.isSensitiveContent)

.setHasMultipleMediaFlag(data.hasMultipleMediaFlag)

.setProfileIsEggFlag(data.profileIsEggFlag)

.setIsUserNewFlag(data.isUserNewFlag)

.setNumMentions(data.numMentions)

.setNumHashtags(data.numHashtags)

.setLinkLanguage(data.linkLanguage)

.setPrevUserTweetEngagement(data.prevUserTweetEngagement);

}

// Set features in new Feature Access API format, in the future this will be the only part

// needed in this method, we don't need to set any other metadata fields any more.

if (options.isReturnSearchResultFeatures()) {

// If the features are unset, and they were requested, then we can retrieve them. If they are

// already set, then we don't need to re-read the document features, and the reader

// is probably positioned over the wrong document so it will return incorrect results.

if (!extraMetadata.isSetFeatures()) {

// We ignore all features with default values when returning them in the response,

// because it saves a lot of network bandwidth.

ThriftSearchResultFeatures features = createFeaturesForDocument(data, true).getFeatures();

extraMetadata.setFeatures(features);

}

// The raw score may have changed since we created the features, so we should update it.

extraMetadata.getFeatures().getDoubleValues()

.put(ExternalTweetFeature.RAW\_EARLYBIRD\_SCORE.getId(), data.scoreFinal);

}

metadata

.setIsSelfTweet(data.isSelfTweet)

.setIsUserAntiSocial(data.isUserAntiSocial);

}

/\*\*

\* Create earlybird basic features and dervied features for current document.

\* @return a FeatureHandler object where you can keep adding extra feature values, or you can

\* call .getFeatures() on it to get a Thrift object to return.

\*/

protected FeatureHandler createFeaturesForDocument(

LinearScoringData data, boolean ignoreDefaultValues) throws IOException {

ThriftSearchResultFeatures features = documentFeatures.getSearchResultFeatures(getSchema());

if (!ignoreDefaultValues) {

setDefaultFeatureValues(features);

}

// add derived features

return new FeatureHandler(features, ignoreDefaultValues)

.addDouble(ExternalTweetFeature.LUCENE\_SCORE, data.luceneScore)

.addInt(ExternalTweetFeature.TWEET\_AGE\_IN\_SECS, data.tweetAgeInSeconds)

.addBoolean(ExternalTweetFeature.IS\_SELF\_TWEET, data.isSelfTweet)

.addBoolean(ExternalTweetFeature.IS\_FOLLOW\_RETWEET, data.isFollow && data.isRetweet)

.addBoolean(ExternalTweetFeature.IS\_TRUSTED\_RETWEET, data.isTrusted && data.isRetweet)

.addBoolean(ExternalTweetFeature.AUTHOR\_IS\_FOLLOW, data.isFollow)

.addBoolean(ExternalTweetFeature.AUTHOR\_IS\_TRUSTED, data.isTrusted)

.addBoolean(ExternalTweetFeature.AUTHOR\_IS\_ANTISOCIAL, data.isUserAntiSocial)

.addBoolean(ExternalTweetFeature.HAS\_DIFF\_LANG, data.hasDifferentLang)

.addBoolean(ExternalTweetFeature.HAS\_ENGLISH\_TWEET\_DIFF\_UI\_LANG,

data.hasEnglishTweetAndDifferentUILang)

.addBoolean(ExternalTweetFeature.HAS\_ENGLISH\_UI\_DIFF\_TWEET\_LANG,

data.hasEnglishUIAndDifferentTweetLang)

.addDouble(ExternalTweetFeature.SEARCHER\_LANG\_SCORE, data.userLangMult)

.addDouble(ExternalTweetFeature.QUERY\_SPECIFIC\_SCORE, data.querySpecificScore)

.addDouble(ExternalTweetFeature.AUTHOR\_SPECIFIC\_SCORE, data.authorSpecificScore);

}

/\*\*

\* Adds default values for most numeric features that do not have a value set yet in the given

\* ThriftSearchResultFeatures instance.

\*

\* This method is needed because some models do not work properly with missing features. Instead,

\* they expect all features to be present even if they are unset (their values are 0).

\*/

protected void setDefaultFeatureValues(ThriftSearchResultFeatures features) {

for (Map.Entry<Integer, ThriftSearchFeatureSchemaEntry> entry

: getSchema().getSearchFeatureSchema().getEntries().entrySet()) {

int featureId = entry.getKey();

ThriftSearchFeatureSchemaEntry schemaEntry = entry.getValue();

if (shouldSetDefaultValueForFeature(schemaEntry.getFeatureType(), featureId)) {

switch (schemaEntry.getFeatureType()) {

case INT32\_VALUE:

features.getIntValues().putIfAbsent(featureId, 0);

break;

case LONG\_VALUE:

features.getLongValues().putIfAbsent(featureId, 0L);

break;

case DOUBLE\_VALUE:

features.getDoubleValues().putIfAbsent(featureId, 0.0);

break;

default:

throw new IllegalArgumentException(

"Should set default values only for integer, long or double features. Instead, "

+ "found feature " + featureId + " of type " + schemaEntry.getFeatureType());

}

}

}

}

protected void overrideFeatureValues(ThriftSearchResultFeatures features,

ThriftSearchResultFeatures overrideFeatures) {

LOG.info("Features before override {}", features);

if (overrideFeatures.isSetIntValues()) {

overrideFeatures.getIntValues().forEach(features::putToIntValues);

}

if (overrideFeatures.isSetLongValues()) {

overrideFeatures.getLongValues().forEach(features::putToLongValues);

}

if (overrideFeatures.isSetDoubleValues()) {

overrideFeatures.getDoubleValues().forEach(features::putToDoubleValues);

}

if (overrideFeatures.isSetBoolValues()) {

overrideFeatures.getBoolValues().forEach(features::putToBoolValues);

}

if (overrideFeatures.isSetStringValues()) {

overrideFeatures.getStringValues().forEach(features::putToStringValues);

}

if (overrideFeatures.isSetBytesValues()) {

overrideFeatures.getBytesValues().forEach(features::putToBytesValues);

}

if (overrideFeatures.isSetFeatureStoreDiscreteValues()) {

overrideFeatures.getFeatureStoreDiscreteValues().forEach(

features::putToFeatureStoreDiscreteValues);

}

if (overrideFeatures.isSetSparseBinaryValues()) {

overrideFeatures.getSparseBinaryValues().forEach(features::putToSparseBinaryValues);

}

if (overrideFeatures.isSetSparseContinuousValues()) {

overrideFeatures.getSparseContinuousValues().forEach(features::putToSparseContinuousValues);

}

if (overrideFeatures.isSetGeneralTensorValues()) {

overrideFeatures.getGeneralTensorValues().forEach(features::putToGeneralTensorValues);

}

if (overrideFeatures.isSetStringTensorValues()) {

overrideFeatures.getStringTensorValues().forEach(features::putToStringTensorValues);

}

LOG.info("Features after override {}", features);

}

/\*\*

\* Check if a feature is eligible to have its default value automatically set when absent.

\* We have a similar logic for building data record.

\*/

private static boolean shouldSetDefaultValueForFeature(

ThriftSearchFeatureType type, int featureId) {

return ALLOWED\_TYPES\_FOR\_DEFAULT\_FEATURE\_VALUES.contains(type)

&& !NUMERIC\_FEATURES\_FOR\_WHICH\_DEFAULTS\_SHOULD\_NOT\_BE\_SET.contains(featureId)

&& (ExternalTweetFeature.EARLYBIRD\_INDEXED\_FEATURE\_IDS.contains(featureId)

|| ExternalTweetFeature.EARLYBIRD\_DERIVED\_FEATURE\_IDS.contains(featureId));

}

@Override

public void updateRelevanceStats(ThriftSearchResultsRelevanceStats relevanceStats) {

if (relevanceStats == null) {

return;

}

LinearScoringData data = getScoringDataForCurrentDocument();

if (data.tweetAgeInSeconds > relevanceStats.getOldestScoredTweetAgeInSeconds()) {

relevanceStats.setOldestScoredTweetAgeInSeconds(data.tweetAgeInSeconds);

}

relevanceStats.setNumScored(relevanceStats.getNumScored() + 1);

if (data.scoreReturned == SKIP\_HIT) {

relevanceStats.setNumSkipped(relevanceStats.getNumSkipped() + 1);

switch(data.skipReason) {

case ANTIGAMING:

relevanceStats.setNumSkippedForAntiGaming(

relevanceStats.getNumSkippedForAntiGaming() + 1);

break;

case LOW\_REPUTATION:

relevanceStats.setNumSkippedForLowReputation(

relevanceStats.getNumSkippedForLowReputation() + 1);

break;

case LOW\_TEXT\_SCORE:

relevanceStats.setNumSkippedForLowTextScore(

relevanceStats.getNumSkippedForLowTextScore() + 1);

break;

case SOCIAL\_FILTER:

relevanceStats.setNumSkippedForSocialFilter(

relevanceStats.getNumSkippedForSocialFilter() + 1);

break;

case LOW\_FINAL\_SCORE:

relevanceStats.setNumSkippedForLowFinalScore(

relevanceStats.getNumSkippedForLowFinalScore() + 1);

break;

case LOW\_RETWEET\_COUNT:

break;

default:

LOG.warn("Unknown SkipReason: " + data.skipReason);

}

}

if (data.isFollow) {

relevanceStats.setNumFromDirectFollows(relevanceStats.getNumFromDirectFollows() + 1);

}

if (data.isTrusted) {

relevanceStats.setNumFromTrustedCircle(relevanceStats.getNumFromTrustedCircle() + 1);

}

if (data.isReply) {

relevanceStats.setNumReplies(relevanceStats.getNumReplies() + 1);

if (data.isTrusted) {

relevanceStats.setNumRepliesTrusted(relevanceStats.getNumRepliesTrusted() + 1);

} else if (!data.isFollow) {

relevanceStats.setNumRepliesOutOfNetwork(relevanceStats.getNumRepliesOutOfNetwork() + 1);

}

}

if (data.isSelfTweet) {

relevanceStats.setNumSelfTweets(relevanceStats.getNumSelfTweets() + 1);

}

if (data.hasImageUrl || data.hasVideoUrl) {

relevanceStats.setNumWithMedia(relevanceStats.getNumWithMedia() + 1);

}

if (data.hasNewsUrl) {

relevanceStats.setNumWithNews(relevanceStats.getNumWithNews() + 1);

}

if (data.isUserSpam) {

relevanceStats.setNumSpamUser(relevanceStats.getNumSpamUser() + 1);

}

if (data.isUserNSFW) {

relevanceStats.setNumOffensive(relevanceStats.getNumOffensive() + 1);

}

if (data.isUserBot) {

relevanceStats.setNumBot(relevanceStats.getNumBot() + 1);

}

}

@VisibleForTesting

static final class ExplanationWrapper {

private Explanation explanation;

public Explanation getExplanation() {

return explanation;

}

@Override

public String toString() {

return explanation.toString();

}

}

}