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

import java.io.IOException;

import java.util.Map;

import java.util.function.Function;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.NumericDocValues;

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

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

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

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

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

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

import com.twitter.search.common.schema.thriftjava.ThriftCSFType;

import com.twitter.search.common.schema.thriftjava.ThriftFeatureNormalizationType;

public class EarlybirdDocumentFeatures {

private static final Map<Integer, SearchCounter> FEATURE\_CONFIG\_IS\_NULL\_MAP = Maps.newHashMap();

private static final Map<Integer, SearchCounter> FEATURE\_OUTPUT\_TYPE\_IS\_NULL\_MAP =

Maps.newHashMap();

private static final Map<Integer, SearchCounter> NO\_SCHEMA\_FIELD\_FOR\_FEATURE\_MAP =

Maps.newHashMap();

private static final String FEATURE\_CONFIG\_IS\_NULL\_COUNTER\_PATTERN =

"null\_feature\_config\_for\_feature\_id\_%d";

private static final String FEATURE\_OUTPUT\_TYPE\_IS\_NULL\_COUNTER\_PATTERN =

"null\_output\_type\_for\_feature\_id\_%d";

private static final String NO\_SCHEMA\_FIELD\_FOR\_FEATURE\_COUNTER\_PATTERN =

"no\_schema\_field\_for\_feature\_id\_%d";

private static final SearchCounter UNKNOWN\_FEATURE\_OUTPUT\_TYPE\_COUNTER =

SearchCounter.export("unknown\_feature\_output\_type");

private final Map<String, NumericDocValues> numericDocValues = Maps.newHashMap();

private final LeafReader leafReader;

private int docId = -1;

/\*\*

\* Creates a new EarlybirdDocumentFeatures instance that will return feature values based on the

\* NumericDocValues stored in the given LeafReader for the given document.

\*/

public EarlybirdDocumentFeatures(LeafReader leafReader) {

this.leafReader = Preconditions.checkNotNull(leafReader);

}

/\*\*

\* Advances this instance to the given doc ID. The new doc ID must be greater than or equal to the

\* current doc ID stored in this instance.

\*/

public void advance(int target) {

Preconditions.checkArgument(

target >= 0,

"Target (%s) cannot be negative.",

target);

Preconditions.checkArgument(

target >= docId,

"Target (%s) smaller than current doc ID (%s).",

target,

docId);

Preconditions.checkArgument(

target < leafReader.maxDoc(),

"Target (%s) cannot be greater than or equal to the max doc ID (%s).",

target,

leafReader.maxDoc());

docId = target;

}

/\*\*

\* Returns the feature value for the given field.

\*/

public long getFeatureValue(EarlybirdFieldConstant field) throws IOException {

// The index might not have a NumericDocValues instance for this feature.

// This might happen if we dynamically update the feature schema, for example.

//

// Cache the NumericDocValues instances for all accessed features, even if they're null.

String fieldName = field.getFieldName();

NumericDocValues docValues;

if (numericDocValues.containsKey(fieldName)) {

docValues = numericDocValues.get(fieldName);

} else {

docValues = leafReader.getNumericDocValues(fieldName);

numericDocValues.put(fieldName, docValues);

}

return docValues != null && docValues.advanceExact(docId) ? docValues.longValue() : 0L;

}

/\*\*

\* Determines if the given flag is set.

\*/

public boolean isFlagSet(EarlybirdFieldConstant field) throws IOException {

return getFeatureValue(field) != 0;

}

/\*\*

\* Returns the unnormalized value for the given field.

\*/

public double getUnnormalizedFeatureValue(EarlybirdFieldConstant field) throws IOException {

long featureValue = getFeatureValue(field);

ThriftFeatureNormalizationType normalizationType = field.getFeatureNormalizationType();

if (normalizationType == null) {

normalizationType = ThriftFeatureNormalizationType.NONE;

}

switch (normalizationType) {

case NONE:

return featureValue;

case LEGACY\_BYTE\_NORMALIZER:

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

case LEGACY\_BYTE\_NORMALIZER\_WITH\_LOG2:

return MutableFeatureNormalizers.BYTE\_NORMALIZER.unnormAndLog2((byte) featureValue);

case SMART\_INTEGER\_NORMALIZER:

return MutableFeatureNormalizers.SMART\_INTEGER\_NORMALIZER.unnormUpperBound(

(byte) featureValue);

case PREDICTION\_SCORE\_NORMALIZER:

return IntNormalizers.PREDICTION\_SCORE\_NORMALIZER.denormalize((int) featureValue);

default:

throw new IllegalArgumentException(

"Unsupported normalization type " + normalizationType + " for feature "

+ field.getFieldName());

}

}

/\*\*

\* Creates a ThriftSearchResultFeatures instance populated with values for all available features

\* that have a non-zero value set.

\*/

public ThriftSearchResultFeatures getSearchResultFeatures(ImmutableSchemaInterface schema)

throws IOException {

return getSearchResultFeatures(schema, (featureId) -> true);

}

/\*\*

\* Creates a ThriftSearchResultFeatures instance populated with values for all available features

\* that have a non-zero value set.

\*

\* @param schema The schema.

\* @param shouldCollectFeatureId A predicate that determines which features should be collected.

\*/

public ThriftSearchResultFeatures getSearchResultFeatures(

ImmutableSchemaInterface schema,

Function<Integer, Boolean> shouldCollectFeatureId) throws IOException {

Map<Integer, Boolean> boolValues = Maps.newHashMap();

Map<Integer, Double> doubleValues = Maps.newHashMap();

Map<Integer, Integer> intValues = Maps.newHashMap();

Map<Integer, Long> longValues = Maps.newHashMap();

Map<Integer, FeatureConfiguration> idToFeatureConfigMap = schema.getFeatureIdToFeatureConfig();

for (int featureId : schema.getSearchFeatureSchema().getEntries().keySet()) {

if (!shouldCollectFeatureId.apply(featureId)) {

continue;

}

FeatureConfiguration featureConfig = idToFeatureConfigMap.get(featureId);

if (featureConfig == null) {

FEATURE\_CONFIG\_IS\_NULL\_MAP.computeIfAbsent(

featureId,

(fId) -> SearchCounter.export(

String.format(FEATURE\_CONFIG\_IS\_NULL\_COUNTER\_PATTERN, fId))).increment();

continue;

}

ThriftCSFType outputType = featureConfig.getOutputType();

if (outputType == null) {

FEATURE\_OUTPUT\_TYPE\_IS\_NULL\_MAP.computeIfAbsent(

featureId,

(fId) -> SearchCounter.export(

String.format(FEATURE\_OUTPUT\_TYPE\_IS\_NULL\_COUNTER\_PATTERN, fId))).increment();

continue;

}

if (!EarlybirdFieldConstants.hasFieldConstant(featureId)) {

// Should only happen for features that were dynamically added to the schema.

NO\_SCHEMA\_FIELD\_FOR\_FEATURE\_MAP.computeIfAbsent(

featureId,

(fId) -> SearchCounter.export(

String.format(NO\_SCHEMA\_FIELD\_FOR\_FEATURE\_COUNTER\_PATTERN, fId))).increment();

continue;

}

EarlybirdFieldConstant field = EarlybirdFieldConstants.getFieldConstant(featureId);

switch (outputType) {

case BOOLEAN:

if (isFlagSet(field)) {

boolValues.put(featureId, true);

}

break;

case BYTE:

// It's unclear why we don't add this feature to a separate byteValues map...

byte byteFeatureValue = (byte) getFeatureValue(field);

if (byteFeatureValue != 0) {

intValues.put(featureId, (int) byteFeatureValue);

}

break;

case INT:

int intFeatureValue = (int) getFeatureValue(field);

if (intFeatureValue != 0) {

intValues.put(featureId, intFeatureValue);

}

break;

case LONG:

long longFeatureValue = getFeatureValue(field);

if (longFeatureValue != 0) {

longValues.put(featureId, longFeatureValue);

}

break;

case FLOAT:

// It's unclear why we don't add this feature to a separate floatValues map...

float floatFeatureValue = (float) getFeatureValue(field);

if (floatFeatureValue != 0) {

doubleValues.put(featureId, (double) floatFeatureValue);

}

break;

case DOUBLE:

double doubleFeatureValue = getUnnormalizedFeatureValue(field);

if (doubleFeatureValue != 0) {

doubleValues.put(featureId, doubleFeatureValue);

}

break;

default:

UNKNOWN\_FEATURE\_OUTPUT\_TYPE\_COUNTER.increment();

}

}

return new ThriftSearchResultFeatures()

.setBoolValues(boolValues)

.setIntValues(intValues)

.setLongValues(longValues)

.setDoubleValues(doubleValues);

}

}