package com.twitter.search.ingester.pipeline.twitter;

import java.util.List;

import java.util.Set;

import com.google.common.base.Preconditions;

import com.google.common.collect.HashBasedTable;

import com.google.common.collect.Sets;

import com.google.common.collect.Table;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.search.common.indexing.thriftjava.ThriftVersionedEvents;

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

import com.twitter.search.common.schema.SchemaBuilder;

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

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

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

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

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

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

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

/\*\*

\* This class exports counts of fields that are present on processed tweets. It is used to ensure

\* that we are not missing important fields. It is not threadsafe.

\*/

public class FieldStatExporter {

private static final String STAT\_FORMAT = "%s\_penguin\_%d\_documents\_with\_field\_%s";

private static final String UNKNOWN\_FIELD = "%s\_penguin\_%d\_documents\_with\_unknown\_field\_%d";

private final String statPrefix;

private final Schema schema;

private final Table<PenguinVersion, Integer, SearchRateCounter> fieldCounters

= HashBasedTable.create();

private final Set<EarlybirdFieldConstant> encodedTweetFeaturesFields;

private final Set<EarlybirdFieldConstant> extendedEncodedTweetFeaturesFields;

private List<PenguinVersion> penguinVersions;

FieldStatExporter(String statPrefix, Schema schema, List<PenguinVersion> penguinVersions) {

this.statPrefix = statPrefix;

this.schema = schema;

this.penguinVersions = penguinVersions;

this.encodedTweetFeaturesFields =

getEncodedTweetFeaturesFields(EarlybirdFieldConstant.ENCODED\_TWEET\_FEATURES\_FIELD);

this.extendedEncodedTweetFeaturesFields =

getEncodedTweetFeaturesFields(EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD);

for (PenguinVersion version : penguinVersions) {

for (Schema.FieldInfo info : schema.getFieldInfos()) {

String name =

String.format(STAT\_FORMAT, statPrefix, version.getByteValue(), info.getName());

SearchRateCounter counter = SearchRateCounter.export(name);

fieldCounters.put(version, info.getFieldId(), counter);

}

}

}

/\*\*

\* Exports stats counting the number of fields that are present on each document.

\*/

public void addFieldStats(ThriftVersionedEvents event) {

for (PenguinVersion penguinVersion : penguinVersions) {

byte version = penguinVersion.getByteValue();

ThriftIndexingEvent indexingEvent = event.getVersionedEvents().get(version);

Preconditions.checkNotNull(indexingEvent);

// We only want to count each field once per tweet.

Set<Integer> seenFields = Sets.newHashSet();

for (ThriftField field : indexingEvent.getDocument().getFields()) {

int fieldId = field.getFieldConfigId();

if (seenFields.add(fieldId)) {

if (fieldId == EarlybirdFieldConstant.ENCODED\_TWEET\_FEATURES\_FIELD.getFieldId()) {

exportEncodedFeaturesStats(EarlybirdFieldConstant.ENCODED\_TWEET\_FEATURES\_FIELD,

encodedTweetFeaturesFields,

penguinVersion,

field);

} else if (fieldId

== EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD.getFieldId()) {

exportEncodedFeaturesStats(EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD,

extendedEncodedTweetFeaturesFields,

penguinVersion,

field);

} else if (isFeatureField(field)) {

updateCounterForFeatureField(

field.getFieldConfigId(), field.getFieldData().getIntValue(), penguinVersion);

} else {

SearchRateCounter counter = fieldCounters.get(penguinVersion, fieldId);

if (counter == null) {

counter = SearchRateCounter.export(

String.format(UNKNOWN\_FIELD, statPrefix, version, fieldId));

fieldCounters.put(penguinVersion, fieldId, counter);

}

counter.increment();

}

}

}

}

}

private boolean isFeatureField(ThriftField field) {

String fieldName =

EarlybirdFieldConstants.getFieldConstant(field.getFieldConfigId()).getFieldName();

return fieldName.startsWith(EarlybirdFieldConstants.ENCODED\_TWEET\_FEATURES\_FIELD\_NAME

+ SchemaBuilder.CSF\_VIEW\_NAME\_SEPARATOR)

|| fieldName.startsWith(EarlybirdFieldConstants.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD\_NAME

+ SchemaBuilder.CSF\_VIEW\_NAME\_SEPARATOR);

}

private Set<EarlybirdFieldConstant> getEncodedTweetFeaturesFields(

EarlybirdFieldConstant featuresField) {

Set<EarlybirdFieldConstant> schemaFeatureFields = Sets.newHashSet();

String baseFieldNamePrefix =

featuresField.getFieldName() + SchemaBuilder.CSF\_VIEW\_NAME\_SEPARATOR;

for (EarlybirdFieldConstant field : EarlybirdFieldConstant.values()) {

if (field.getFieldName().startsWith(baseFieldNamePrefix)) {

schemaFeatureFields.add(field);

}

}

return schemaFeatureFields;

}

private void exportEncodedFeaturesStats(EarlybirdFieldConstant featuresField,

Set<EarlybirdFieldConstant> schemaFeatureFields,

PenguinVersion penguinVersion,

ThriftField thriftField) {

byte[] encodedFeaturesBytes = thriftField.getFieldData().getBytesValue();

EarlybirdEncodedFeatures encodedTweetFeatures = EarlybirdEncodedFeaturesUtil.fromBytes(

schema.getSchemaSnapshot(), featuresField, encodedFeaturesBytes, 0);

for (EarlybirdFieldConstant field : schemaFeatureFields) {

updateCounterForFeatureField(

field.getFieldId(), encodedTweetFeatures.getFeatureValue(field), penguinVersion);

}

}

private void updateCounterForFeatureField(int fieldId, int value, PenguinVersion penguinVersion) {

if (value != 0) {

SearchRateCounter counter = fieldCounters.get(penguinVersion, fieldId);

if (counter == null) {

counter = SearchRateCounter.export(

String.format(UNKNOWN\_FIELD, statPrefix, penguinVersion, fieldId));

fieldCounters.put(penguinVersion, fieldId, counter);

}

counter.increment();

}

}

public void updatePenguinVersions(List<PenguinVersion> updatedPenguinVersions) {

penguinVersions = updatedPenguinVersions;

}

}