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

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.TimeUnit;

import com.twitter.common.collections.Pair;

import com.twitter.common.util.Clock;

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

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

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

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

import com.twitter.search.common.search.EarlyTerminationState;

import com.twitter.search.earlybird.common.userupdates.UserTable;

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

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

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

import com.twitter.search.earlybird.search.relevance.scoring.BatchHit;

import com.twitter.search.earlybird.search.relevance.scoring.ScoringFunction;

import com.twitter.search.earlybird.stats.EarlybirdSearcherStats;

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

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

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

/\*\*

\* BatchRelevanceTopCollector is similar to the `RelevanceTopCollector` in what it outputs:

\* Collects the top numResults by score, filtering out duplicates

\* and results with scores equal to Flat.MIN\_VALUE.

\* The way that it achieves that is different though: it will score documents through the batch score

\* function instead of scoring documents one by one.

\*/

public class BatchRelevanceTopCollector extends RelevanceTopCollector {

protected final List<BatchHit> hits;

public BatchRelevanceTopCollector(

ImmutableSchemaInterface schema,

RelevanceSearchRequestInfo searchRequestInfo,

ScoringFunction scoringFunction,

EarlybirdSearcherStats searcherStats,

EarlybirdCluster cluster,

UserTable userTable,

Clock clock,

int requestDebugMode) {

super(schema, searchRequestInfo, scoringFunction, searcherStats, cluster, userTable, clock,

requestDebugMode);

this.hits = new ArrayList<>((int) getMaxHitsToProcess());

}

@Override

protected void doCollectWithScore(long tweetID, float score) throws IOException {

Pair<LinearScoringData, ThriftSearchResultFeatures> pair =

scoringFunction.collectFeatures(score);

ThriftSearchResultMetadata metadata = collectMetadata();

hits.add(new BatchHit(pair.getFirst(),

pair.getSecond(),

metadata,

tweetID,

currTimeSliceID));

}

@Override

public EarlyTerminationState innerShouldCollectMore() {

if (hits.size() >= getMaxHitsToProcess()) {

return setEarlyTerminationState(EarlyTerminationState.TERMINATED\_MAX\_HITS\_EXCEEDED);

}

return EarlyTerminationState.COLLECTING;

}

@Override

protected RelevanceSearchResults doGetRelevanceResults() throws IOException {

final long scoringStartNanos = getClock().nowNanos();

float[] scores = scoringFunction.batchScore(hits);

final long scoringEndNanos = getClock().nowNanos();

addToOverallScoringTimeNanos(scoringStartNanos, scoringEndNanos);

exportBatchScoringTime(scoringEndNanos - scoringStartNanos);

for (int i = 0; i < hits.size(); i++) {

BatchHit hit = hits.get(i);

ThriftSearchResultMetadata metadata = hit.getMetadata();

if (!metadata.isSetExtraMetadata()) {

metadata.setExtraMetadata(new ThriftSearchResultExtraMetadata());

}

metadata.getExtraMetadata().setFeatures(hit.getFeatures());

// Populate the ThriftSearchResultMetadata post batch scoring with information from the

// LinearScoringData, which now includes a score.

scoringFunction.populateResultMetadataBasedOnScoringData(

searchRequestInfo.getSearchQuery().getResultMetadataOptions(),

metadata,

hit.getScoringData());

collectWithScoreInternal(

hit.getTweetID(),

hit.getTimeSliceID(),

scores[i],

metadata

);

}

return getRelevanceResultsInternal();

}

private void exportBatchScoringTime(long scoringTimeNanos) {

ThriftSearchRelevanceOptions relevanceOptions = searchRequestInfo.getRelevanceOptions();

if (relevanceOptions.isSetRankingParams()

&& relevanceOptions.getRankingParams().isSetSelectedTensorflowModel()) {

String model = relevanceOptions.getRankingParams().getSelectedTensorflowModel();

SearchTimerStats batchScoringPerModelTimer = SearchTimerStats.export(

String.format("batch\_scoring\_time\_for\_model\_%s", model),

TimeUnit.NANOSECONDS,

false,

true);

batchScoringPerModelTimer.timerIncrement(scoringTimeNanos);

}

}

}