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

import java.io.IOException;

import java.util.List;

import java.util.Map;

import com.google.common.base.Optional;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.apache.lucene.search.Explanation;

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

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

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

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

import com.twitter.search.common.util.ml.prediction\_engine.LightweightLinearModel;

import com.twitter.search.common.util.ml.prediction\_engine.SchemaBasedScoreAccumulator;

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

import com.twitter.search.earlybird.exception.ClientException;

import com.twitter.search.earlybird.ml.ScoringModelsManager;

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

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

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

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

/\*\*

\* Scoring function that uses the scoring models specified from the request.

\*/

public class ModelBasedScoringFunction extends FeatureBasedScoringFunction {

private final SelectedModel[] selectedModels;

private final boolean useLogitScore;

private final boolean isSchemaBased;

private static final SearchCounter NUM\_LEGACY\_MODELS =

SearchCounter.export("scoring\_function\_num\_legacy\_models");

private static final SearchCounter NUM\_SCHEMA\_BASED\_MODELS =

SearchCounter.export("scoring\_function\_num\_schema\_based\_models");

private static final SearchCounter MIXED\_MODEL\_TYPES =

SearchCounter.export("scoring\_function\_mixed\_model\_types");

public ModelBasedScoringFunction(

ImmutableSchemaInterface schema,

ThriftSearchQuery searchQuery,

AntiGamingFilter antiGamingFilter,

ThriftSearchResultType searchResultType,

UserTable userTable,

ScoringModelsManager scoringModelsManager

) throws IOException, ClientException {

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

userTable);

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

Preconditions.checkNotNull(rankingParams);

if (rankingParams.getSelectedModelsSize() <= 0) {

throw new ClientException("Scoring type is MODEL\_BASED but no models were selected");

}

Map<String, Double> models = rankingParams.getSelectedModels();

selectedModels = new SelectedModel[models.size()];

int numSchemaBased = 0;

int i = 0;

for (Map.Entry<String, Double> nameAndWeight : models.entrySet()) {

Optional<LightweightLinearModel> model =

scoringModelsManager.getModel(nameAndWeight.getKey());

if (!model.isPresent()) {

throw new ClientException(String.format(

"Scoring function is MODEL\_BASED. Selected model '%s' not found",

nameAndWeight.getKey()));

}

selectedModels[i] =

new SelectedModel(nameAndWeight.getKey(), nameAndWeight.getValue(), model.get());

if (selectedModels[i].model.isSchemaBased()) {

++numSchemaBased;

NUM\_SCHEMA\_BASED\_MODELS.increment();

} else {

NUM\_LEGACY\_MODELS.increment();

}

++i;

}

// We should either see all models schema-based, or none of them so, if this is not the case,

// we log an error message and fall back to use just the first model, whatever it is.

if (numSchemaBased > 0 && numSchemaBased != selectedModels.length) {

MIXED\_MODEL\_TYPES.increment();

throw new ClientException(

"You cannot mix schema-based and non-schema-based models in the same request, "

+ "models are: " + models.keySet());

}

isSchemaBased = selectedModels[0].model.isSchemaBased();

useLogitScore = rankingParams.isUseLogitScore();

}

@Override

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

ThriftSearchResultFeatures features =

isSchemaBased ? createFeaturesForDocument(data, false).getFeatures() : null;

double score = 0;

for (SelectedModel selectedModel : selectedModels) {

double modelScore = isSchemaBased

? new SchemaBasedScoreAccumulator(selectedModel.model).scoreWith(features, useLogitScore)

: new LegacyScoreAccumulator(selectedModel.model).scoreWith(data, useLogitScore);

score += selectedModel.weight \* modelScore;

}

return score;

}

@Override

protected void generateExplanationForScoring(

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

boolean schemaBased = selectedModels[0].model.isSchemaBased();

ThriftSearchResultFeatures features =

schemaBased ? createFeaturesForDocument(scoringData, false).getFeatures() : null;

// 1. Model-based score

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

float finalScore = 0;

for (SelectedModel selectedModel : selectedModels) {

double modelScore = schemaBased

? new SchemaBasedScoreAccumulator(selectedModel.model).scoreWith(features, useLogitScore)

: new LegacyScoreAccumulator(selectedModel.model).scoreWith(scoringData, useLogitScore);

float weightedScore = (float) (selectedModel.weight \* modelScore);

details.add(Explanation.match(

weightedScore, String.format("model=%s score=%.6f weight=%.3f useLogitScore=%s",

selectedModel.name, modelScore, selectedModel.weight, useLogitScore)));

finalScore += weightedScore;

}

details.add(Explanation.match(

finalScore, String.format("Total model-based score (hit=%s)", isHit), modelExplanations));

}

private static final class SelectedModel {

public final String name;

public final double weight;

public final LightweightLinearModel model;

private SelectedModel(String name, double weight, LightweightLinearModel model) {

this.name = name;

this.weight = weight;

this.model = model;

}

}

}