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

import java.io.IOException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

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

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

import com.twitter.search.common.util.ml.tensorflow\_engine.TensorflowModelsManager;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

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.thrift.EarlybirdRequest;

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

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

import com.twitter.search.queryparser.query.Query;

/\*\*

\* Returns a scoring function for a particular experiment ID.

\*

\* Can be used for a/b testing of different scoring formulas.

\*/

public abstract class ScoringFunctionProvider {

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

/\*\*

\* Returns the scoring function.

\*/

public abstract ScoringFunction getScoringFunction() throws IOException, ClientException;

public static final String RETWEETS\_SCORER\_NAME = "retweets";

public static final String NO\_SPAM\_SCORER\_NAME = "no\_spam";

public static final String TEST\_SCORER\_NAME = "test";

// Whether to avoid time decay when scoring top tweets.

// Top archive does not need time decay.

private static final boolean TOP\_TWEET\_WITH\_DECAY =

EarlybirdConfig.getBool("top\_tweet\_scoring\_with\_decay", true);

/\*\*

\* Abstract class that can be used for ScoringFunctions that don't throw a ClientException.

\*

\* It does throw an IOException but it doesn't throw a ClientException so the name can be a bit

\* misleading.

\*/

public abstract static class NamedScoringFunctionProvider extends ScoringFunctionProvider {

/\*\*

\* Returns the scoring function.

\*/

public abstract ScoringFunction getScoringFunction() throws IOException;

}

/\*\*

\* Returns the scoring function provider with the given name, or null if no such provider exists.

\*/

public static NamedScoringFunctionProvider getScoringFunctionProviderByName(

String name, final ImmutableSchemaInterface schema) {

if (name.equals(NO\_SPAM\_SCORER\_NAME)) {

return new NamedScoringFunctionProvider() {

@Override

public ScoringFunction getScoringFunction() throws IOException {

return new SpamVectorScoringFunction(schema);

}

};

} else if (name.equals(RETWEETS\_SCORER\_NAME)) {

return new NamedScoringFunctionProvider() {

@Override

public ScoringFunction getScoringFunction() throws IOException {

// Production top tweet actually uses this.

if (TOP\_TWEET\_WITH\_DECAY) {

return new RetweetBasedTopTweetsScoringFunction(schema);

} else {

return new RetweetBasedTopTweetsScoringFunction(schema, true);

}

}

};

} else if (name.equals(TEST\_SCORER\_NAME)) {

return new NamedScoringFunctionProvider() {

@Override

public ScoringFunction getScoringFunction() throws IOException {

return new TestScoringFunction(schema);

}

};

}

return null;

}

/\*\*

\* Returns default scoring functions for different scoring function type

\* and provides fallback behavior if model-based scoring function fails

\*/

public static class DefaultScoringFunctionProvider extends ScoringFunctionProvider {

private final EarlybirdRequest request;

private final ImmutableSchemaInterface schema;

private final ThriftSearchQuery searchQuery;

private final AntiGamingFilter antiGamingFilter;

private final UserTable userTable;

private final HitAttributeHelper hitAttributeHelper;

private final Query parsedQuery;

private final ScoringModelsManager scoringModelsManager;

private final TensorflowModelsManager tensorflowModelsManager;

private static final SearchCounter MODEL\_BASED\_SCORING\_FUNCTION\_CREATED =

SearchCounter.export("model\_based\_scoring\_function\_created");

private static final SearchCounter MODEL\_BASED\_FALLBACK\_TO\_LINEAR\_SCORING\_FUNCTION =

SearchCounter.export("model\_based\_fallback\_to\_linear\_scoring\_function");

private static final SearchCounter TENSORFLOW\_BASED\_SCORING\_FUNCTION\_CREATED =

SearchCounter.export("tensorflow\_based\_scoring\_function\_created");

private static final SearchCounter TENSORFLOW\_BASED\_FALLBACK\_TO\_LINEAR\_SCORING\_FUNCTION =

SearchCounter.export("tensorflow\_fallback\_to\_linear\_function\_scoring\_function");

public DefaultScoringFunctionProvider(

final EarlybirdRequest request,

final ImmutableSchemaInterface schema,

final ThriftSearchQuery searchQuery,

final AntiGamingFilter antiGamingFilter,

final UserTable userTable,

final HitAttributeHelper hitAttributeHelper,

final Query parsedQuery,

final ScoringModelsManager scoringModelsManager,

final TensorflowModelsManager tensorflowModelsManager) {

this.request = request;

this.schema = schema;

this.searchQuery = searchQuery;

this.antiGamingFilter = antiGamingFilter;

this.userTable = userTable;

this.hitAttributeHelper = hitAttributeHelper;

this.parsedQuery = parsedQuery;

this.scoringModelsManager = scoringModelsManager;

this.tensorflowModelsManager = tensorflowModelsManager;

}

@Override

public ScoringFunction getScoringFunction() throws IOException, ClientException {

if (searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isSetRankingParams()) {

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

ThriftScoringFunctionType type = params.isSetType()

? params.getType() : ThriftScoringFunctionType.LINEAR; // default type

switch (type) {

case LINEAR:

return createLinear();

case MODEL\_BASED:

if (scoringModelsManager.isEnabled()) {

MODEL\_BASED\_SCORING\_FUNCTION\_CREATED.increment();

return createModelBased();

} else {

// From ScoringModelsManager.NO\_OP\_MANAGER. Fall back to LinearScoringFunction

MODEL\_BASED\_FALLBACK\_TO\_LINEAR\_SCORING\_FUNCTION.increment();

return createLinear();

}

case TENSORFLOW\_BASED:

if (tensorflowModelsManager.isEnabled()) {

TENSORFLOW\_BASED\_SCORING\_FUNCTION\_CREATED.increment();

return createTensorflowBased();

} else {

// Fallback to linear scoring if tf manager is disabled

TENSORFLOW\_BASED\_FALLBACK\_TO\_LINEAR\_SCORING\_FUNCTION.increment();

return createLinear();

}

case TOPTWEETS:

return createTopTweets();

default:

throw new IllegalArgumentException("Unknown scoring type: in " + searchQuery);

}

} else {

LOG.error("No relevance options provided query = " + searchQuery);

return new DefaultScoringFunction(schema);

}

}

private ScoringFunction createLinear() throws IOException {

LinearScoringFunction scoringFunction = new LinearScoringFunction(

schema, searchQuery, antiGamingFilter, ThriftSearchResultType.RELEVANCE,

userTable);

scoringFunction.setHitAttributeHelperAndQuery(hitAttributeHelper, parsedQuery);

return scoringFunction;

}

/\*\*

\* For model based scoring function, ClientException will be throw if client selects an

\* unknown model for scoring manager.

\* {@link com.twitter.search.earlybird.search.relevance.scoring.ModelBasedScoringFunction}

\*/

private ScoringFunction createModelBased() throws IOException, ClientException {

ModelBasedScoringFunction scoringFunction = new ModelBasedScoringFunction(

schema, searchQuery, antiGamingFilter, ThriftSearchResultType.RELEVANCE, userTable,

scoringModelsManager);

scoringFunction.setHitAttributeHelperAndQuery(hitAttributeHelper, parsedQuery);

return scoringFunction;

}

private ScoringFunction createTopTweets() throws IOException {

return new LinearScoringFunction(

schema, searchQuery, antiGamingFilter, ThriftSearchResultType.POPULAR, userTable);

}

private TensorflowBasedScoringFunction createTensorflowBased()

throws IOException, ClientException {

TensorflowBasedScoringFunction tfScoringFunction = new TensorflowBasedScoringFunction(

request, schema, searchQuery, antiGamingFilter,

ThriftSearchResultType.RELEVANCE, userTable, tensorflowModelsManager);

tfScoringFunction.setHitAttributeHelperAndQuery(hitAttributeHelper, parsedQuery);

return tfScoringFunction;

}

}

}