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

import java.io.IOException;

import java.nio.FloatBuffer;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableMap;

import org.apache.lucene.search.Explanation;

import org.tensorflow.Tensor;

import com.twitter.common.collections.Pair;

import com.twitter.search.common.constants.thriftjava.ThriftQuerySource;

import com.twitter.search.common.features.EarlybirdRankingDerivedFeature;

import com.twitter.search.common.features.FeatureHandler;

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

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

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

import com.twitter.search.earlybird.EarlybirdSearcher;

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

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

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

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

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

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

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

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

import com.twitter.search.modeling.common.TweetFeaturesUtils;

import com.twitter.tfcompute\_java.TFModelRunner;

/\*\*

\* TensorflowBasedScoringFunction relies on a TF model for scoring tweets

\* Only the `batchScore` part is implemented

\*/

public class TensorflowBasedScoringFunction extends FeatureBasedScoringFunction {

private final TFModelRunner tfModelRunner;

// https://stackoverflow.com/questions/37849322/how-to-understand-the-term-tensor-in-tensorflow

// for more information on this notation - in short, a TF graph is made

// of TF operations and doesn't have a first order notion of tensors

// The notation <operation>:<index> will maps to the <index> output of the

// <operation> contained in the TF graph.

private static final String INPUT\_VALUES = "input\_sparse\_tensor\_values:0";

private static final String INPUT\_INDICES = "input\_sparse\_tensor\_indices:0";

private static final String INPUT\_SHAPE = "input\_sparse\_tensor\_shape:0";

private static final String OUTPUT\_NODE = "output\_scores:0";

private final Map<Integer, Long> featureSchemaIdToMlApiId;

private final Map<Long, Float> tweetIdToScoreMap = new HashMap<>();

private final EarlybirdRequest request;

public TensorflowBasedScoringFunction(

EarlybirdRequest request,

ImmutableSchemaInterface schema,

ThriftSearchQuery searchQuery,

AntiGamingFilter antiGamingFilter,

ThriftSearchResultType searchResultType,

UserTable userTable,

TensorflowModelsManager tensorflowModelsManager

) throws IOException, ClientException {

super(

"TensorflowBasedScoringFunction",

schema,

searchQuery,

antiGamingFilter,

searchResultType,

userTable

);

this.request = request;

String modelName = searchQuery.getRelevanceOptions().getRankingParams().selectedTensorflowModel;

this.featureSchemaIdToMlApiId = tensorflowModelsManager.getFeatureSchemaIdToMlApiId();

if (modelName == null) {

throw new ClientException("Scoring type is TENSORFLOW\_BASED but no model was selected");

} else if (!tensorflowModelsManager.getModel(modelName).isPresent()) {

throw new ClientException(

"Scoring type is TENSORFLOW\_BASED. Model "

+ modelName

+ " is not present."

);

}

if (searchQuery.getRelevanceOptions().getRankingParams().isEnableHitDemotion()) {

throw new ClientException(

"Hit attribute demotion is not supported with TENSORFLOW\_BASED scoring type");

}

tfModelRunner = tensorflowModelsManager.getModel(modelName).get();

}

/\*\*

\* Single item scoring just returns the lucene score to be used during the batching phase.

\*/

@Override

protected float score(float luceneQueryScore) {

return luceneQueryScore;

}

@Override

public Pair<LinearScoringData, ThriftSearchResultFeatures> collectFeatures(

float luceneQueryScore) throws IOException {

LinearScoringData linearScoringData = updateLinearScoringData(luceneQueryScore);

ThriftSearchResultFeatures features =

createFeaturesForDocument(linearScoringData, true).getFeatures();

return new Pair<>(linearScoringData, features);

}

@Override

protected FeatureHandler createFeaturesForDocument(

LinearScoringData linearScoringData,

boolean ignoreDefaultValues) throws IOException {

return super.createFeaturesForDocument(linearScoringData,

ignoreDefaultValues)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_TREND\_CLICK,

request.querySource == ThriftQuerySource.TREND\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_TYPED\_QUERY,

request.querySource == ThriftQuerySource.TYPED\_QUERY)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_TYPEAHEAD\_CLICK,

request.querySource == ThriftQuerySource.TYPEAHEAD\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_HASHTAG\_CLICK,

request.querySource == ThriftQuerySource.RECENT\_SEARCH\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_RECENT\_SEARCH\_CLICK,

request.querySource == ThriftQuerySource.RECENT\_SEARCH\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_PROFILE\_CLICK,

request.querySource == ThriftQuerySource.PROFILE\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_API\_CALL,

request.querySource == ThriftQuerySource.API\_CALL)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_PROMOTED\_TREND\_CLICK,

request.querySource == ThriftQuerySource.PROMOTED\_TREND\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_SAVED\_SEARCH\_CLICK,

request.querySource == ThriftQuerySource.SAVED\_SEARCH\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_CASHTAG\_CLICK,

request.querySource == ThriftQuerySource.CASHTAG\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_SPELLING\_EXPANSION\_REVERT\_CLICK,

request.querySource == ThriftQuerySource.SPELLING\_EXPANSION\_REVERT\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_SPELLING\_SUGGESTION\_CLICK,

request.querySource == ThriftQuerySource.SPELLING\_SUGGESTION\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_LOGGED\_OUT\_HOME\_TREND\_CLICK,

request.querySource == ThriftQuerySource.LOGGED\_OUT\_HOME\_TREND\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_RELATED\_QUERY\_CLICK,

request.querySource == ThriftQuerySource.RELATED\_QUERY\_CLICK)

.addBoolean(EarlybirdRankingDerivedFeature.QUERY\_SOURCE\_AUTO\_SPELL\_CORRECT\_REVERT\_CLICK,

request.querySource == ThriftQuerySource.AUTO\_SPELL\_CORRECT\_REVERT\_CLICK);

}

/\*\*

\* Return scores computed in batchScore() if forExplanation is true.

\*/

@Override

protected double computeScore(LinearScoringData data, boolean forExplanation) {

Preconditions.checkState(forExplanation,

"forExplanation is false. computeScore() should only be used for explanation creation");

return tweetIdToScoreMap.get(tweetIDMapper.getTweetID(getCurrentDocID()));

}

@Override

protected void generateExplanationForScoring(

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

}

@VisibleForTesting

SparseTensor createInputTensor(ThriftSearchResultFeatures[] featuresForDocs) {

// Moving this across outside of the request path

// would reduce the allocation cost and make the `ByteBuffer`s

// long lived - would need one per thread.

SparseTensor sparseTensor =

new SparseTensor(featuresForDocs.length, featureSchemaIdToMlApiId.size());

for (ThriftSearchResultFeatures features : featuresForDocs) {

updateSparseTensor(sparseTensor, features);

}

return sparseTensor;

}

private void addSchemaBooleanFeatures(SparseTensor sparseTensor,

Map<Integer, Boolean> booleanMap) {

if (booleanMap == null || booleanMap.isEmpty()) {

return;

}

for (Map.Entry<Integer, Boolean> entry : booleanMap.entrySet()) {

Preconditions.checkState(featureSchemaIdToMlApiId.containsKey(entry.getKey()));

sparseTensor.addValue(

featureSchemaIdToMlApiId.get(entry.getKey()), entry.getValue() ? 1f : 0f);

}

}

private void addSchemaContinuousFeatures(SparseTensor sparseTensor,

Map<Integer, ? extends Number> valueMap) {

if (valueMap == null || valueMap.isEmpty()) {

return;

}

for (Map.Entry<Integer, ? extends Number> entry : valueMap.entrySet()) {

Integer id = entry.getKey();

// SEARCH-26795

if (!TweetFeaturesUtils.isFeatureDiscrete(id)) {

Preconditions.checkState(featureSchemaIdToMlApiId.containsKey(id));

sparseTensor.addValue(

featureSchemaIdToMlApiId.get(id), entry.getValue().floatValue());

}

}

}

private void updateSparseTensor(SparseTensor sparseTensor, ThriftSearchResultFeatures features) {

addSchemaBooleanFeatures(sparseTensor, features.getBoolValues());

addSchemaContinuousFeatures(sparseTensor, features.getIntValues());

addSchemaContinuousFeatures(sparseTensor, features.getLongValues());

addSchemaContinuousFeatures(sparseTensor, features.getDoubleValues());

sparseTensor.incNumRecordsSeen();

}

private float[] batchScoreInternal(ThriftSearchResultFeatures[] featuresForDocs) {

int nbDocs = featuresForDocs.length;

float[] backingArrayResults = new float[nbDocs];

SparseTensor sparseTensor = createInputTensor(featuresForDocs);

Tensor<?> sparseValues =

Tensor.create(

Float.class,

sparseTensor.getSparseValuesShape(),

sparseTensor.getSparseValues());

Tensor<?> sparseIndices =

Tensor.create(

Long.class,

sparseTensor.getSparseIndicesShape(),

sparseTensor.getSparseIndices());

Tensor<?> sparseShape =

Tensor.create(

Long.class,

sparseTensor.getSparseShapeShape(),

sparseTensor.getSparseShape());

Map<String, Tensor<?>> inputMap = ImmutableMap.of(

INPUT\_VALUES, sparseValues,

INPUT\_INDICES, sparseIndices,

INPUT\_SHAPE, sparseShape

);

List<String> output = ImmutableList.of(OUTPUT\_NODE);

Map<String, Tensor<?>> outputs = tfModelRunner.run(

inputMap,

output,

ImmutableList.of()

);

Tensor<?> outputTensor = outputs.get(OUTPUT\_NODE);

try {

FloatBuffer finalResultBuffer =

FloatBuffer.wrap(backingArrayResults, 0, nbDocs);

outputTensor.writeTo(finalResultBuffer);

} finally {

// Close tensors to avoid memory leaks

sparseValues.close();

sparseIndices.close();

sparseShape.close();

if (outputTensor != null) {

outputTensor.close();

}

}

return backingArrayResults;

}

/\*\*

\* Compute the score for a list of hits. Not thread safe.

\* @return Array of scores

\*/

@Override

public float[] batchScore(List<BatchHit> hits) throws IOException {

ThriftSearchResultFeatures[] featuresForDocs = new ThriftSearchResultFeatures[hits.size()];

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

// This is a gigantic allocation, but the models are trained to depend on unset values having

// a default.

BatchHit hit = hits.get(i);

ThriftSearchResultFeatures features = hit.getFeatures().deepCopy();

// Adjust features of a hit based on overrides provided by relevance options. Should mostly

// be used for debugging purposes.

adjustHitScoringFeatures(hit, features);

setDefaultFeatureValues(features);

featuresForDocs[i] = features;

}

float[] scores = batchScoreInternal(featuresForDocs);

float[] finalScores = new float[hits.size()];

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

LinearScoringData data = hits.get(i).getScoringData();

if (data.skipReason != null && data.skipReason != LinearScoringData.SkipReason.NOT\_SKIPPED) {

// If the hit should be skipped, overwrite the score with SKIP\_HIT

scores[i] = SKIP\_HIT;

}

// If explanations enabled, Add scores to map. Will be used in computeScore()

if (EarlybirdSearcher.explanationsEnabled(debugMode)) {

tweetIdToScoreMap.put(hits.get(i).getTweetID(), scores[i]);

}

finalScores[i] = postScoreComputation(

data,

scores[i],

false, // cannot get the hit attribution info for this hit at this point in time

null);

}

return finalScores;

}

private void adjustHitScoringFeatures(BatchHit hit, ThriftSearchResultFeatures features) {

if (request.isSetSearchQuery() && request.getSearchQuery().isSetRelevanceOptions()) {

ThriftSearchRelevanceOptions relevanceOptions =

request.getSearchQuery().getRelevanceOptions();

if (relevanceOptions.isSetPerTweetFeaturesOverride()

&& relevanceOptions.getPerTweetFeaturesOverride().containsKey(hit.getTweetID())) {

overrideFeatureValues(

features,

relevanceOptions.getPerTweetFeaturesOverride().get(hit.getTweetID()));

}

if (relevanceOptions.isSetPerUserFeaturesOverride()

&& relevanceOptions.getPerUserFeaturesOverride().containsKey(

hit.getScoringData().fromUserId)) {

overrideFeatureValues(

features,

relevanceOptions.getPerUserFeaturesOverride().get(hit.getScoringData().fromUserId));

}

if (relevanceOptions.isSetGlobalFeaturesOverride()) {

overrideFeatureValues(

features, relevanceOptions.getGlobalFeaturesOverride());

}

}

}

}