package com.twitter.search.common.util.ml.prediction\_engine;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Map;

import javax.annotation.Nullable;

import com.google.common.base.Preconditions;

import com.twitter.ml.api.Feature;

import com.twitter.search.common.file.AbstractFile;

/\*\*

\* Provides an interface to the weights associated to the features of a linear model trained

\* with Prediction Engine.

\*

\* This class is used along with ScoreAccumulator to efficiently score instances. It supports only

\* a limited set of features:

\*

\* - Only linear models are supported.

\* - Only binary and continuous features (i.e. it doesn't support discrete/categorical features).

\* - It supports the MDL discretizer (but not the one based on trees).

\* - It doesn't support feature crossings.

\*

\* Instances of this class should be created using only the load methods (loadFromHdfs and

\* loadFromLocalFile).

\*

\* IMPORTANT:

\*

\* Use this class, and ScoreAccumulator, ONLY when runtime is a major concern. Otherwise, consider

\* using Prediction Engine as a library. Ideally, we should access directly the structures that

\* Prediction Engine creates when it loads a model, instead of parsing a text file with the

\* feature weights.

\*

\* The discretized feature bins created by MDL may be too fine to be displayed properly in the

\* parsed text file and there may be bins with the same min value. A binary search finding the

\* bin for a same feature value therefore may end up with different bins/scores in different runs,

\* producing unstable scores. See SEARCHQUAL-15957 for more detail.

\*

\* @see com.twitter.ml.tool.prediction.ModelInterpreter

\*/

public class LightweightLinearModel {

protected final double bias;

protected final boolean schemaBased;

protected final String name;

// for legacy metadata based model

protected final Map<Feature<Boolean>, Double> binaryFeatures;

protected final Map<Feature<Double>, Double> continuousFeatures;

protected final Map<Feature<Double>, DiscretizedFeature> discretizedFeatures;

// for schema-based model

protected final Map<Integer, Double> binaryFeaturesById;

protected final Map<Integer, Double> continuousFeaturesById;

protected final Map<Integer, DiscretizedFeature> discretizedFeaturesById;

private static final String SCHEMA\_BASED\_SUFFIX = ".schema\_based";

LightweightLinearModel(

String modelName,

double bias,

boolean schemaBased,

@Nullable Map<Feature<Boolean>, Double> binaryFeatures,

@Nullable Map<Feature<Double>, Double> continuousFeatures,

@Nullable Map<Feature<Double>, DiscretizedFeature> discretizedFeatures,

@Nullable Map<Integer, Double> binaryFeaturesById,

@Nullable Map<Integer, Double> continuousFeaturesById,

@Nullable Map<Integer, DiscretizedFeature> discretizedFeaturesById) {

this.name = modelName;

this.bias = bias;

this.schemaBased = schemaBased;

// legacy feature maps

this.binaryFeatures =

schemaBased ? null : Preconditions.checkNotNull(binaryFeatures);

this.continuousFeatures =

schemaBased ? null : Preconditions.checkNotNull(continuousFeatures);

this.discretizedFeatures =

schemaBased ? null : Preconditions.checkNotNull(discretizedFeatures);

// schema based feature maps

this.binaryFeaturesById =

schemaBased ? Preconditions.checkNotNull(binaryFeaturesById) : null;

this.continuousFeaturesById =

schemaBased ? Preconditions.checkNotNull(continuousFeaturesById) : null;

this.discretizedFeaturesById =

schemaBased ? Preconditions.checkNotNull(discretizedFeaturesById) : null;

}

public String getName() {

return name;

}

/\*\*

\* Create model for legacy features

\*/

protected static LightweightLinearModel createForLegacy(

String modelName,

double bias,

Map<Feature<Boolean>, Double> binaryFeatures,

Map<Feature<Double>, Double> continuousFeatures,

Map<Feature<Double>, DiscretizedFeature> discretizedFeatures) {

return new LightweightLinearModel(modelName, bias, false,

binaryFeatures, continuousFeatures, discretizedFeatures,

null, null, null);

}

/\*\*

\* Create model for schema-based features

\*/

protected static LightweightLinearModel createForSchemaBased(

String modelName,

double bias,

Map<Integer, Double> binaryFeaturesById,

Map<Integer, Double> continuousFeaturesById,

Map<Integer, DiscretizedFeature> discretizedFeaturesById) {

return new LightweightLinearModel(modelName, bias, true,

null, null, null,

binaryFeaturesById, continuousFeaturesById, discretizedFeaturesById);

}

public boolean isSchemaBased() {

return schemaBased;

}

/\*\*

\* Loads a model from a text file.

\*

\* See the javadoc of the constructor for more details on how to create the file from a trained

\* Prediction Engine model.

\*

\* If schemaBased is true, the featureContext is ignored.

\*/

public static LightweightLinearModel load(

String modelName,

BufferedReader reader,

boolean schemaBased,

CompositeFeatureContext featureContext) throws IOException {

ModelBuilder builder = schemaBased

? new SchemaBasedModelBuilder(modelName, featureContext.getFeatureSchema())

: new LegacyModelBuilder(modelName, featureContext.getLegacyContext());

String line;

while ((line = reader.readLine()) != null) {

builder.parseLine(line);

}

return builder.build();

}

/\*\*

\* Loads a model from a local text file.

\*

\* See the javadoc of the constructor for more details on how to create the file from a trained

\* Prediction Engine model.

\*/

public static LightweightLinearModel loadFromLocalFile(

String modelName,

CompositeFeatureContext featureContext,

String fileName) throws IOException {

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

boolean schemaBased = modelName.endsWith(SCHEMA\_BASED\_SUFFIX);

return load(modelName, reader, schemaBased, featureContext);

}

}

/\*\*

\* Loads a model from a file in the local filesystem or in HDFS.

\*

\* See the javadoc of the constructor for more details on how to create the file from a trained

\* Prediction Engine model.

\*/

public static LightweightLinearModel load(

String modelName, CompositeFeatureContext featureContext, AbstractFile modelFile)

throws IOException {

try (BufferedReader reader = modelFile.getCharSource().openBufferedStream()) {

boolean schemaBased = modelName.endsWith(SCHEMA\_BASED\_SUFFIX);

return load(modelName, reader, schemaBased, featureContext);

}

}

public String toString() {

return String.format("LightweightLinearModel. {bias=%s binary=%s continuous=%s discrete=%s}",

this.bias, this.binaryFeatures, this.continuousFeatures, this.discretizedFeatures);

}

}