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

import java.util.Collection;

import java.util.Comparator;

import java.util.List;

import com.google.common.collect.Lists;

import com.twitter.ml.api.FeatureParser;

import com.twitter.ml.api.transform.DiscretizerTransform;

import com.twitter.ml.tool.prediction.ModelInterpreter;

/\*\*

\* The base model builder for LightweightLinearModels.

\*/

public abstract class BaseModelBuilder implements ModelBuilder {

// Ignore features that have an absolute weight lower than this value

protected static final double MIN\_WEIGHT = 1e-9;

private static final String BIAS\_FIELD\_NAME = ModelInterpreter.BIAS\_FIELD\_NAME;

static final String DISCRETIZER\_NAME\_SUFFIX =

"." + DiscretizerTransform.DEFAULT\_FEATURE\_NAME\_SUFFIX;

protected final String modelName;

protected double bias;

public BaseModelBuilder(String modelName) {

this.modelName = modelName;

this.bias = 0.0;

}

/\*\*

\* Collects all the ranges of a discretized feature and sorts them.

\*/

static DiscretizedFeature buildFeature(Collection<DiscretizedFeatureRange> ranges) {

List<DiscretizedFeatureRange> sortedRanges = Lists.newArrayList(ranges);

sortedRanges.sort(Comparator.comparingDouble(a -> a.minValue));

double[] splits = new double[ranges.size()];

double[] weights = new double[ranges.size()];

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

splits[i] = sortedRanges.get(i).minValue;

weights[i] = sortedRanges.get(i).weight;

}

return new DiscretizedFeature(splits, weights);

}

/\*\*

\* Parses a line from the interpreted model text file. See the javadoc of the constructor for

\* more details about how to create the text file.

\* <p>

\* The file uses TSV format with 3 columns:

\* <p>

\* Model name (Generated by ML API, but ignored by this class)

\* Feature definition:

\* Name of the feature or definition from the MDL discretizer.

\* Weight:

\* Weight of the feature using LOGIT scale.

\* <p>

\* When it parses each line, it stores the weights for all the features defined in the context,

\* as well as the bias, but it ignores any other feature (e.g. label, prediction or

\* meta.record\_weight) and features with a small absolute weight (see MIN\_WEIGHT).

\* <p>

\* Example lines:

\* <p>

\* model\_name bias 0.019735312089324074

\* model\_name demo.binary\_feature 0.06524706073105327

\* model\_name demo.continuous\_feature 0.0

\* model\_name demo.continuous\_feature.dz/dz\_model=mdl/dz\_range=-inf\_3.58e-01 0.07155931927263737

\* model\_name demo.continuous\_feature.dz/dz\_model=mdl/dz\_range=3.58e-01\_inf -0.08979256264865387

\*

\* @see ModelInterpreter

\* @see DiscretizerTransform

\*/

@Override

public ModelBuilder parseLine(String line) {

String[] columns = line.split("\t");

if (columns.length != 3) {

return this;

}

// columns[0] has the model name, which we don't need

String featureName = columns[1];

double weight = Double.parseDouble(columns[2]);

if (BIAS\_FIELD\_NAME.equals(featureName)) {

bias = weight;

return this;

}

FeatureParser parser = FeatureParser.parse(featureName);

String baseName = parser.getBaseName();

if (Math.abs(weight) < MIN\_WEIGHT && !baseName.endsWith(DISCRETIZER\_NAME\_SUFFIX)) {

// skip, unless it represents a range of a discretized feature.

// discretized features with all zeros should also be removed, but will handle that later

return this;

}

addFeature(baseName, weight, parser);

return this;

}

/\*\*

\* Adds feature to the model

\*/

protected abstract void addFeature(String baseName, double weight, FeatureParser parser);

@Override

public abstract LightweightLinearModel build();

}