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

import java.util.Map;

import com.google.common.collect.HashMultimap;

import com.google.common.collect.Maps;

import com.google.common.collect.Multimap;

import com.twitter.ml.api.Feature;

import com.twitter.ml.api.FeatureContext;

import com.twitter.ml.api.FeatureParser;

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

/\*\*

\* The builder for a model based on the legacy (non-schema-based) features.

\* See also SchemaBasedModelBuilder.

\*/

public final class LegacyModelBuilder extends BaseModelBuilder {

private final Map<String, Feature> featuresByName;

// for legacy features

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

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

private final Multimap<Feature<Double>, DiscretizedFeatureRange> discretizedFeatureRanges;

LegacyModelBuilder(String modelName, FeatureContext context) {

super(modelName);

featuresByName = getFeaturesByName(context);

binaryFeatures = Maps.newHashMap();

continuousFeatures = Maps.newHashMap();

discretizedFeatureRanges = HashMultimap.create();

}

private static Map<String, Feature> getFeaturesByName(FeatureContext featureContext) {

Map<String, Feature> featuresByName = Maps.newHashMap();

for (Feature<?> feature : featureContext.getAllFeatures()) {

featuresByName.put(feature.getFeatureName(), feature);

}

return featuresByName;

}

@Override

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

Feature feature = featuresByName.get(baseName);

if (feature != null) {

switch (feature.getFeatureType()) {

case BINARY:

binaryFeatures.put(feature, weight);

break;

case CONTINUOUS:

continuousFeatures.put(feature, weight);

break;

default:

throw new IllegalArgumentException(

String.format("Unsupported feature type: %s", feature));

}

} else if (baseName.endsWith(DISCRETIZER\_NAME\_SUFFIX)

&& parser.getExtension().containsKey(DiscretizerTransform.DEFAULT\_RANGE\_EXT)) {

String featureName =

baseName.substring(0, baseName.length() - DISCRETIZER\_NAME\_SUFFIX.length());

feature = featuresByName.get(featureName);

if (feature == null) {

return;

}

String rangeSpec = parser.getExtension().get(DiscretizerTransform.DEFAULT\_RANGE\_EXT);

discretizedFeatureRanges.put(feature, new DiscretizedFeatureRange(weight, rangeSpec));

}

}

@Override

public LightweightLinearModel build() {

Map<Feature<Double>, DiscretizedFeature> discretizedFeatures = Maps.newHashMap();

for (Feature<Double> feature : discretizedFeatureRanges.keySet()) {

DiscretizedFeature discretizedFeature =

BaseModelBuilder.buildFeature(discretizedFeatureRanges.get(feature));

if (!discretizedFeature.allValuesBelowThreshold(MIN\_WEIGHT)) {

discretizedFeatures.put(feature, discretizedFeature);

}

}

return LightweightLinearModel.createForLegacy(

modelName, bias, binaryFeatures, continuousFeatures, discretizedFeatures);

}

}