package com.twitter.search.common.util.ml;

import java.util.Map;

import com.google.common.annotations.VisibleForTesting;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.base.Function;

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

import com.twitter.search.common.util.io.TextFileLoadingUtils;

import it.unimi.dsi.fastutil.objects.Object2FloatMap;

import it.unimi.dsi.fastutil.objects.Object2FloatOpenHashMap;

/\*\*

\* Represents a linear model for scoring and classification.

\*

\* Features are represented as arbitrary strings, making this a fairly flexible implementation

\* (at the cost of some performance, since all operations require hash lookups). Instances

\* and weights are both encoded sparsely (as maps) so this implementation is well suited to

\* models with large feature sets where most features are inactive at a given time. Weights

\* for unknown features are assumed to be 0.

\*

\*/

public class StringMapBasedLinearModel implements MapBasedLinearModel<String> {

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

protected final Object2FloatMap<String> model = new Object2FloatOpenHashMap<>();

/\*\*

\* Creates a model from a map of weights.

\*

\* @param weights Feature weights.

\*/

public StringMapBasedLinearModel(Map<String, Float> weights) {

model.putAll(weights);

model.defaultReturnValue(0.0f);

}

/\*\*

\* Get the weight of a feature

\* @param featureName

\* @return

\*/

public float getWeight(String featureName) {

return model.getFloat(featureName);

}

/\*\*

\* Get the full weight map

\*/

@VisibleForTesting

protected Map<String, Float> getWeights() {

return model;

}

/\*\*

\* Evaluate using this model given a feature vector.

\* @param values The feature vector in format of a hashmap.

\* @return

\*/

@Override

public float score(Map<String, Float> values) {

float score = 0.0f;

for (Map.Entry<String, Float> value : values.entrySet()) {

String featureName = value.getKey();

float weight = getWeight(featureName);

if (weight != 0.0f) {

score += weight \* value.getValue();

if (LOG.isDebugEnabled()) {

LOG.debug(String.format("%s = %.3f \* %.3f = %.3f, ",

featureName, weight, value.getValue(),

weight \* value.getValue()));

}

}

}

if (LOG.isDebugEnabled()) {

LOG.debug(String.format("Score = %.3f", score));

}

return score;

}

/\*\*

\* Determines whether an instance is positive.

\*/

@Override

public boolean classify(Map<String, Float> values) {

return classify(0.0f, values);

}

@Override

public boolean classify(float threshold, Map<String, Float> values) {

return score(values) > threshold;

}

public int size() {

return model.size();

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("StringMapBasedLinearModel[");

for (Map.Entry<String, Float> entry : model.entrySet()) {

sb.append(String.format("(%s = %.3f), ", entry.getKey(), entry.getValue()));

}

sb.append("]");

return sb.toString();

}

/\*\*

\* Loads the model from a TSV file with the following format:

\*

\* feature\_name \t weight

\*/

public static StringMapBasedLinearModel loadFromFile(AbstractFile fileHandle) {

Map<String, Float> weights =

TextFileLoadingUtils.loadMapFromFile(

fileHandle,

(Function<String, Float>) item -> Float.parseFloat(item));

return new StringMapBasedLinearModel(weights);

}

}