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

import java.io.IOException;

import java.util.EnumMap;

import java.util.EnumSet;

import java.util.Map;

import java.util.Set;

import com.google.common.base.Preconditions;

import com.google.common.base.Predicates;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Maps;

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

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

/\*\*

\* Represents a linear model for scoring and classification.

\*

\* The list of features is defined by an Enum class. The model weights and instances are

\* represented as maps that must contain an entry for all the values of the enum.

\*

\*/

public class EnumBasedLinearModel<K extends Enum<K>> implements MapBasedLinearModel<K> {

private final EnumSet<K> features;

private final EnumMap<K, Float> weights;

/\*\*

\* Creates a model from a map of weights.

\*

\* @param enumType Enum used for the keys

\* @param weights Feature weights.

\*/

public EnumBasedLinearModel(Class<K> enumType, Map<K, Float> weights) {

features = EnumSet.allOf(enumType);

EnumMap<K, Float> enumWeights =

new EnumMap<>(Maps.filterValues(weights, Predicates.notNull()));

Preconditions.checkArgument(features.equals(enumWeights.keySet()),

"The model does not include weights for all the available features");

this.weights = enumWeights;

}

public ImmutableMap<K, Float> getWeights() {

return Maps.immutableEnumMap(weights);

}

@Override

public float score(Map<K, Float> instance) {

float total = 0;

for (Map.Entry<K, Float> weightEntry : weights.entrySet()) {

Float feature = instance.get(weightEntry.getKey());

if (feature != null) {

total += weightEntry.getValue() \* feature;

}

}

return total;

}

/\*\*

\* Determines whether an instance is positive.

\*/

@Override

public boolean classify(float threshold, Map<K, Float> instance) {

return score(instance) > threshold;

}

@Override

public boolean classify(Map<K, Float> instance) {

return classify(0, instance);

}

@Override

public String toString() {

return String.format("EnumBasedLinearModel[%s]", weights);

}

/\*\*

\* Creates a model where all the features have the same weight.

\* This method is useful for generating the feature vectors for training a new model.

\*/

public static <T extends Enum<T>> EnumBasedLinearModel<T> createWithEqualWeight(Class<T> enumType,

Float weight) {

EnumSet<T> features = EnumSet.allOf(enumType);

EnumMap<T, Float> weights = Maps.newEnumMap(enumType);

for (T feature : features) {

weights.put(feature, weight);

}

return new EnumBasedLinearModel<>(enumType, weights);

}

/\*\*

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

\*

\* feature\_name \t weight

\*/

public static <T extends Enum<T>> EnumBasedLinearModel<T> createFromFile(

Class<T> enumType, AbstractFile path) throws IOException {

return new EnumBasedLinearModel<>(enumType, loadWeights(enumType, path, true));

}

/\*\*

\* Loads the model from a TSV file, using a default weight of 0 for missing features.

\*

\* File format:

\*

\* feature\_name \t weight

\*/

public static <T extends Enum<T>> EnumBasedLinearModel<T> createFromFileSafe(

Class<T> enumType, AbstractFile path) throws IOException {

return new EnumBasedLinearModel<>(enumType, loadWeights(enumType, path, false));

}

/\*\*

\* Creates a map of (feature\_name, weight) from a TSV file.

\*

\* If strictMode is true, it will throw an exception if the file doesn't contain all the

\* features declared in the enum. Otherwise, it will use zero as default value.

\*

\*/

private static <T extends Enum<T>> EnumMap<T, Float> loadWeights(

Class<T> enumType, AbstractFile fileHandle, boolean strictMode) throws IOException {

Map<String, Float> weightsFromFile =

TextFileLoadingUtils.loadMapFromFile(fileHandle, input -> Float.parseFloat(input));

EnumMap<T, Float> weights = Maps.newEnumMap(enumType);

Set<T> expectedFeatures = EnumSet.allOf(enumType);

if (!strictMode) {

for (T feature : expectedFeatures) {

weights.put(feature, 0f);

}

}

for (String featureName : weightsFromFile.keySet()) {

Float weight = weightsFromFile.get(featureName);

weights.put(Enum.valueOf(enumType, featureName.toUpperCase()), weight);

}

Preconditions.checkArgument(expectedFeatures.equals(weights.keySet()),

"Model does not contain weights for all the features");

return weights;

}

}