package com.twitter.search.common.encoding.features;

import java.util.Map;

import java.util.SortedSet;

import java.util.TreeMap;

import com.google.common.base.Preconditions;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

/\*\*

\* Normalizes values to predefined bins.

\* If the value to normalize is lower than the lowest bin defined, normalizes to Byte.MIN\_VALUE.

\*/

public class BinByteNormalizer extends ByteNormalizer {

private final TreeMap<Double, Byte> bins = Maps.newTreeMap();

private final TreeMap<Byte, Double> reverseBins = Maps.newTreeMap();

/\*\*

\* Constructs a normalizer using predefined bins.

\* @param bins A mapping between the upper bound of a value and the bin it should normalize to.

\* For example providing a map with 2 entries, {5=>1, 10=>2} will normalize as follows:

\* values under 5: Byte.MIN\_VALUE

\* values between 5 and 10: 1

\* values over 10: 2

\*/

public BinByteNormalizer(final Map<Double, Byte> bins) {

Preconditions.checkNotNull(bins);

Preconditions.checkArgument(!bins.isEmpty(), "No bins provided");

Preconditions.checkArgument(hasIncreasingValues(bins));

this.bins.putAll(bins);

for (Map.Entry<Double, Byte> entry : bins.entrySet()) {

reverseBins.put(entry.getValue(), entry.getKey());

}

}

/\*\*

\* check that if key1 > key2 then val1 > val2 in the {@code map}.

\*/

private static boolean hasIncreasingValues(final Map<Double, Byte> map) {

SortedSet<Double> orderedKeys = Sets.newTreeSet(map.keySet());

byte prev = Byte.MIN\_VALUE;

for (Double key : orderedKeys) { // save the unboxing

byte cur = map.get(key);

if (cur <= prev) {

return false;

}

prev = cur;

}

return true;

}

@Override

public byte normalize(double val) {

Map.Entry<Double, Byte> lowerBound = bins.floorEntry(val);

return lowerBound == null

? Byte.MIN\_VALUE

: lowerBound.getValue();

}

@Override

public double unnormLowerBound(byte norm) {

return reverseBins.get(reverseBins.floorKey(norm));

}

@Override

public double unnormUpperBound(byte norm) {

return norm == reverseBins.lastKey()

? Double.POSITIVE\_INFINITY

: reverseBins.get(reverseBins.floorKey((byte) (1 + norm)));

}

}