```
1  /*
2   * Copyright (c) 2013 Villu Ruusmann
3   *
4   * This file is part of JPMML-Evaluator
5   *
6   * JPMML-Evaluator is free software: you can redistribute it and/or modify
7   * it under the terms of the GNU Affero General Public License as published by
8   * the Free Software Foundation, either version 3 of the License, or
9   * (at your option) any later version.
10  *
11  * JPMML-Evaluator is distributed in the hope that it will be useful,
12  * but WITHOUT ANY WARRANTY; without even the implied warranty of
13  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  * GNU Affero General Public License for more details.
15  *
16  * You should have received a copy of the GNU Affero General Public License
17  * along with JPMML-Evaluator.  If not, see <http://www.gnu.org/licenses/>.
18  */
19 package org.jpmml.evaluator;
20
21 import java.util.BitSet;
22 import java.util.List;
23
24 import org.dmg.pmml.BinarySimilarity;
25 import org.dmg.pmml.Chebychev;
26 import org.dmg.pmml.CityBlock;
27 import org.dmg.pmml.CompareFunction;
28 import org.dmg.pmml.ComparisonField;
29 import org.dmg.pmml.ComparisonMeasure;
30 import org.dmg.pmml.Distance;
31 import org.dmg.pmml.Euclidean;
```

```
32  import org.dmg.pmml.Jaccard;
33  import org.dmg.pmml.Minkowski;
34  import org.dmg.pmml.PMMLAttributes;
35  import org.dmg.pmml.Similarity;
36  import org.dmg.pmml.SimpleMatching;
37  import org.dmg.pmml.SquaredEuclidean;
38  import org.dmg.pmml.Tanimoto;
39  import org.jpmml.model.InvalidAttributeException;
40  import org.jpmml.model.InvalidElementException;
41  import org.jpmml.model.UnsupportedAttributeException;
42  import org.jpmml.model.UnsupportedElementException;
43
44  public class MeasureUtil {
45
46          private MeasureUtil(){
47          }
48
49          static
50          public <V extends Number> Value<V> evaluateSimilarity(ValueFactory<V> valueFactory, Co
51                  Similarity measure = TypeUtil.cast(Similarity.class, comparisonMeasure.require
52
53                  int a11 = 0;
54                  int a10 = 0;
55                  int a01 = 0;
56                  int a00 = 0;
57
58                  for(int i = 0, max = comparisonFields.size(); i < max; i++){
59
60                          if(flags.get(i)){
61
62                                  if(referenceFlags.get(i)){
63                                          a11 += 1;
64                                  } else
65
66                                  {
67                                          a10 += 1;
68                                  }
69                          } else
70
71                          {
72                                  if(referenceFlags.get(i)){
73                                          a01 += 1;
74                                  } else
75
76                                  {
77                                          a00 += 1;
78                                  }
79                          }
80                  }
81
82                  Value<V> numerator = valueFactory.newValue();
83                  Value<V> denominator = valueFactory.newValue();
```

```
84
85                  if(measure instanceof SimpleMatching){
86                          numerator.add(a11 + a00);
87                          denominator.add(a11 + a10 + a01 + a00);
88                  } else
89
90                  if(measure instanceof Jaccard){
91                          numerator.add(a11);
92                          denominator.add(a11 + a10 + a01);
93                  } else
94
95                  if(measure instanceof Tanimoto){
96                          numerator.add(a11 + a00);
97                          denominator
98                                  .add(a11)
99                                  .add(Numbers.DOUBLE_TWO, (a10 + a01))
100                                 .add(a00);
101                 } else
102
103                 if(measure instanceof BinarySimilarity){
104                         BinarySimilarity binarySimilarity = (BinarySimilarity)measure;
105
106                         Number c00 = binarySimilarity.requireC00Parameter();
107                         Number c01 = binarySimilarity.requireC01Parameter();
108                         Number c10 = binarySimilarity.requireC10Parameter();
109                         Number c11 = binarySimilarity.requireC11Parameter();
110
111                         numerator
112                                 .add(c11, a11)
113                                 .add(c10, a10)
114                                 .add(c01, a01)
115                                 .add(c00, a00);
116
117                         Number d00 = binarySimilarity.requireD00Parameter();
118                         Number d01 = binarySimilarity.requireD01Parameter();
119                         Number d10 = binarySimilarity.requireD10Parameter();
120                         Number d11 = binarySimilarity.requireD11Parameter();
121
122                         denominator
123                                 .add(d11, a11)
124                                 .add(d10, a10)
125                                 .add(d01, a01)
126                                 .add(d00, a00);
127                 } else
128
129                 {
130                         throw new UnsupportedElementException(measure);
131                 } // End if
132
133                 if(denominator.isZero()){
134                         throw new UndefinedResultException();
135                 }
```

```
136
137                 return numerator.divide(denominator);
138         }
139
140         static
141         public BitSet toBitSet(List<FieldValue> values){
142                 BitSet result = new BitSet(values.size());
143
144                 for(int i = 0, max = values.size(); i < max; i++){
145                         FieldValue value = values.get(i);
146
147                         if(value.equalsValue(Boolean.FALSE)){
148                                 result.set(i, false);
149                         } else
150
151                         if(value.equalsValue(Boolean.TRUE)){
152                                 result.set(i, true);
153                         } else
154
155                         {
156                                 throw new EvaluationException("Expected " + EvaluationExceptio
157                         }
158                 }
159
160                 return result;
161         }
162
163         static
164         public <V extends Number> Value<V> evaluateDistance(ValueFactory<V> valueFactory, Compa
165                 Distance measure = TypeUtil.cast(Distance.class, comparisonMeasure.requireMeas
166
167                 Number innerPower;
168                 Number outerPower;
169
170                 if(measure instanceof Euclidean){
171                         innerPower = outerPower = Numbers.DOUBLE_TWO;
172                 } else
173
174                 if(measure instanceof SquaredEuclidean){
175                         innerPower = Numbers.DOUBLE_TWO;
176                         outerPower = Numbers.DOUBLE_ONE;
177                 } else
178
179                 if(measure instanceof Chebychev || measure instanceof CityBlock){
180                         innerPower = outerPower = Numbers.DOUBLE_ONE;
181                 } else
182
183                 if(measure instanceof Minkowski){
184                         Minkowski minkowski = (Minkowski)measure;
185
186                         Number p = minkowski.requirePParameter();
187                         if(p.doubleValue() < 0d){
```

```
188                             throw new InvalidAttributeException(minkowski, PMMLAttributes.M
189                     }
190
191                     innerPower = outerPower = p;
192             } else
193
194             {
195                     throw new UnsupportedElementException(measure);
196             }
197
198             Vector<V> distances = valueFactory.newVector(0);
199
200             for(int i = 0, max = comparisonFields.size(); i < max; i++){
201                     ComparisonField<?> comparisonField = comparisonFields.get(i);
202
203                     FieldValue value = values.get(i);
204                     if(FieldValueUtil.isMissing(value)){
205                             continue;
206                     }
207
208                     FieldValue referenceValue = referenceValues.get(i);
209
210                     Value<V> distance = evaluateInnerFunction(valueFactory, comparisonMeasu
211
212                     distances.add(distance);
213             }
214
215             if(measure instanceof Euclidean || measure instanceof SquaredEuclidean || measu
216                     Value<V> result = distances.sum()
217                             .multiply(adjustment)
218                             .inversePower(outerPower);
219
220                     return result;
221             } else
222
223             if(measure instanceof Chebychev){
224                     Value<V> result = distances.max()
225                             .multiply(adjustment);
226
227                     return result;
228             } else
229
230             {
231                     throw new UnsupportedElementException(measure);
232             }
233     }
234
235     static
236     private <V extends Number> Value<V> evaluateInnerFunction(ValueFactory<V> valueFactory,
237             CompareFunction compareFunction = comparisonField.getCompareFunction();
238
239             if(compareFunction == null){
```

```
240                         compareFunction = comparisonMeasure.getCompareFunction();
241
242                         // The ComparisonMeasure element is limited to "attribute-less" compari
243                         switch(compareFunction){
244                                 case ABS_DIFF:
245                                 case DELTA:
246                                 case EQUAL:
247                                         break;
248                                 case GAUSS_SIM:
249                                 case TABLE:
250                                         throw new InvalidAttributeException(comparisonMeasure,
251                                 default:
252                                         throw new UnsupportedAttributeException(comparisonMeasu
253                         }
254                 }
255
256                 Value<V> distance;
257
258                 switch(compareFunction){
259                         case ABS_DIFF:
260                                 {
261                                         distance = valueFactory.newValue(value.asNumber())
262                                                 .subtract(referenceValue.asNumber())
263                                                 .abs();
264                                 }
265                                 break;
266                         case GAUSS_SIM:
267                                 {
268                                         Number similarityScale = comparisonField.getSimilarityS
269                                         if(similarityScale == null){
270                                                 throw new InvalidElementException(comparisonFie
271                                         }
272
273                                         distance = valueFactory.newValue(value.asNumber())
274                                                 .subtract(referenceValue.asNumber())
275                                                 .gaussSim(similarityScale);
276                                 }
277                                 break;
278                         case DELTA:
279                                 {
280                                         boolean equals = (value).equalsValue(referenceValue);
281
282                                         distance = valueFactory.newValue(equals ? Numbers.DOUBL
283                                 }
284                                 break;
285                         case EQUAL:
286                                 {
287                                         boolean equals = (value).equalsValue(referenceValue);
288
289                                         distance = valueFactory.newValue(equals ? Numbers.DOUBL
290                                 }
291                                 break;
```

```
292                         case TABLE:
293                                 throw new UnsupportedAttributeException(comparisonField, compa
294                         default:
295                                 throw new UnsupportedAttributeException(comparisonField, compa
296                 }
297
298                 distance.power(power);
299
300                 Number fieldWeight = comparisonField.getFieldWeight();
301                 if(fieldWeight != null){
302                         distance.multiply(fieldWeight);
303                 }
304
305                 return distance;
306         }
307
308         static
309         public <V extends Number> Value<V> calculateAdjustment(ValueFactory<V> valueFactory, L
310                 return calculateAdjustment(valueFactory, values, null);
311         }
312
313         static
314         public <V extends Number> Value<V> calculateAdjustment(ValueFactory<V> valueFactory, L
315                 Value<V> sum = valueFactory.newValue();
316                 Value<V> nonmissingSum = valueFactory.newValue();
317
318                 for(int i = 0, max = values.size(); i < max; i++){
319                         FieldValue value = values.get(i);
320                         Number adjustmentValue = (adjustmentValues != null ? adjustmentValues.
321
322                         sum.add(adjustmentValue);
323
324                         if(!FieldValueUtil.isMissing(value)){
325                                 nonmissingSum.add(adjustmentValue);
326                         }
327                 }
328
329                 if(nonmissingSum.isZero()){
330                         throw new UndefinedResultException();
331                 }
332
333                 return sum.divide(nonmissingSum);
334         }
335 }
```