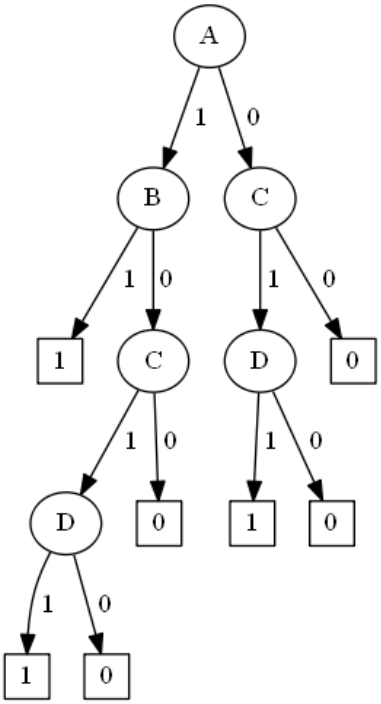
Machine Learning
Blatt 1

Markus Vieth, David Klopp, Christian Stricker

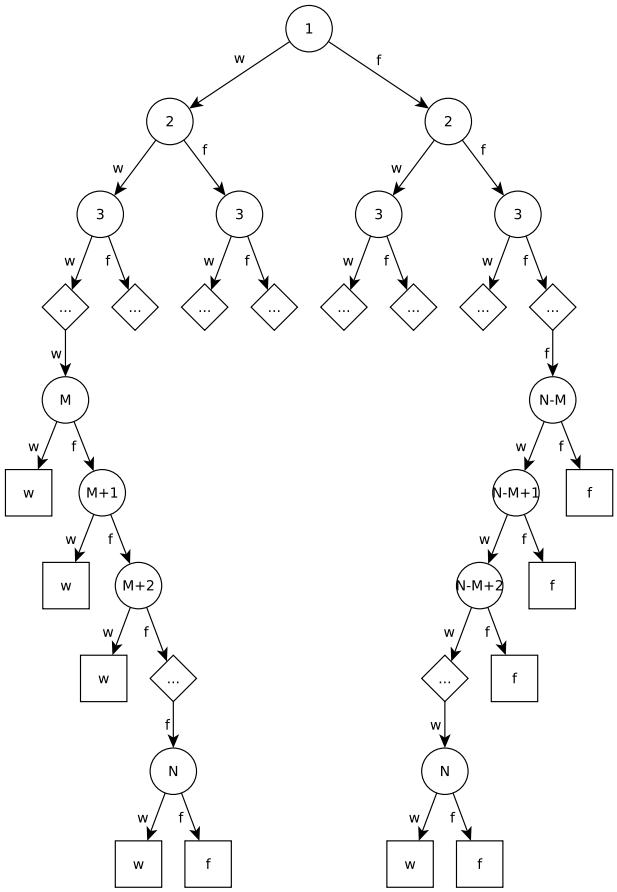
3. Mai 2016

Nr.1

a)



b)



Skizze des M-von-N-Baumes. Die Teilbäume der Knoten M bis N und N-M bis N finden sich (teilweise verkürzt) in der Mitte des Baumes wieder. Der Baum hat eine maximale Tiefe von N und eine maximale Breite von 2^{M+1} im Level $M+1$
Leichter ist es den Baum formal zu beschreiben (wie ein Programm). So bleibt einiges offen. -0,5

Nr.3

a)

$S = \{A, B, C, D\}$

$$H(S) = -p(A)log2(p(A)) - p(B)log2(p(B)) - p(C)log2(p(C)) - p(D)log2(p(D))$$

$$= -0,5 * log2(0,5) - 0,3 * log2(0,3) - 0,1 * log2(0,1) - 0,1 * log2(0,1)$$

$$= 1.685475297227334319499038031560350135304471374204545987890...$$

b)

$max = log2(|S|) \quad , \forall x \in S \quad$ gilt: $\quad p(x) = 1/|S|$
$min = 0 \Leftrightarrow \exists x \in S \quad$ mit $\quad p(x) = 1 \Rightarrow \forall y \in S \setminus \{x\} : p(y) = 0$

c)

Entropie ist eine Skala für die Verteilung der Werte des Klassenattributes auf die Werte des betrachteten Attributes. Somit bedeutet eine niedrige Entropie, dass sich die Werte des Klassenattributes auf einige wenige Werte des betrachteten Attributes verteilen, wo hingegen eine hohe Entropie bedeutet, dass sich die Werte des Klassenattributes gleichmäßig auf die Werte des betrachteten Attributes verteilen.

Kodierung

Symbol	Code-Wort
A	0
B	10
C	110
D	111

d)

Bei einem Entscheidungsbaum ist die Reihenfolge der Betrachtung der Attribute für dessen Komplexität von Bedeutung. Die Qualität *naja* eines Attributes kann mithilfe des InformationGains bewertet werden. Dazu betrachten wir die erwartete Verminderung der aktuellen Entropie bei Wahl des betrachteten Attributes. Durch eine niedrige erwartete Entropie wird der Baum kürzer. *Wie wird das gemacht? -1,5*

Nr.5

Anmerkung Es wurden im Prinzip die Klassen aus der bereitgestellten .jar verwendet. Es wurden jedoch 2 Änderungen vorgenommen.

1. Ein Dataset kennt seinen ClassIndex
2. Ein Attribute weiß, welchen Typ es hat

Code

```
1  import org.kramerlab.teaching.ml.datasets.*;
2
3  import java.io.File;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  //TODO exceptions
8  /**
9   * Created by Markus Vieth on 21.04.2016.
10  */
11 public class DecisionTree {
12
13     /**
14      * Inner Node class
15      */
16     private class Node {
17         Attribute attribute = null;
18         List<Edge> edges = new ArrayList<>();
19         List<Integer> indices;
20         /**
21          * Constructor
22          * @param indices
23          */
24         public Node(List<Integer> indices) {
25             this.indices = indices;
26         }
27
28         public void addEdge(Edge edge) {
29             this.edges.add(edge);
30         }
31     }
32
33
34     /**
35      * Edge class
36      */
37     private class Edge {
38         Value value = null;
39         Node start;
40         Node end;
41
42         /**
43          * Constructor
44          * @param value
45          */
46         public Edge(Value value, Node start) {
47             this.value = value;
48             this.start = start;
49             this.start.addEdge(this);
50         }
51     }
52
53
54
55
56
57     private Instance[] data;
58     private Dataset dataset;
```

```
60      /**
61       * default constructor
62       */
63      public DecisionTree() {
64      }

66      /**
67       * loads arff
68       * @param path path to arff
69       */
70      public DecisionTree(String path) {
71          try {
72              this.loadArff(path);
73          } catch (Exception e) {
74              e.printStackTrace();
75          }
76      }

78      /**
79       * loads arff
80       * @param file arff file
81       */
82      public DecisionTree(File file) {
83          try {
84              this.loadArff(file);
85          } catch (Exception e) {
86              e.printStackTrace();
87          }
88      }

90      /**
91       * loads arff
92       * @param path path to arff
93       * @throws Exception see kramerlabs dataset
94       */
95      public void loadArff(String path) throws Exception {
96          File file = new File(path);
97          this.loadArff(file);
98      }

100     /**
101      * loads arff
102      * @param file arff file
103      * @throws Exception see kramerlabs dataset
104      */
105     public void loadArff(File file) throws Exception {
106         this.dataset = new Dataset();
107         dataset.load(file);
108         this.data = new Instance[dataset.getNumberOfInstances()];

110         for (int i = 0; i < data.length; i++) {
111             this.data[i] = dataset.getInstance(i);
112         }
113     }


116     // Implement a method informationGain that takes two arguments: attribute A
117     // and a list of indices i 1 , i 2 , i m .
118     /**
119      * calculates information gain for given attribute and instances
120      * @param attribute given attribute
```

```
121      * @param indices given indices of instances
122      * @return information gain
123      */
124     public double informationGain(Attribute attribute, List<Integer> indices) {
125         List<Attribute> attributes = this.dataset.getAttributes();
126         Attribute classAttr = attributes.get(this.dataset.getClassIndex());
127
128         //TODO throw Exception
129         // Check if nominal
130         if (!attribute.isNominal()) {
131             System.err.println(attribute.getName() + "is not nominal");
132             return Double.NaN;
133         } else if (!classAttr.isNominal()) {
134             System.err.println(classAttr.getName() + "is not nominal");
135             return Double.NaN;
136         }
137
138         NominalAttribute attr = (NominalAttribute) attribute;
139         // Init gain
140         double gain = calculateEntropy((NominalAttribute)classAttr,
141                 indices);
142
143         // sum over all values in attribute
144         for (int v = 0; v < attr.getNumberOfValues(); v++) {
145             List<Integer> subIndices = new ArrayList<>();
146             // Alternative we could copy data in a list and remove already
147             // picked instances to improve runtime
148             // creates subset with indices of instances with value v
149             for (int i : indices) {
150                 Instance instance = data[i];
151                 NominalValue value = (NominalValue)instance.getValue(attr);
152                 if (attr.getValue(v).equals(value)) {
153                     subIndices.add(i);
154                 }
155             }
156
157             // calculates entropy
158             double entropy = calculateEntropy((NominalAttribute)classAttr,
159                     subIndices);
160
161             // see formula
162             gain -= entropy * ((double)subIndices.size())
163                     /((double)indices.size());
164         }
165
166         return gain;
167     }
168
169
170     /**
171      * calculates entropy
172      * @param classAttr given class attribute
173      * @param indices indices of instances
174      * @return entropy
175      */
176     private double calculateEntropy(NominalAttribute classAttr, List<Integer>
177             indices
178     ) {
179         int[] values = new int[classAttr.getNumberOfValues()];
180         // calculates number of instances with value v in class attribute
181         for (int v = 0; v < classAttr.getNumberOfValues(); v++) {
```

```
182             values[v] = 0;
183             for(int i : indices) {
184                 Instance instance = data[i];
185                 NominalValue value = (NominalValue)instance.getValue(classAttr);
186                 NominalValue classValue = classAttr.getValue(v);
187                 if (classValue.equals(value)) {
188                     values[v]++;
189                 }
190             }
191         }
192         return calculateEntropy(values);
193     }

195     /**
196      * calculates entropy
197      * @param values given values
198      * @return entropy
199      */
200     private double calculateEntropy(int[] values) {
201         double sum = 0;
202         for (int i : values) {
203             sum += i;
204         }
205         double entropy = 0.0;

207         for (int value : values) {
208             double p = value/sum;
209             entropy -= p * log2(p);
210         }

212         return entropy;
213     }

215     public Attribute selectAttribute(Node node) {
216         Attribute select = null;
217         double maxGain = -1.0;
218         for (Attribute attribute : dataset.getAttributes()) {
219             double gain = this.informationGain(attribute, node.indices);
220             if (gain > maxGain) {
221                 select = attribute;
222             }
223         }

225         return select;
226     }

228     /**
229      * calculates log to base 2
230      * @param a given parameter
231      * @return log2(a) or 0 if a == 0
232      */
233     private double log2(double a) {
234         if ( Double.compare(0.0, Math.abs(a)) == 0 )
235             return 0;
236         return Math.log(a) / Math.log(2);
237     }


240     /**
241      * prints some test data
242      */
```

```
243     private void testPrint() {
244         List<Integer> indices = new ArrayList<>();
245         for (int i = 0; i < data.length; i++) {
246             indices.add(i);
247         }
248         for (Attribute attr : this.dataset.getAttributes()) {
249             System.out.print("Attribute " + attr.getName());
250             System.out.print(" has an InformationGain of " + informationGain
251                     (attr, indices));
252             System.out.println();
253         }
254     }

256     /**
257      * a test
258      * @param args none
259      */
260     public static void main(String[] args) {
261         DecisionTree dt = new DecisionTree("res/weather.nominal.arff");
262         dt.testPrint();
263     }
264 }
```