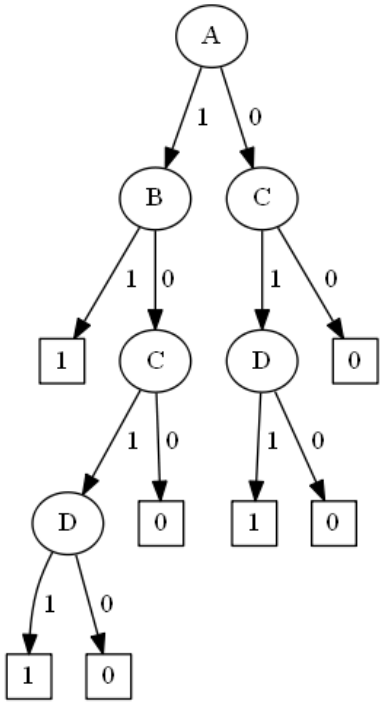
Machine Learning
Blatt 1

Markus Vieth, David Klopp, Christian Stricker

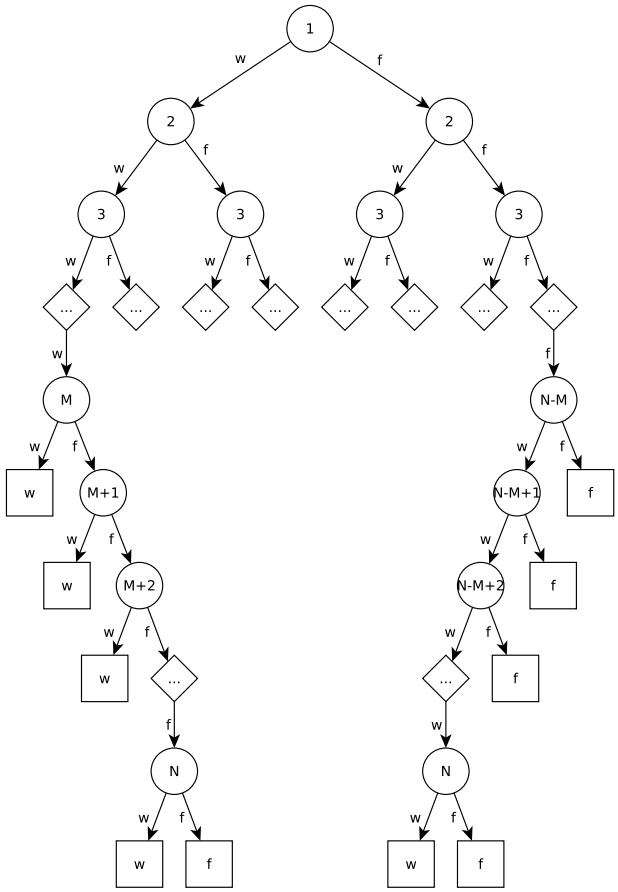
27. April 2016

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 $M + 1$

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 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.

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     private Instance[] data;
14     private Dataset dataset;
15
16     /**
17      * default constructor
18      */
19     public DecisionTree() {
20     }
21
22     /**
23      * loads arff
24      * @param path path to arff
25      */
26     public DecisionTree(String path) {
27         try {
28             this.loadArff(path);
29         } catch (Exception e) {
30             e.printStackTrace();
31         }
32     }
33
34     /**
35      * loads arff
36      * @param file arff file
37      */
38     public DecisionTree(File file) {
39         try {
40             this.loadArff(file);
41         } catch (Exception e) {
42             e.printStackTrace();
43         }
44     }
45
46     /**
47      * loads arff
48      * @param path path to arff
49      * @throws Exception see kramerlabs dataset
50      */
51     public void loadArff(String path) throws Exception {
52         File file = new File(path);
53         this.loadArff(file);
54     }
55
56     /**
57      * loads arff
58      * @param file arff file
59      * @throws Exception see kramerlabs dataset
```

```
60      */
61     public void loadArff(File file) throws Exception {
62         this.dataset = new Dataset();
63         dataset.load(file);
64         this.data = new Instance[dataset.getNumberOfInstances()];
65
66         for (int i = 0; i < data.length; i++) {
67             this.data[i] = dataset.getInstance(i);
68         }
69     }
70
71
72     // Implement a method informationGain that takes two arguments: attribute A
73     // and a list of indices i 1 , i 2 , i m .
74     /**
75      * calculates information gain for given attribute and instances
76      * @param attribute given attribute
77      * @param indices given indices of instances
78      * @return information gain
79      */
80     public double informationGain(Attribute attribute, List<Integer> indices) {
81         List<Attribute> attributes = this.dataset.getAttributes();
82         Attribute classAttr = attributes.get(this.dataset.getClassIndex());
83
84         //TODO throw Exception
85         // Check if nominal
86         if (!attribute.isNominal()) {
87             System.err.println(attribute.getName() + "is not nominal");
88             return Double.NaN;
89         } else if (!classAttr.isNominal()) {
90             System.err.println(classAttr.getName() + "is not nominal");
91             return Double.NaN;
92         }
93
94         NominalAttribute attr = (NominalAttribute) attribute;
95         // Init gain
96         double gain = calculateEntropy((NominalAttribute)classAttr,
97                 indices);
98
99         // sum over all values in attribute
100        for (int v = 0; v < attr.getNumberOfValues(); v++) {
101            List<Integer> subIndices = new ArrayList<>();
102            // Alternative we could copy data in a list and remove already
103            // picked instances to improve runtime
104            // creates subset with indices of instances with value v
105            for (int i : indices) {
106                Instance instance = data[i];
107                NominalValue value = (NominalValue)instance.getValue(attr);
108                if (attr.getValue(v).equals(value)) {
109                    subIndices.add(i);
110                }
111            }
112
113            // calculates entropy
114            double entropy = calculateEntropy((NominalAttribute)classAttr,
115                    subIndices);
116
117            // see formula
118            gain -= entropy * ((double)subIndices.size())
119                    /((double)indices.size());
120        }
```

```
122         return gain;
123     }


126     /**
127      * calculates entropy
128      * @param classAttr given class attribute
129      * @param indices indices of instances
130      * @return entropy
131      */
132     private double calculateEntropy(NominalAttribute classAttr, List<Integer>
133             indices
134     ) {
135         int[] values = new int[classAttr.getNumberOfValues()];
136         // calculates number of instances with value v in class attribute
137         for (int v = 0; v < classAttr.getNumberOfValues(); v++) {
138             values[v] = 0;
139             for(int i : indices) {
140                 Instance instance = data[i];
141                 NominalValue value = (NominalValue)instance.getValue(classAttr);
142                 NominalValue classValue = classAttr.getValue(v);
143                 if (classValue.equals(value)) {
144                     values[v]++;
145                 }
146             }
147         }
148         return calculateEntropy(values);
149     }

151     /**
152      * calculates entropy
153      * @param values given values
154      * @return entropy
155      */
156     private double calculateEntropy(int[] values) {
157         double sum = 0;
158         for (int i : values) {
159             sum += i;
160         }
161         double entropy = 0.0;

163         for (int value : values) {
164             double p = value/sum;
165             entropy -= p * log2(p);
166         }

168         return entropy;
169     }

171     /**
172      * calculates log to base 2
173      * @param a given parameter
174      * @return log2(a) or 0 if a == 0
175      */
176     private double log2(double a) {
177         if ( Double.compare(0.0, Math.abs(a)) == 0 )
178             return 0;
179         return Math.log(a) / Math.log(2);
180     }
```

```
183     /**
184      * prints some test data
185      */
186     private void testPrint() {
187         List<Integer> indices = new ArrayList<>();
188         for (int i = 0; i < data.length; i++) {
189             indices.add(i);
190         }
191         for (Attribute attr : this.dataset.getAttributes()) {
192             System.out.print("Attribute " + attr.getName());
193             System.out.print(" has an InformationGain of " + informationGain
194                     (attr, indices));
195             System.out.println();
196         }
197     }

199     /**
200      * a test
201      * @param args none
202      */
203     public static void main(String[] args) {
204         DecisionTree dt = new DecisionTree("res/weather.nominal.arff");
205         dt.testPrint();
206     }
207 }
```