

Mustererkennung: Übungsblatt 10

von

Naja v. Schmude (4127652), Lisa Dohrmann (4130066), Adrian Neumann (4140810)

Aufgabe 1

Wir sollten mit Hilfe des ID3 Algorithmus aus den verlinkten Wetterdaten einen Entscheidungsbaum bauen, der uns ermöglicht festzustellen, ob wir bei bestimmten Wetterbedingungen ein mysteriöses Spiel spielen können.

Wir haben uns also erstmal eine Klasse für Bäume mit beschrifteten Knoten und Kanten geschrieben. An den Kanten hängen wieder Bäume. Sie haben ein extra Feld, das nur gefüllt wird, wenn sie ein Blatt sind (stattdessen hätte man wohl auch eine extre Klasse nehmen können). Die Klasse ist nicht weiter spannend, ihre einzige Besonderheit ist ihre Fähigkeit sich in DOT-Graphenbeschreibungssprache zu rendern:

```
public String toString() {
    String thisNodeId = getUniqueId();
    String s = "graph TD\n"
        + "nodesep=0.7;\n"
        + "label=\\\"\n"
        + name
        + "\\\";\n"
        + thisNodeId + "[label=\\\" + content + "\\\"];\n"
        + toStringHelp(thisNodeId) + "\\n}";
    return s;
}

/*
 * Damit nur die Wurzel noch das drumherum des DOT graphen produziert wird
 * die Funktion hier für die Rekursion benutzt.
 */
private String toStringHelp(String thisNodeId) {
    StringBuilder b = new StringBuilder();
    for (C key : children.keySet()) {
        Tree<A, B, C> c = children.get(key);
        String nodeId = getUniqueId(); //dem Kind wird eine Zahl zugeordnet

        //erst den Knoten dazu
        if (c.isLeaf()) //Blätter bekommen eine besondere Form
            b.append(nodeId + "[shape=diamond,label=\\\" + c.leafContent.toString() + "\\\"];\n");
        else
            b.append(nodeId + "[label=\\\" + c.content + "\\\"];\n");

        //und noch die Kante dazu
        b.append(thisNodeId + "--" + nodeId + "\n");
        b.append(String.format("[label=\\\"%s\\\";\n", key.toString()));

        //dann den Restbaum unter diesem Kind
        b.append(c.toStringHelp(nodeId));
    }
    return b.toString();
}
```

Von dieser Klasse erbt der DecisionTree. In seinem Konstruktor wird der ID3 Algorithmus ausgeführt:

```
private DecisionTree(List<Datum> data) {
    // was für Attribute haben wir überhaupt?
    attributes = Utils.extractAttributes(data);
}
```

```

// welches ist das coolste im ganzen Land?
Attribute best = Utils.maxEntropy(attributes, data);
System.out.println("The best attribute is " + best);
content = best; //Dieser Knoten wird damit beschriftet

// oh wir sind wohl fertig?
// dann machen wir noch ein Blatt dran
// das uns die Klasse sagt
if (content == null) {
    System.out
        .println("Kein bestes mehr gefunden, alle scheinen " +
            "Entropiegewinn 0 zu haben");
    //indem wir leafContent füllen, machen wir uns zum Blatt
    leafContent = getDominantLabel(Utils.getLabels(data));
    return;
}

//wird lazy erstellt, wenn wir ein Blatt machen wollen
HashMap<String, List<Datum>> labelPartition = null;
String dominantLabel = null;

// mit dem coolen Attribut die Menge teilen
LinkedList<Tupel<String, LinkedList<Datum>>> partitions = Utils
    .partition(content, data);

//und für jede Teilmenge ein neues Kind erzeugen.
for (Tupel<String, LinkedList<Datum>> p : partitions) {
    if (!p.b.isEmpty())
        this.addChild(new DecisionTree(p.b), p.a);
    else { // da war nix drin?
        // da machen wir ein Blatt mit der dominanten Klasse dieses
        // Knotens
        // das ist ein bisschen fragwürdig, weil wir eigentlich keine
        // Daten haben. Aber was soll's
        if (labelPartition == null) { // lazy labelpartitionen erstellen
            labelPartition = Utils.getLabels(Utils.selectWhere(best,
                data));
            dominantLabel = getDominantLabel(labelPartition);
        }
        addLeaf(dominantLabel); // und ein blatt dazumachen
    }
}
}

```

Zum Entscheiden laufen wir einfach den Baum lang, bis wir an ein Blatt kommen. Die Kanten sind ja mit den möglichen Werten des Attributs beschriftet für das in diesem Knoten getrennt wird.

```

public String decide(Datum d) {
    if (isLeaf())
        return leafContent;
    //wir holen das Attribut aus dem Datum
    Attribute a = d.getAttribute(content.name);
    //und laufen in das Kind, das an der entsprechenden Kante hängt
    return getChild(a.value).decide(d);
}

```

Die Entropie eines Sets mit Daten berechnen wir so:

```

static double entropy(List<Datum> set) {
    HashMap<String, List<Datum>> labels = getLabels(set);
    double ent = 0;
    //Wie in der Wikipedia: Für jedes Label i die
    //Wahrscheinlichkeit p_i bestimmen
    //und mit dem Logarithmus verwursten
    for (String i : labels.keySet()) {
        double pi = (float) labels.get(i).size() / (float) set.size();
        ent += pi * Math.log(pi) / Math.log(2);
    }
    return -ent;
}

```

Den Informationsgewinn eines Attributs bestimmen wir mit dieser Funktion:

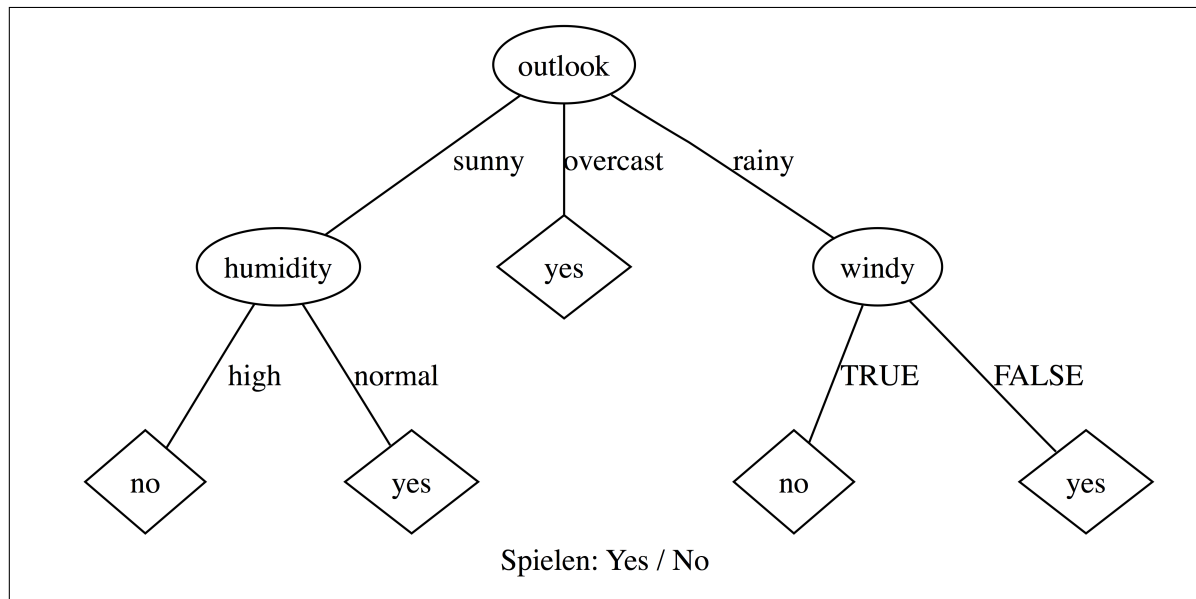
```

private static double entropy(Attribute a, List<Datum> set) {
    //Die Menge mit Hilfe des Attributs zerlegen
    LinkedList<Tupel<String, LinkedList<Datum>>> partitions = partition(a,
        set);
    double ent = 0;
    //Für alle Partitionen die gewichtete Entropie bestimmen
    for (Tupel<String, LinkedList<Datum>> p : partitions)
        ent += entropy(p.b) * ((float) p.b.size() / (float) set.size());

    //und die Summe von der Entropie des Ausgangsdaten abziehen
    return entropy(set) - ent;
}

```

Wenn man einen Baum für den Testdatensatz erstellt und mit Graphviz rendert, sieht er so aus



Er ist überraschend klein, trennt aber die Testdaten perfekt (bei der geringen Anzahl der Daten ist das wohl kaum verwunderlich). Das Spiel wird aber dadurch nicht weniger mysteriös. Man spielt es, wenn es regnet (solange es nicht windig ist), aber nicht, wenn es sonnig ist und die Luftfeuchtigkeit hoch.