**Dokumentation zu den Programmen in Java  
Data Mining mit Entscheidungsbäumen**

Inhaltsverzeichnis

[1. Einordnung des Themas 2](#_Toc74157386)

[2. Kurze Zusammenfassung der Dokumentation 2](#_Toc74157387)

[3. Bearbeitung der Aufgaben 3](#_Toc74157388)

[Aufgabe 2.1: Vorverarbeitung der Daten 3](#_Toc74157389)

[1.1 CSVReader 3](#_Toc74157390)

[1.2 BinningDiscretizer 3](#_Toc74157391)

[1.3 Categorizer 3](#_Toc74157392)

[1.4 Categoric 3](#_Toc74157393)

[1.5 Continuously 3](#_Toc74157394)

[Aufgabe 2.2: Entscheidungsbaum lernen mit ID3 4](#_Toc74157395)

[2.1 ID3Utils 4](#_Toc74157396)

[2.2 DecisionTreeNode 4](#_Toc74157397)

[2.3 EntropyUtils 4](#_Toc74157398)

[Aufgabe 2.3: Reduced Error Pruning 5](#_Toc74157399)

[3.1 ReducedErrorPruner 5](#_Toc74157400)

[3.2 CrossValidator 5](#_Toc74157401)

[Aufgabe 2.4: Ausgabe des Entscheidungsbaums 6](#_Toc74157402)

[4.1 XMLWriter 6](#_Toc74157403)

[Aufgabe 2.5: Rapidminer 7](#_Toc74157404)

[5 Kurze Reflexion der Problematiken und Umsetzung 8](#_Toc74157405)

[6 Design Entscheidungen 8](#_Toc74157406)

# Einordnung des Themas

Dieses Programm ist in der Lage ein eine CSV-Datei einzulesen und auf Grundlage der eigelesenen Daten einen Entscheidungsbaum zu generieren. Hierbei wird der ID3 Algorithmus aus der Vorlesung genutzt, der durch Rekursion den Baum rekursiv generiert. Anschließend kann man den Baum als XML-Datei ausgeben lassen oder den Baum zur weiteren Informationsverarbeitung nutzen… NOT FINAL

# Kurze Zusammenfassung der Dokumentation

# Bearbeitung der Aufgaben

## Aufgabe 2.1: Vorverarbeitung der Daten

### CSVReader

Die CSVReader Klasse wir zur einlesen von Daten aus einer CSV-Datei genutzt. Hierbei liest die Klasse über readCsvToArray und einen Buffered Reader Zeilenweise ein uns zerlegt jede Zeile in ihre einzelnen Attribute. Nach der Zerlegung werden die Daten in einer Liste zusammengefasst im Return zurückgegeben. Hierbei wird noch geguckt, ob wir den ersten Eintrag ausschließen sollen oder nicht, dies hängt davon ab, ob es sich hierbei um den Header handelt.

### BinningDiscretizer

### Categorizer

### Categoric

### Continuously

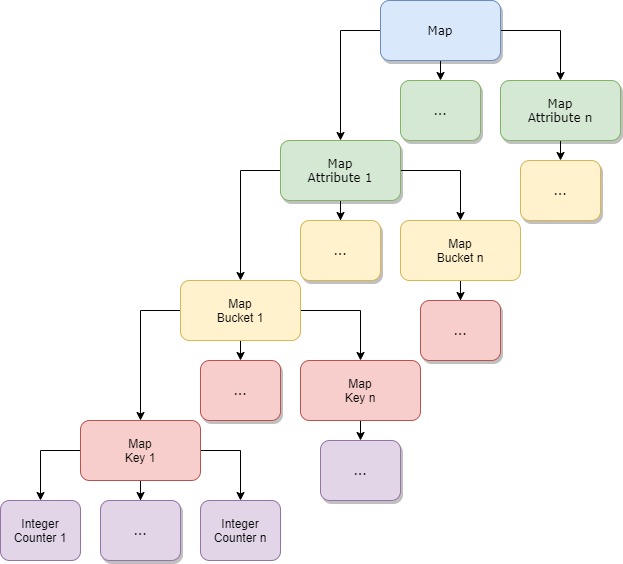
## Aufgabe 2.2: Entscheidungsbaum lernen mit ID3

### ID3Utils

### DecisionTreeNode

### EntropyUtils

Die Klasse EntropyUtils wird genutzt, um die Entropie zu jedem Attribut zu berechnen. Hierbei wurde eine Baumstruktur genutzt, um die Verteilung der Attribute in Relation zu den Schlüsseln zu berechnen.

Dabei handelt es sich um eine Verschachtlung von Hash Maps. Zur vereinfachten Visualisierung wurden die Ebenen des Baums eingefärbt.

(Blaue) verweist auf alle möglichen Attribute,

(Grün) verweist auf alle möglichen Ausprägungen zu dem jeweiligen Attribut,

(Gelb) verweist auf alle möglichen Schlüsselattribute,

(Rot) verweist auf den Zähler des jeweiligen Schlüssels,

(Violet) zählt wievielte Instanzen zu der gegebenen Ausprägung passen.

Haben wir alle Ausprägungen gezählt, dann können wir die Verteilungen an die jeweiligen Funktionen übergeben. Dabei wird zuerst H(E) für die Schlüssel berechnen, indem wir die Schlüsselmap von Index an HE() übergeben, dieser wird uns ein Double als startwert zurückgeben. Anschließend berechnen wir H(E) für alle übrigen Schlüsselmaps, dafür übergeben wir die restlichen Schlüsselmaps an H(), dieser wird uns zwar auch H(E) berechnen aber uns sofort R(E) zurückliefern. Dieser muss schlussendlich dann jeweils noch von unserem Startwert abgezogen werden. Wiederholen wir das für alle Attribute so erhalten wir zu, jedem Attribut ein Gain().

## Aufgabe 2.3: Reduced Error Pruning

### ReducedErrorPruner

### CrossValidator

## Aufgabe 2.4: Ausgabe des Entscheidungsbaums

### XMLWriter

## Aufgabe 2.5: Rapidminer

# Kurze Reflexion der Problematiken und Umsetzung

Auf Grundlage der Reorganisation hatten wir anfängliche Probleme mit der Organisation sowie der Synchronisation der Daten. Dieses Problem wurde dann schlussendlich von Herrn Moll, welcher die Nutzung von GitHub vorschlug, gelöst. Dies erlaubte es uns dann schnell und synchron and den Aufgaben im Team zu arbeiten.

Nach den anfänglichen Synchronisation Probleme, haben wir öfters Backtracking betreiben müssen, weil wir Fehler in Programmen hatten, die in späteren Iterationen komplett umgeschrieben werden mussten, diese Fehlerbehebung hat uns schlussendlich viel Zeit gekostet, uns aber auch die Möglichkeit gegeben uns näher mit verschieden Prozessen zu beschäftigen und diese zu verinnerlichen.

Es wurde versucht das Programm so effizient wie möglich zu haten, hierbei wurden Bibliotheken genutzt, die in Relation der Aufgaben am besten geeignet waren. Wir konnten somit effizient die Daten durch ein Buffer Reader einlesen. Der Umgang mit großen Daten wurde durch Stream abgewickelt, die es erlaubten sehr schnell über die Daten zu gehen.

# Design Entscheidungen