

## Todo list

## Contents

# 1 Einleitung

## Struktur vom Dokument erläutern

### 1.1 Begriffe

**Definition:** Eigenes, Gesamtes

$M$  sei eine Menge von Elementen, die in zwei Teilmengen  $M_A$  und  $M_B$  zerfällt, sodass  $M = M_A \cup M_B$  ist. Wir nehmen desweiteren an, dass Alice  $M_A$  kennt, aber weder  $M$  noch  $M_B$  und dass Bob  $M_B$  kennt, aber weder  $M$  noch  $M_A$ . Dann bezeichnen wir:

- $M$  als **gesamtes** Wissen
- $M_A$  als das **eigene** Wissen von Alice
- $M_B$  als das **eigene** Wissen von Bob
- $M_B$  als das **andere** Wissen von Alice
- $M_A$  als das **andere** Wissen von Bob

**Definition:** Gemeinsam

Wenn beide Anwender das gleiche Wissen  $w$  haben, dann bezeichnen wir  $w$  als **gemeinsames Wissen**.

**Definition:** Vorwissen

Wenn Anwender verschiedene Phasen hintereinander ausführen, dann bezeichnen wir das Wissen aus den bereits ausgeführten Phasen als **Vorwissen**.

## Definition: Entscheidungsbaum

**Definition:** Attribut

Wir definieren eine Menge von Wahrscheinlichkeiten  $P = [0, 1] \subset \mathbb{R}$  und eine Menge von Buchstaben  $\Sigma = \{a, b, \dots, z, A, B, \dots, Z\}$ . Damit definieren wir ein **Attribut** als  $\Sigma^+ \times P \times P$ . Wenn ein Attribut  $A = (w, l, h)$  gegeben ist, bezeichnen wir  $w$  als **Wort**,  $l$  als **unterer Schwellwert** und  $h$  als **oberer Schwellwert**. Es wird desweiteren von allen Attributen gefordert, dass  $l \leq h$  ist.

**Definition:** Wahrheitstafel

Für eine boolesche Funktion ist eine **Wahrheitstafel** eine Tabelle, die für jede Kombination der Eingaben genau eine Ausgabe der beschriebenen Funktion definiert.

**Definition:** entstellte Wahrheitstafel

Gegeben sei eine Wahrheitstafel mit Eingabevariablen  $\{i_1, i_2, \dots, i_n\}$  und Ausgabevariable  $o$  und wir stellen für jede Variable  $v$  eine bijektive Verschleierung  $f_v : 0, 1 \mapsto \mathbb{Z}$  auf. Dann entsteht eine entstellte Wahrheitstafel für  $W$ , wenn wir für jede Zeile in der Wahrheitstafel die verschleierte Ausgabe mittels einer symmetrischen Verschlüsselung mit den verschleierten Werte der Eingangsvariablen in einem Verschlüsselungsschritt pro Eingangsvariable verschlüsseln. Es

ist notwendig zu bemerken, dass eine Implementierung einen Weg anbieten muss, eine sinnlose Verschlüsselung zu erkennen.

Wenn also beispielsweise in einer Wahrheitstafel die Eingaben 1 und 1 auf 1 abgebildet werden und die erste Eingabevariable verschleiert 1 als 23, die zweite Eingabevariable verschleiert 1 als 49 und die Ausgabevariable verschleiert 1 als 12 und  $E$  ist die symmetrische Verschlüsselung, dann ist das daraus entstehende Element der erstellten Wahrheitstafel  $E_{f_{i_1}(1)}(E_{f_{i_2}(1)}(f_o(1))) = E_{49}(E_{23}(12))$

**Definition:** Schaltkreis

Wir definieren einen **Schaltkreis** als einen gerichteten azyklischen Graphen. Die Knoten dieses Graphen sind annotiert mit Wahrheitstafeln. Die Kanten sind eingeteilt in **Eingabekanten**, **innere Kanten** und **Ausgangskanten**. Der Wert einer Eingabekante wird vom Anwender zu Beginn festgelegt. Der Wert einer inneren Kante oder einer Ausgangskante ergibt sich durch anwenden der Wahrheitstafel auf die Eingangskanten. Da wir nur boolsche Schaltkreise mit den Knotenannotationen „and“, „or“, „xor“ und „not“ brauchen und all diese Funktionen entweder unär oder kommutativ sind, brauchen wir keine Ordnung der Eingabekanten festlegen.

**Definition:** Entstellter Schaltkreis

Ein **entstellter Schaltkreis** entsteht aus einem normalen Schaltkreis, indem jede an einen Knoten annotierte Wahrheitstafel entstellt wird, wenn dabei garantiert wird, dass für die Verbindung einer Ausgangsvariablen  $o$  mit einer Eingangsvariablen  $i$  auf beide Variablen die gleiche Verschleierung angewendet wird.

Desweiteren wird für jede Eingabekante die Verschleierungsfunktion gespeichert, um die Eingabe des Schaltkreises kodieren zu können und für jede Ausgangskante die Inversion der Verschleierungsfunktion gespeichert, um die Ausgabe dekodieren zu können.

## 1.2 Annahmen

Annahme: ehrliche anwender := "handeln nach protokoll"

## 2 Grundlagen der Anwendung

Vision der Anwendung

### 2.1 Form der Benutzereingabe

Festlegen, wie die E-Mails ins Programm kommen

### 2.2 Interaktion der verteilten Programme

Festlegen, wie das Programm verteilt wird und die Teile kommunizieren

### 2.3 Phasen der Anwendung

Kurz die Einzelnen phasen der Anwendung beschreiben

### 3 Finden der gemeinsamen Wortliste

In dieser Phase berechnen die beiden Parteien aus den gesamten E-Mails eine gemeinsame Wortliste, die mit grosser Wahrscheinlichkeit aussagekräftige Attribute für den Entscheidungsbaum liefert. Die Phase besteht aus zwei Schritten. Im ersten Schritt berechnen beide Parteien getrennt eine eigene Wortliste. Jedes Wort auf dieser Liste hat in den eigenen E-Mails eine starke Aussagekraft über die Klassifikation der E-Mails, die das Wort oft enthalten. Im zweiten Schritt vereinen beide Parteien ihre eigenen Wortlisten, um die gemeinsame Wortliste zu berechnen. Dieser Vorgang ist in Abbildung ?? illustriert.

#### 3.1 Berechnung der Anteile an den Wortmengen

Wir verwenden eine Heuristik für den Informationsgehalt des Vorkommens eines Wortes in einer E-Mail, die auf dem Verhältnis der Vorkommnisse des Wortes zu der Gesamtanzahl Worte in einer Klasse basiert. Um diese zu berechnen werden konzeptionell alle Worte in E-Mails einer Klasse zu einer Multimenge hinzugefügt. Für jedes Wort in dieser Multimenge ist das Verhältnis der Vielfachheit des Wortes zur Mächtigkeit der Multimenge das gesuchte Verhältnis. Damit ergibt sich folgendes Akzeptanzkriterium:

**Requirement 1:** Wenn die Spam-Mails die Mails "Foo Bar", "Foo Bar", "Foo Foo" und "Foo" sind und die Nicht-Spam-Mails "Bar Bar", "Foo" und "Bar", dann muss diese Phase die folgende Tabelle berechnen:

Wort	Spam-Anteil	Nicht-Spam-Anteil
Foo	$\frac{5}{7}$	$\frac{1}{4}$
Bar	$\frac{2}{7}$	$\frac{3}{4}$

Eine Approximation der Werte durch Fließkommazahlen ist ebenfalls akzeptabel.

#### 3.2 Auswahl der Worte nach Informationsheuristik

Wir wollen nun Worte auswählen, deren häufiges Vorkommen in einer E-Mail viel über die Klasse der E-Mail aussagen. Wir verwenden dabei eine deduktive Heuristik, die annimmt, dass die Wahrscheinlichkeit, dass eine E-Mail zu einer Klasse gehört, direkt mit den Wahrscheinlichkeiten zusammenhängt, dass die Wörter in dieser E-Mail zu dieser Klasse gehören. Das bedeutet, wir wollen Worte selektieren, bei denen genau eine Wahrscheinlichkeit, zu einer Klasse zu gehören, drastisch unterschiedlich ist.

Eine mögliche Quantifizierung dieser Unterschiedlichkeit ist im binären Fall  $\Delta(x, y) = \|x - y\|$ . Diese Quantifizierung hat die Eigenschaft, für  $x = 1$  und  $y = 0$  bzw  $x = 0$  und  $y = 1$  maximal 1 zu sein, und für identische  $x$  und  $y$  0 zu sein. Da  $x$  und  $y$  für die Belegung (0, 1) bzw (1, 0) wirklich

maximal weit auseinander liegen und bei gleichen Werten wirklich die gerinstmögliche Aussage über die Klassezugehörigkeit eines Wortes getroffen wird, ist dies wirklich eine Quantifizierung der Aussagekraft, die wir anstreben. Damit wählen wir die aussagekräftigsten Worte aus, indem wir alle Worte nach  $\Delta(\text{Vorkommen in Spam} - \text{E} - \text{Mails}, \text{Vorkommen in Nicht} - \text{Spam} - \text{E} - \text{Mails})$  absteigend sortieren und die ersten N Worte wählen. Damit ergibt sich folgendes Akzeptanzkriterium:

**Requirement 2:** Wenn  $N = 2$  ist, und als Worte mit Vorkommnissen gegeben sind:

Wort	Vorkommnisse in Spam-E-Mails	Vorkommnisse in Nicht-Spam-Emails
A	0.5	0.5
B	0.2	0.2
C	0.3	0.5
D	0.2	0.8
E	0.9	0.2

dann werden als Wortliste D und E gewählt.

### 3.3 Zusammenfassen der Wortlisten

Wir haben nun zwei separate, lokale Wortlisten berechnet. Diese müssen zusammengefasst werden zu einer gemeinsamen Wortliste. Da wir annehmen, dass die Benutzer ehrlich sind, können wir annehmen, dass die Benutzer als Eingabe dieses Protokolles keine beliebige Liste festlegen und auch ihre Daten nicht so manipulieren, dass eine bestimmte Wortliste versendet wird. Zudem werden die Attribute des Baumes öffentlich sein, sodass eine Geheimhaltung der Wortlisten keinen Sinn macht. Deswegen können wir die Wortlisten durch eine einfache Vereinigung der separaten Wortlisten zu einer gesamten Wortliste vereinigen. Deswegen wird das Zusammenfassen der Wortliste implementiert, indem beide Anwender ihre lokale Wortliste an den jeweils anderen Anwender versenden und beide Anwender lokal die Wortlisten vereinigen. Damit ergibt sich folgendes Akzeptanzkriterium:

**Requirement 3:** Wenn Alice die Wortliste A, B, C berechnet hat, und Bob die Wortliste C, D, E berechnet hat, dann muss die zusammengefasste Wortliste A, B, C, D, E sein.

## 4 Finden der gemeinsamen Schwellwerte

Einleitung, auf Figure für Schwellwerte verweisen

Für section-Titel besseren Begriff für "Vorkommnisse der Worte in eigenen Spam/Nicht Spam E-Mails" finden

### 4.1 Berechnung der Anteile

Content

### 4.2 Bestimmung der eigenen Schwellwerte

Es muss nun für ein Wort  $w$  eine Schwelle gefunden werden, ab wann das Wort  $w$  in einer E-Mail oft vorkommt. Wir wählen hierzu den Mittelwert der Vorkommnisse des Wortes in allen eigene E-Mails eines Anwenders, da wir dann die Begriffe „überdurchschnittlich oft“ bzw „unterdurchschnittlich oft“ als eigenen Schwellwert quantifizieren. Damit ergibt sich folgendes Akzeptanzkriterium:

**Requirement 4:** Wenn die E-Mails "A A A", "A B B", "A C C" und "A A C" sind, dann muss die Software als eigenen Mittelwert für das Wort A  $\frac{1}{4} \cdot (1 + \frac{1}{3} + \frac{1}{3} + \frac{2}{3}) = \frac{7}{12}$  bestimmen.

Content

### 4.3 Synchronisierung der Schwellwerte

Content



## 5 Diskretisieren der eigenen E-Mails

Einleitung, Figure referenzieren

Content

## 6 Lernen der gesamten E-Mails

verteiltes ID3 beschreiben

### 6.1 Yaos Protokoll

Die Eingabe jedes Anwenders für Yaos Protokoll ist eine Eingabe, und nach Kerckhoffs Prinzip auch der auszuführende Schaltkreis in Reinform. Einer der beiden Anwender, ohne Beeinträchtigung der Allgemeinheit Alice, muss nun aus diesem Schaltkreis einen enstestellten Schaltkreis konstruieren. Dies kann am einfachsten passieren, indem der Schaltkreis in einer Breitensuche traversiert wird und die Kanten verschleiert werden. Das bedeutet, dass in einem Schritt die zusammenhängenden Eingangs- und Ausgangsvariablen verschleiert werden. Dadurch werden die Anforderungen aus der Definition der enstestellten Schaltkreise erfüllt. Dabei kann dann auch erkannt werden, dass eine Eingangsvariable bzw Ausgabevariable kein „anderes Ende“ hat, d.h. diese Verschleierung in die Eingabekodierung bzw Ausgabekodierung geschrieben werden muss. Dieser Schaltkreis und die Ausgangskodierung wird dann an Bob übertragen, während die Eingangskodierung bei Alice verbleibt.

Bei der Verschlüsselung der Ausgaben verwenden wir ein einfaches XOR. Um zu garantieren, dass wir Unsinnige Verschlüsselungen erkennen können, erweitern wir jeden Tabelleneintrag um eine zweiten Tabelleneintrag, der aus einem analog zum Nutzeintrag verschlüsselten Nullvektor besteht. Wir bezeichnen diesen Eintrag als Markierungseintrag. Dann ist es möglich, diesen Eintrag zu entschlüsseln, zu überprüfen, ob dies der 0-Vektor ist und falls dies der Fall ist, den Nutzeintrag zu entschlüsseln. Der einzige Fall, wo dies Probleme bereitet ist, wenn die bei zwei Eingaben die Verschleierungen genau konträr sind, da dann  $a \oplus b = 0 = b \oplus a$  gilt und somit die Eingabe (0,1) und (1,0) die gleiche Ausgabe liefert. Dies muss also beim Verschleiern verhindert werden, was jedoch kein Problem ist.

Bob muss nun noch die Belegung der Eingangskanten lernen. Die Eingabevariablen, die von Alice Eingabe belegt werden müssen werden belegt, indem Alice anhand der Eingangskodierung ihre Eingabe kodiert und diesen String an Bob versendet. Da Bob keine Informationen darüber hat, welcher der beiden Verschleierungswerte für 0 oder für 1 steht, erhält Bob damit keine Informationen über Alice Eingabe.

Für die Kodierung von Bob's Eingabe muss etwas mehr Aufwand getrieben werden, da weder Bob die Eingabekodierung lernen darf (weil er sonst Alice Eingabe lernt), noch Alice Bob's Eingabe lernen darf (indem z.B. Bob seine Eingabe an Alice schickt, damit Alice die Eingabe kodieren kann). Daher wird dies durch eine Anwendung des 1-2-Oblivious-Transfers gelöst: Alice bietet für jede von Bob zu belegende Eingabevariable die Werte für 0 bzw 1 an, und Bob wählt mithilfe seines Eingabebits den richtigen Wert aus. Aus den Eigenschaften von OT folgt, dass Alice nicht weiss, welchen der beiden Werte Bob gewählt hat, und dass Bob den nicht gewählten Wert nicht kennt. Somit ist die Sicherheit

von Bobs Eingabe gegeben und sichergestellt, dass Bob wirklich seine Eingabe verwendet.

Wir implementieren den 1-2 Oblivious Transfer mittels RSA. Die Eingabe von Alice seien  $m_1, m_2$  und die Eingabe von Bob sei  $b$ . Alice ein Schlüsselpaar, den privaten Exponenten  $e$ , den öffentlichen Exponenten  $d$  und sendet den Modulus  $N$ ,  $d$  und zwei zufällige Nachrichten  $x_1$  und  $x_2$  an Bob. Bob wählt eine zufällige Nachricht  $k$ , verschlüsselt  $k$  zu  $k'$  und sendet  $v = x_b + k' \bmod N$  an Alice. Alice berechnet nun  $k_0$  als Entschlüsselung von  $v - x_0$  und  $k_1$  als Entschlüsselung von  $v - x_1$  und sendet  $m_0 + k_0$  und  $m_1 + k_1$  an Bob. Wenn Bob  $b = 0$  gewählt hatte, kann Bob  $m_0$  durch Subtraktion von  $k'$  von Alice erster Nachricht berechnen, und wenn Bob  $b = 1$  gewählt hatte, kann Bob  $m_1$  durch Subtraktion von  $k'$  von Alice zweiter Nachricht berechnen. Das folgt (oBdA für  $b = 0$ ) aus  $m_0 + k_0 - k' = m_0 + D(v - x_0) - k' = m_0 + k' - k' = m_0$ . Falls  $b$  nicht 0 war, heben sich in der Entschlüsselung die Faktoren  $x_0$  und  $x_1$  nicht auf und ein undefinierter Wert wird berechnet.

Danach kann Bob den Schaltkreis auswerten, indem er immer für eine Entscheidungstabelle, deren Eingabevariablen vollständig belegt sind, denjenigen Eintrag sucht, indem für jeden Eintrag der Markierungseintrag entschlüsselt wird und falls das Ergebnis 0 ist, der Nutzeintrag entschlüsselt wird und der Wert entsprechend weiterverwendet wird. Sobald alle Ausgangsvariablen des gesamten Schaltkreises bekannt sind, kann anhand der Ausgabedekodierung die Ausgabe in Klartext berechnet werden und Alice zugeschickt werden. (Hier wird wieder die Annahme des ehrlichen Anwenders getroffen).

## 6.2 Feststellen der dominierenden Ausgabe

Feststellung der benötigten Bytezahl beschreiben

Schaltkreis designen: Maximum von Summen von positiven Zahlensequenzen

## 6.3 Feststellen ob Ausgabe eindeutig

Feststellung der benötigten Bytezahl beschreiben

Schaltkreis designen: Gleichheit.

## 6.4 Das Entropien-Protokoll

Schaltkreis für  $x * \log x$ -Protokoll aus dem Paper zusammenfassen

## 6.5 Attribut mit maximalem Informationsgewinn finden

Feststellung der benötigten Bytezahl beschreiben

Vorgheen zusammenfassen, Schaltkreis aus dominierender Ausgabe  
wiederverwenden

## 7 Verwenden des Klassifikators

Es muss zusätzlich zum Lern-Programm ein Programm geschrieben werden, welches den Klassifikator auf eine Menge von E-Mails anwendet. Wir bieten hierzu ein Programm an, welches den Klassifikator in einem einfachen Textformat einliest und diesen auf eine Menge von E-Mails anwendet. Diese Menge von E-Mails ist als Dateien in einem Verzeichnis gegeben und für jeden Dateinamen wird eine Klassifikation als Spam oder Not Spam ausgegeben.

### 7.1 Eingabe des Klassifikators

Ausgehend von den Definitionen eines Attributes und eines Entscheidungsbaumes kann leicht eine Grammatik erzeugt werden, welche die Form des eingegebenen Klassifikators eindeutig bestimmt. Wir notieren diese Grammatik in BNF. Diese ist so gewählt, dass sie eine LL(1)-Grammatik ist, d.h., sie ist besonders einfach zu parsen.

```
tree -> 'Decide' '(' attribute (',' tree)+ ')'
      | 'Output' '(' class ')'.
class -> 'Spam' | 'Not Spam'.
attribute -> '(' word ',' probability ',' probability ')'.
word -> ('a' | 'b' | ... | 'z' | 'A' | ... | 'Z')+
probability -> number '.' number .
number -> ('0' | '1' | ... | '9')+.
```

Somit wäre ein kodierter Klassifikationsbaum beispielsweise:

```
Decide((Foo, 0.3, 0.6),
      Output(Spam),
      Decide((Bar, 0.5, 0.6),
            Output(Spam),
            Output(Not Spam)
          )
    )
```

Die Produktion „probability“ beschreibt eine Zahl, die sich als eine Folge von Ziffern vor einem Dezimalpunkt und einer Folge von Ziffern nach dem Dezimalpunkt beschreiben lassen. Dies sind alle reellen Zahlen, und da wir keine komplexen Zahlen betrachten, sondern nur Verhältnisse von natürlichen Zahlen zueinander ist diese Darstellung ausreichend, um alle möglichen Zahlenwerte darzustellen. Da die Buchstabenmenge als die lateinischen Klein- und Grossbuchstaben definiert ist und ein Wort definiert ist als Sequenz dieser Zeichen, ist die Produktion „word“ ausreichend, um alle möglichen Worte darzustellen. Damit ergibt sich, dass die Produktion „attribute“ in der Lage ist, alle Attribute, die in dieser Anwendung auftreten können, darzustellen.

Es gibt desweiteren bei uns nur die Ausgaben „Spam“ und „Nicht Spam“. Damit ist die Produktion „class“ ausreichend, um alle möglichen Ausgaben

des Baumes darzustellen. Daraus folgt, dass die zweite Produktion von `tree` in der Lage ist, alle möglichen Blätter darzustellen. Desweiteren folgt induktiv aus der Vollständigkeit der Produktion „attribute“ und der Darstellbarkeit der Blätter als Induktionsanfang, dass die erste Produktion von `tree` in der Lage ist, alle möglichen auszugebenden Entscheidungsbäume darzustellen. Damit ist die Grammatik mächtig genug für unsere Zwecke.

Es ist weiterhin zu bemerken, dass eine semantische Validierung notwendig ist, da in der Grammatik weder gefordert ist, dass der untere Schwellwert eines Attributes wirklich kleiner ist als der obere Schwellwert eines Attributes noch dass beide Schwellwerte zwischen 0 und 1 liegen, noch dass die Anzahl der Teilbäume richtig ist. Damit ergeben sich die folgenden Akzeptanzkriterien:

**Requirement 5:** Der Klassifizierer weist die Eingabe `Decide((Foo, 0.5, 2), Output(Spam), Output(Spam))` wegen eines zu grossen Schwellwertes zurück

**Requirement 6:** Der Klassifizierer weist die Eingabe `Decide((Foo, 2, 3), Output(Spam), Output(Spam))` wegen eines zu grossen Schwellwertes zurück

**Requirement 7:** Der Klassifizierer weist die Eingabe `Decide((Foo, 1, 0), Output(Spam), Output(Spam))` wegen falsch sortierter Schwellwerte zurück

**Requirement 8:** Der Klassifizierer weist die Eingabe `Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam))` wegen zuweniger Teilbäume zurück

**Requirement 9:** Der Klassifizierer weist die Eingabe `Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam), Output(Spam), Output(Spam))` wegen zuvieler Teilbäume zurück

**Requirement 10:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 0.2, 0.3), Output(Spam), Decide((Bar, 0.3, 0.4) Output(Spam), Output(Not Spam), Output(Spam)), Output(Spam))`

**Requirement 11:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 0, 0.5), Output(Spam), Output(Spam))`

**Requirement 12:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 0.5, 1), Output(Spam), Output(Spam))`

**Requirement 13:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 0.5, 0.5)), Output(Spam), Output(Spam))`

**Requirement 14:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 0, 0), Output(Spam))`

**Requirement 15:** Der Klassifizierer akzeptiert die Eingabe `Decide((Foo, 1, 1), Output(Spam))`

## 7.2 Arbeitsweise des Klassifikators

Der Klassifikator bekommt als Eingabe ein Verzeichnis mit E-Mails und eine Datei meinem einem Klassifikator. Den Klassifikator liest er ein und speichert ihn intern. Danach traversiert der Klassifikator das gegebene Verzeichnis rekursiv und behandelt jede Datei, die er in diesem Verzeichnis findet als E-Mail-Inhalt. (Dadurch ist es möglich, die Trainingsdaten auch als Versuchsdaten zu benutzen, ohne sie zu bewegen).

Für jede E-Mail wird dann der Inhalt der E-Mail eingelesen und die Klassi-

fikation durch den eingegebenen Klassifikator durchgeführt, d.h., für jeden Ast werden die Vorkommnisse des Wortes im Attribut festgestellt und rekursiv im entsprechenden Teilbaum weiterklassifiziert und in einem Blatt wird die Klasse festgestellt. Diese festgestellte Klasse wird dann zusammen mit dem relativen Pfad vom eingegebenen Verzeichnis ausgegeben.

Wenn also beispielsweise folgende Verzeichnisstruktur gegeben ist:

```
mails/hank/mail1
mails/hank/mail2
mails/bob/mail1
```

und wir annehmen, dass der eingegebene Klassifikator E-Mails mit dem Index 1 als Spam erkennt und E-Mails mit dem Index 2 als Nicht Spam, dann wäre die Ausgabe:

```
hank/mail1 Spam
hank/mail2 Not Spam
bob/mail1 Spam
```

Damit ergeben sich folgende Akzeptanzkriterien:

**Requirement 16:** Gegeben der Klassifikator `Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam))`, dann wird die E-Mail "Bar Foo Foo Foo" als Spam klassifiziert

**Requirement 17:** Gegeben der Klassifikator `Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam))`, dann wird die E-Mail "Bar, Bar, Foo, Foo" als Not Spam klassifiziert

**Requirement 18:** Gegeben der Klassifikator `Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam))`, dann wird die E-Mail "Bar, Bar, Bar, Foo" als Spam klassifiziert

**Requirement 19:** Gegen der Klassifikator `Decision((Bar, 0.3, 0.6), Output(Spam), Output(Not Spam), Output(Spam))` und eine Verzeichnisstruktur wie oben skizziert, wobei `hank/mail1` "Bar Foo Foo Foo", `hank/mail2` "Bar Bar Foo Foo" und `bob/mail1` "Bar Bar Bar Foo" enthält, dann wird die oben als Beispiel genannte Ausgabe produziert (oder in einer anderen Reihenfolge)

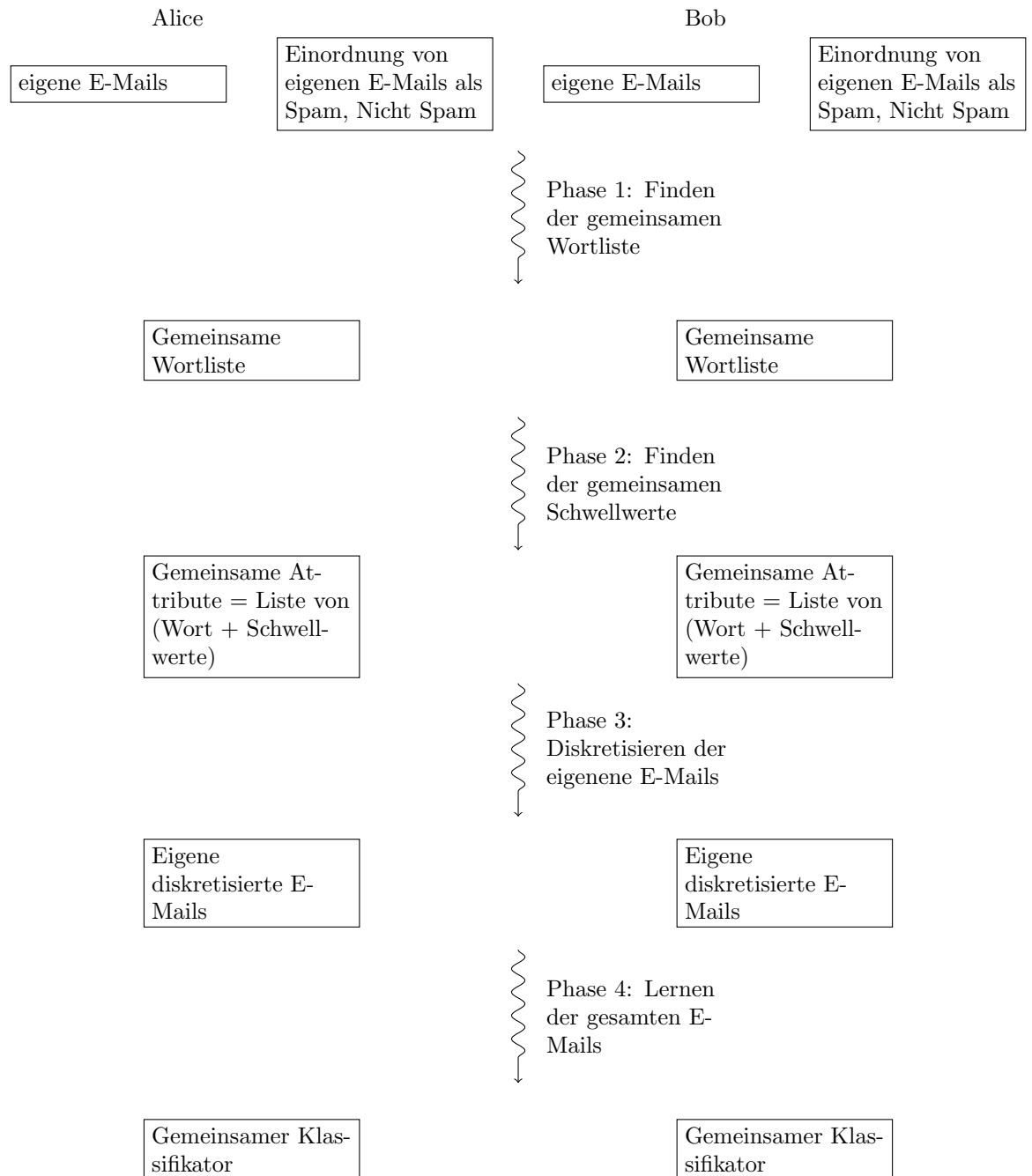


Figure 1: Phasen der Anwendung



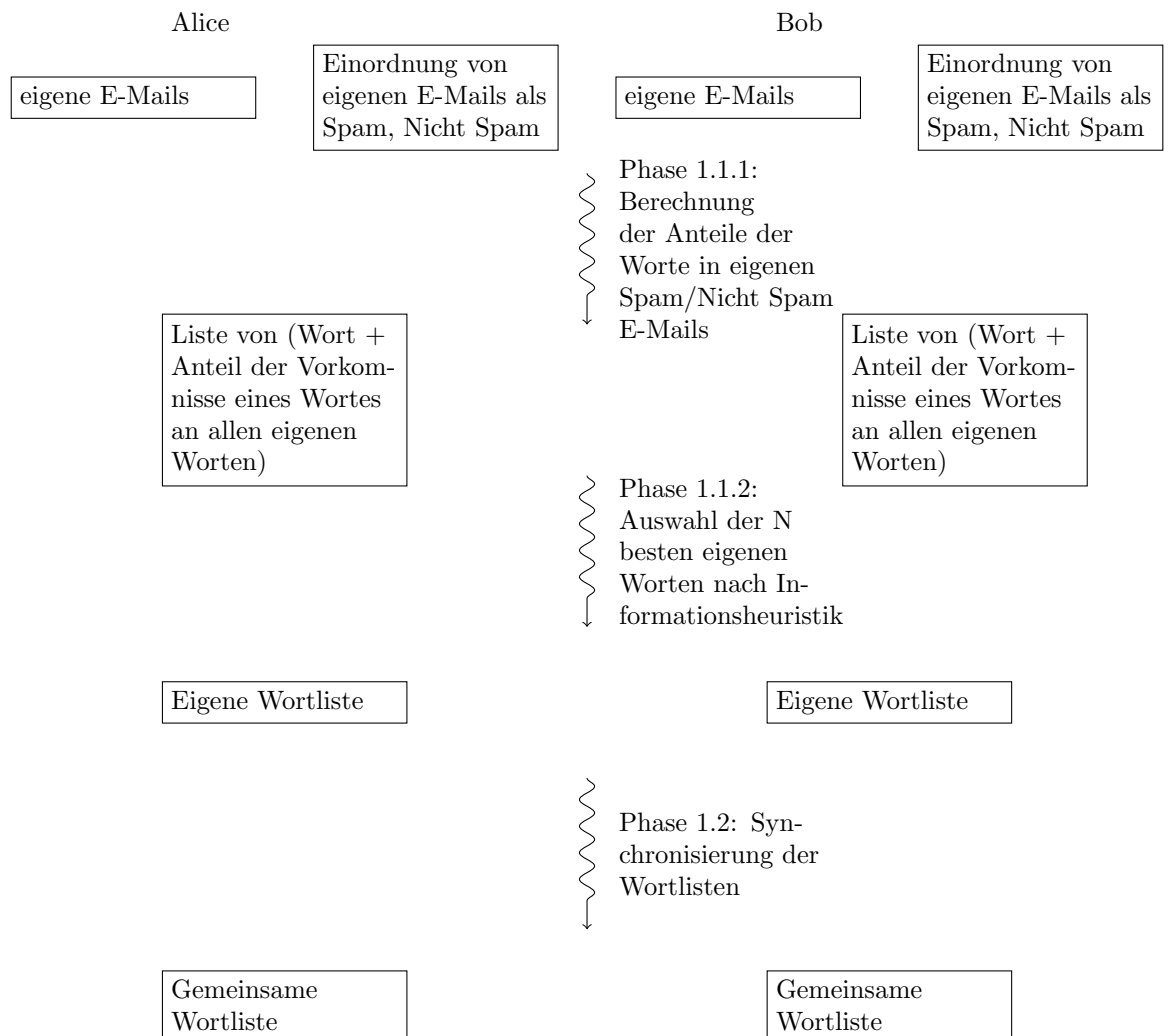
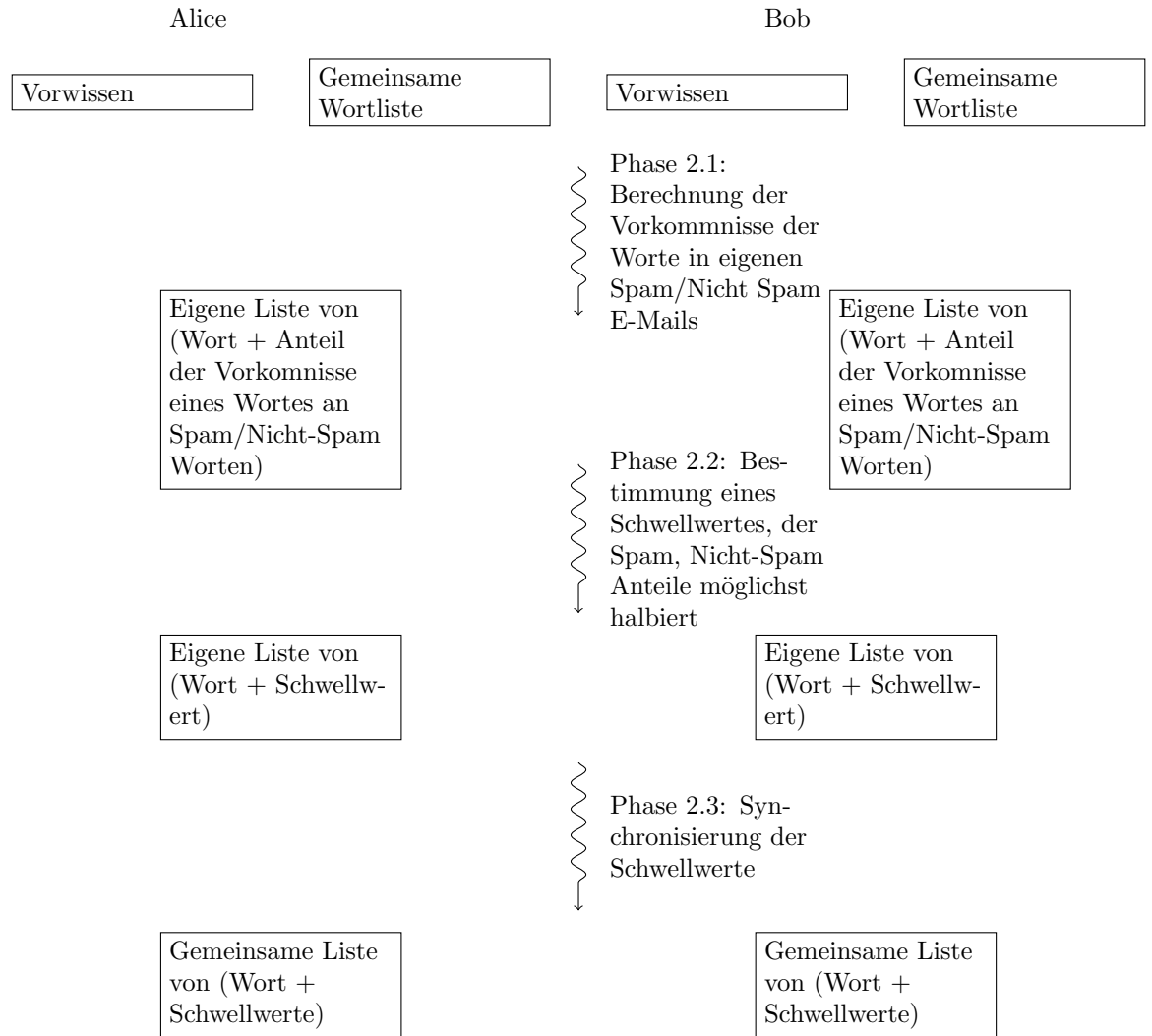
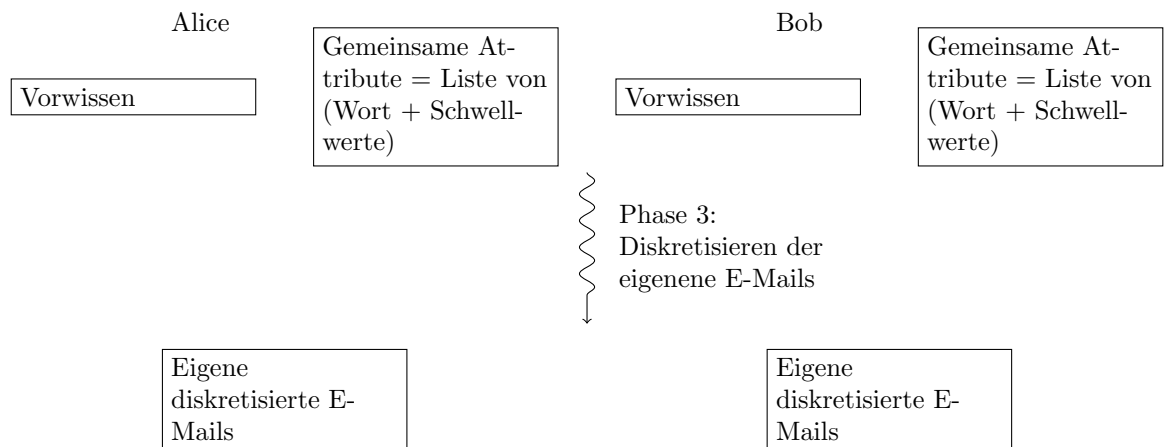


Figure 2: Schritte zum Berechnen der gemeinsamen Wortliste





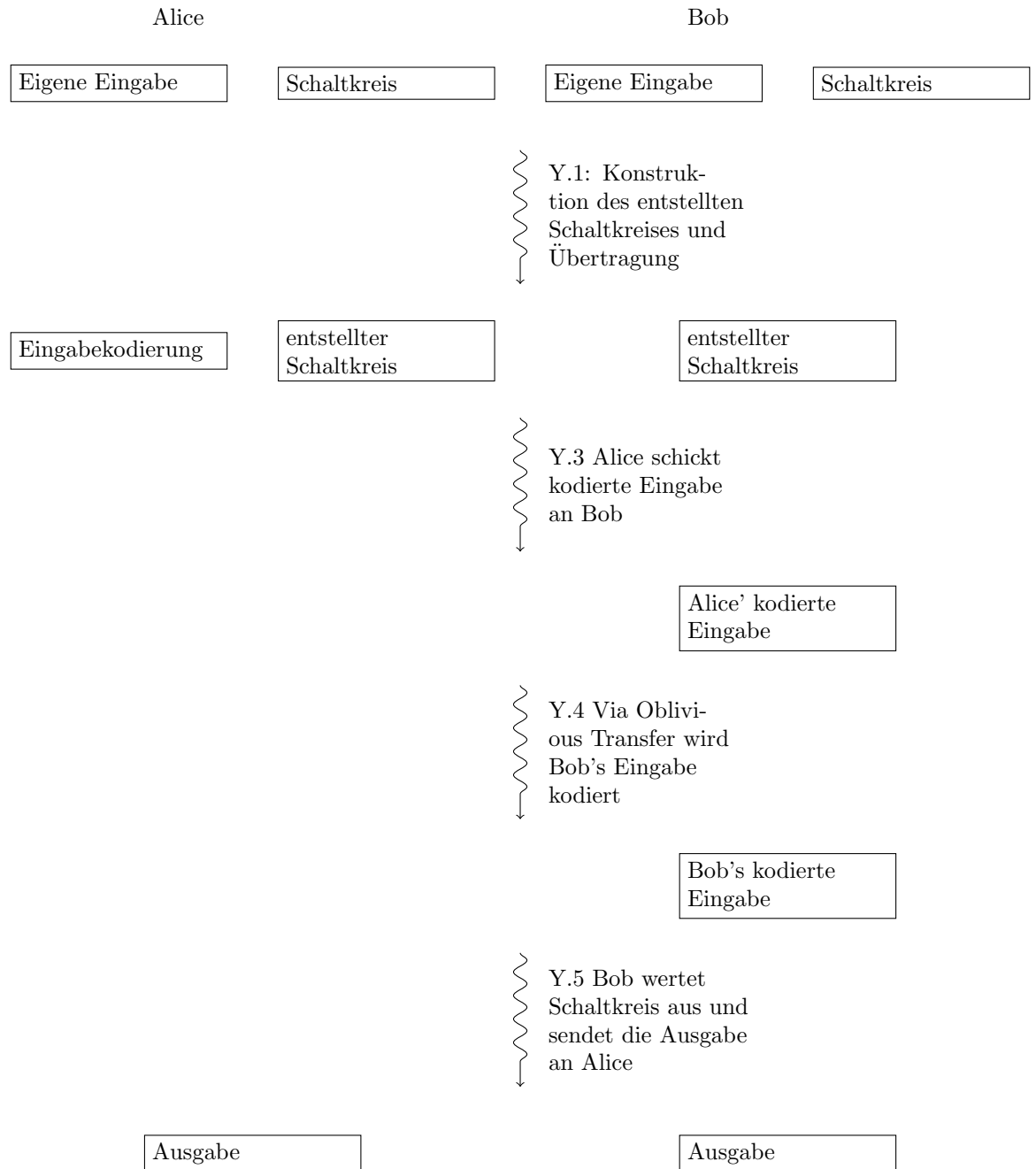


Figure 3: Yaos Protokoll

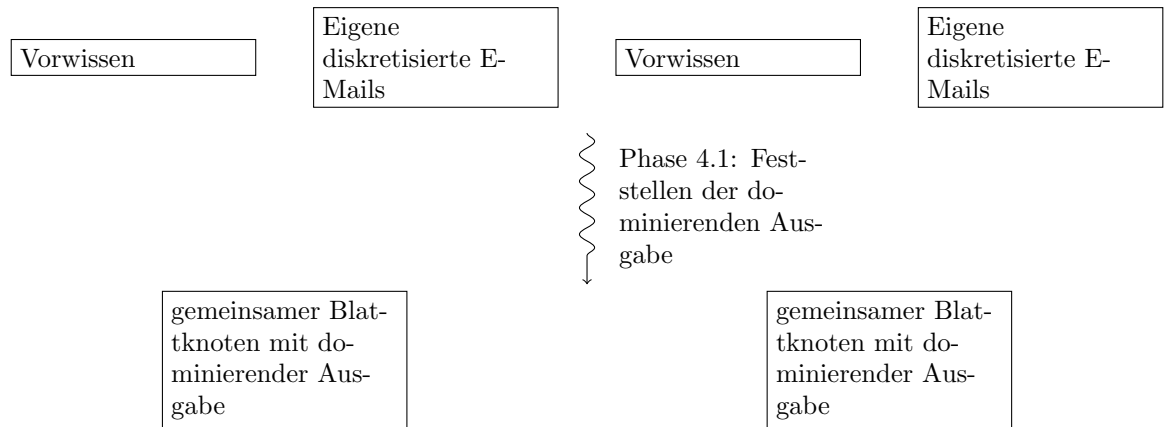


Figure 4: ID3-Algorithmus, Fall 1: Keine Attribute mehr vorhanden

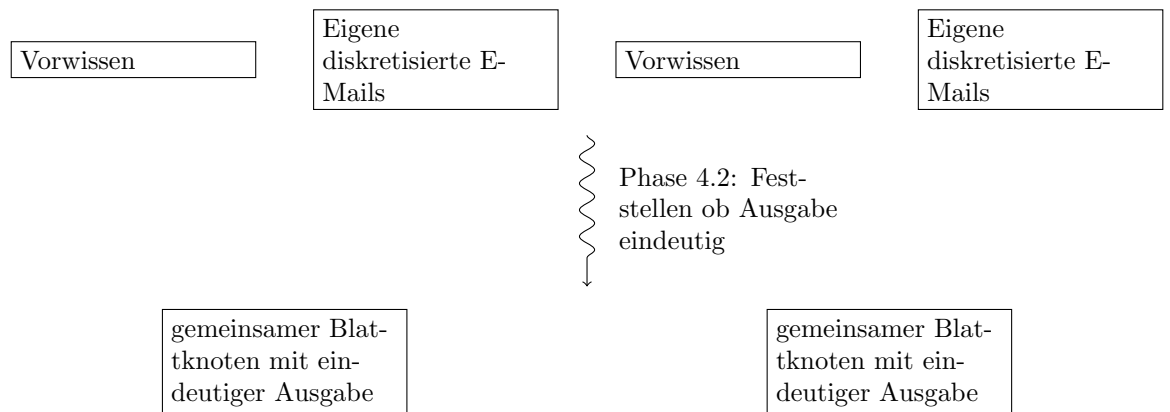


Figure 5: ID3-Algorithmus, Fall 2: Ausgabe eindeutig

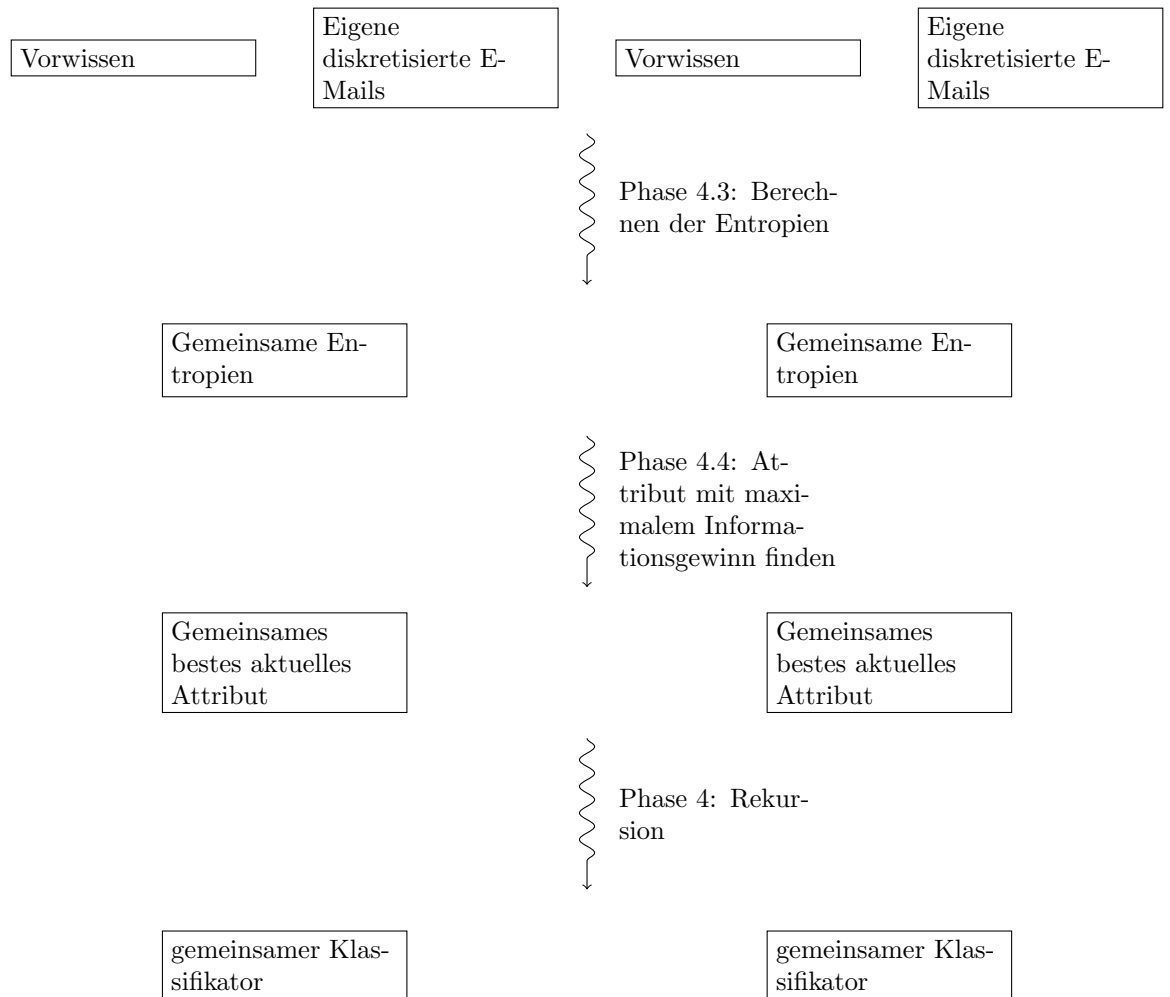


Figure 6: ID3-Algorithmus, Fall 3: Erzeugung eines Astes