Contents

1 Grundlagen

Definition 1. Anwender, aktiver Anwender
Die Benutzer des Programmes werden im folgenden als **Anwender** bezeichnet. Im Folgendenden bezeichnen A, B zwei Anwender, die das Programm verwenden.
Aus technischen Gründen wird einer der beiden Anwender als **aktiver Anwender** bezeichnet und der andere der beiden Anwender als **passiver Anwender**. Der aktive Anwender ist in der Lage, den Ablauf eines Protokolles zu initiieren, während der passive Anwender auf die Initiierung eines Protokolles durch den aktiven Anwender wartet. Es sei im folgenden A der aktive Anwender und B der passive Anwender.

Definition 2. Wissen, Wissen eines Anwenders, gemeinsames Wissen
Wenn A ein Anwender ist, so gibt es **Wissen des Anwenders**. uf das Wissen des Anwenders kann nur der Anwender selbst zugreifen. Wenn ein Ausdruck als **gemeinsames Wissen c** bezeichnet wird, so ist dies äquivalent dazu, dass c im Wissen aller Anwender vorkommt.

Definition 3. Protokoll
Wir bezeichnen im Folgenden eine Sequenz von Nachrichtenaustauschen zwischen zwei Anwendern als **Protokoll**

Definition 4. Zwei-Parteien-Berechnung, private Zwei-Parteien-Berechnung, Eingabe, Ausgabe
Gegeben zwei Anwender A, B mit Wissen $A.i$ und $B.i$ und eine Funktion f, dann wird ein Protokoll als **Zwei-Parteien-Berechnung** von Wissen j bezeichnet, wenn nach Ausführung des Protokolls $A.j = f(A.i, B.i) \wedge B.j = f(A.i, B.i)$ gilt. Wir bezeichnen i als **Eingabe** des Protokolles der Anwender und j als **Ausgabe** des Protokolles.
Wenn ausserdem gilt, dass A nur $A.j$ als Wissen ueber $B.i$ erhält und dass B nur $B.j$ als Wissen ueber $A.i$ erhält, dann bezeichnen wir das Zwei-Parteien-Protokoll als **privates Zwei-Parteien-Protokoll**.

Bemerkung 1. Wenn ein Zwei-Parteien-Protokoll eine Funktion f berechnen soll, die Eingaben X zusätzlich zum Wissen der Benutzer erfordert, kann dies durch Definition einer Funktion *f'(i, j) = f(x, i, j)* erreicht werden. Dadurch können beispielsweise gemeinsames Wissen oder zuvor festzulegende Parameter genutzt werden. Damit ist die Definition 4 ausreichend, um beliebige Funktionen zu beschreiben.

Definition 5. Phase, private Phase
Wir bezeichnen im Folgenden einen Schritt der Anwendung als **Phase**. Eine Phase erfordert ein gewisses **Vorwissen** der beteiligten Anwender und garantiert dann, dass nach Ausfuehren der Phase ein gewisses anderes Wissen erlangt wird. Rein technisch ist eine Phase also nur ein Synonym für eine Funktion auf dem Wissen der Anwender. Damit ergibt sich jedoch, dass sich auch die Phasen klassifizieren lassen. Eine **private Phase** ist somit eine Phase, bei der die beteiligten Anwender über das Vorwissen der anderen beteiligten Anwender nur das Wissen erhalten, dass sie aus dem durch die Phase berechneten Wissen schlussfolgern können. Eine **getrennte Phase** ist eine Phase, welche keine Kommunikation zwischen den Anwendern erfordert.
Im folgenden werden wir Bilder von Phasen darstellen. In diesen Bildern werden im Allgemeinen nur zwei Anwender dargestellt sein. In diesem Bild ist das Vorwissen und das erhaltene Wissen der Anwender als Rechtecke dargestellt werden. Das Wissen der Anwender ist durch eine vertikale Linie getrennt und der linke Anwender wird ,,Alice" genannt, während der rechte Anwender Bob genannt wird. Die Ausführung einer Phase wird als Kreis zwischen beiden Anwendern dargestellt werden, Pfeile von einem Wissens-Rechteck in einen solchen Phasen-Kreis sollen andeuten, dass die Phase dieses Wissen verwendet und Pfeile von einem solchen Phasen-Kreis zu einem Wissensrechteck sollen andeuten, dass die Phase dieses Wissen erzeugt. Falls eine Phase getrennt ist, wird der bzw die Phasenkreise vollständig in die Hälfte eines Anwenders gezeichnet, andernfalls wird der Phasenkreis auf der gestrichelten Trennlinie gezeichnet werden. Eine einfache Übertragung von Wissen wird dargestellt als Pfeil von einem Wissens-Recheckt über die gestrichelte Linie zu einem anderen Wissens-Rechteck. (vgl auch Figure 1)

2 Design der Anwendung

Es wird nun die Anwendung designed. Hierzu wird rückwärts vorgegangen, d.h., wir betrachten zuerst, welche Ausgabe die Anwendung produzieren soll und betrachten dann Alternativen, diese Ausgabe zu erzeugen. Dadurch ergeben sich dann weitere benötigte Eingaben für diese Schlussphase der Anwendung, die dann rekursiv behandelt werden können, bis als Eingabe nur noch die vorklassifizierten E-Mails und andere Anwendereingaben übrig bleiben. An diesem Punkt ist die Anwendung dann fertig designed.
Um zu rekapitulieren, die Vision der Anwendung ist:,,Ich habe eine Menge von vorklassifizierten E-Mails und kenne einen Anwender, der ebenfalls eine Menge

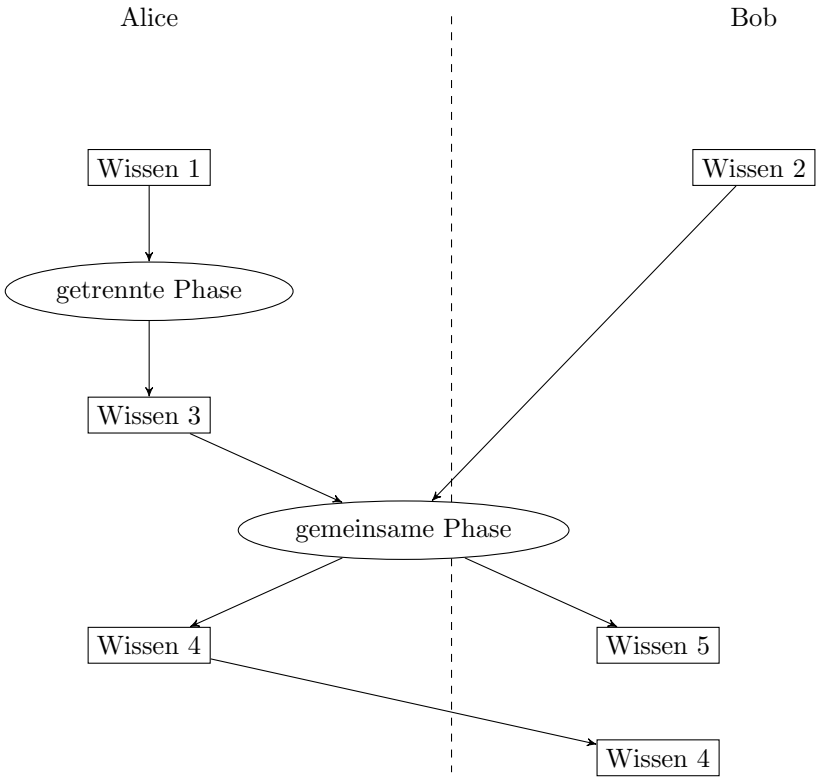

Figure 1: Phasenbild: Alice mit Phase alleine, beide mit gemeinsamer Phase, Alice sendet Wissen an Bob.

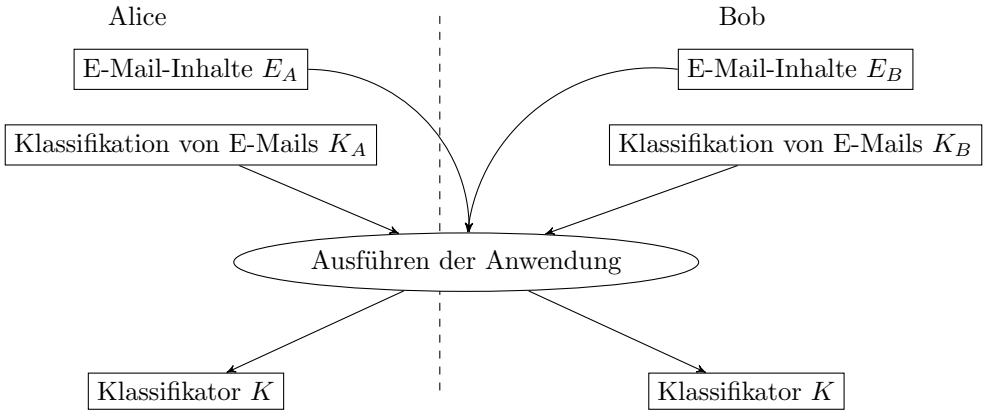

Figure 2: Phasendiagramm der Anwendung: Beide haben Inhalte und Vorklassifizierungen und wollen einen Klassifikator

von vorklassifizierten E-Mails hat und will gemeinsam mit diesem einen Klassifikator fuer Spam und Nicht-Spam berechnen. Dabei will ich jedoch, dass er über meine E-Mails nur soviel lernt, wie ich will und wie der Klassifikator preisgibt.". Wenn wir diese Vision als Phasendiagramm formulieren, ergibt sich Abbildung 2.

2.1 Erstellung des Klassifikators

2.1.1 Verwenden eines nicht verteilten Klassifikators

Um das Vorgehen zu verstehen und der Vollständigkeit halber wird die Möglichkeit aufgeführt, den Klassifikator nicht verteilt zu berechnen. Dazu wird das vollständige Wissen ausgetauscht und dann lokal der Klassifikator berechnet. Dies ist in Abbildung 3 dargestellt. Es ist deutlich zu erkennen, dass die Eingabe der Anwendung die Klassifikation und die E-Mails des Anwenders sind und dass die Ausgabe des Protokolls ein Klassifikator ist. Wenn beide Anwender sich an das Protokoll halten, ist dies sogar der gleiche Klassifikator für beide, sodass die Vision durch diese Implementierung erfüllt wird.
Es gibt in dieser Vorgehensweise keinerlei Geheimhaltung, da alle Eingaben der Anwender an alle anderen Anwender übertragen werden.
Die Implementierung dieses Szenarios ist einfach, da Implementierungen für viele Lernverfahren zu finden sind und die initiale Übertragung einfach zu implementieren ist.

2.1.2 Verwenden von Entscheidungsbäumen

Definition 6. Entscheidungsbaum
Generell lassen sich Entscheidungsbäume wie folgt definieren: Es gibt eine

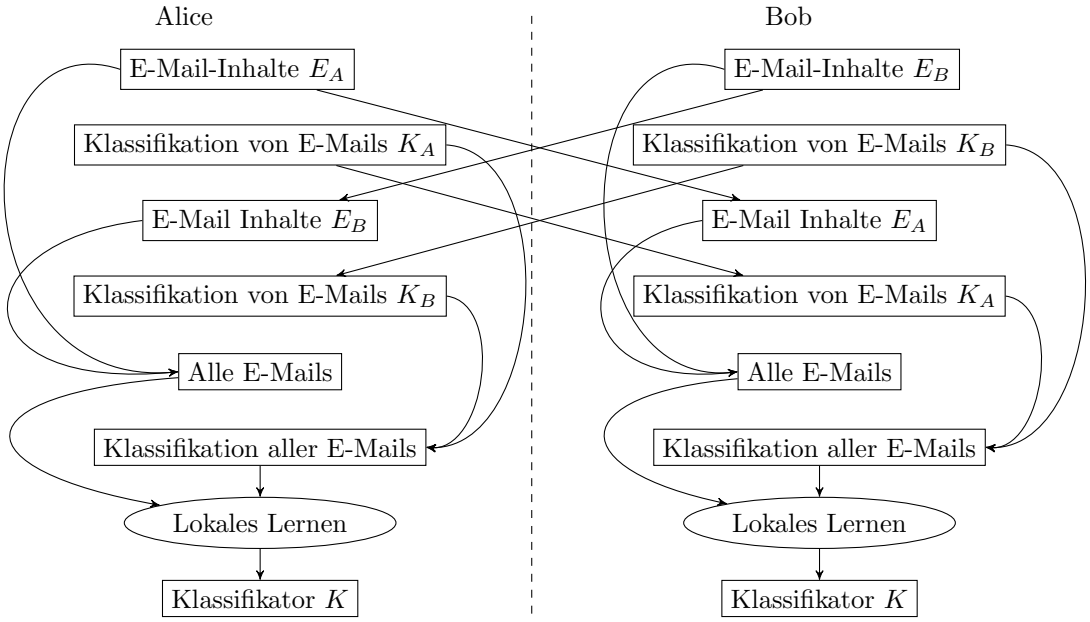

Figure 3: Lokale Berechnung der Klassifikatoren

Menge A von diskreten Attributen $A_1, A_2, \ldots, A_n$ mit Werten $values(A_i) = v_i^1, v_i^2, \ldots, v_i^w$ und es gibt zu erkennde Klassen $K = K_1, K_2, \ldots, K_k$. Es bezeichne im Folgenden desweiteren $Values = \bigcup_{A_i} values(A_i)$ die Menge aller möglichen Attributwerte (dies vereinfacht einige Definitionen, auch wenn einige Typen etwas ungenauer werden). Dann kann die Menge T von Entscheidungsbäumen induktiv definiert werden als: $T = (A \times Values \mapsto T) \cup K$
Ein Entscheidungsbaum klassifiziert eine Abbildung der Attribute auf Werte. Dieser klassifizierungsvorgang ist induktiv definiert. Wenn der zu klassifizierende Kandidat mit $s : A \mapsto Values$ bezeichnet wird, dann ergibt sich als Auswertungsfunktion $evaluate : (A \mapsto Values) \times T \mapsto K$:

$$evaluate(s, (A_i, f)) = evaluate(s, f(s(A_i))) \quad (1)$$

$$evaluate(s, K_i) = K_i \quad (2)$$

Verbal beschrieben besteht ein Entscheidungsbaum also aus Blättern, die aussagen, dass alle zu klassifizierenden Objekte, die dieses Blatt „erreichen" zu dieser bestimmten Klasse gehören und Ästen, die mit einem Attribut annotiert sind und zu jedem möglichen Wert des Attributes einen Unterbaum haben. Bei der Klassifizierung werden dann die Äste entsprechend der Attributwerte des zu klassifizierenden Objektes traversiert, bis ein Blatt erreicht wird und die Klassifizierung abgeschlossen ist.
Wenn wir beispielsweise annehmen, dass wir uns bei der Spamerkennung dadrauf

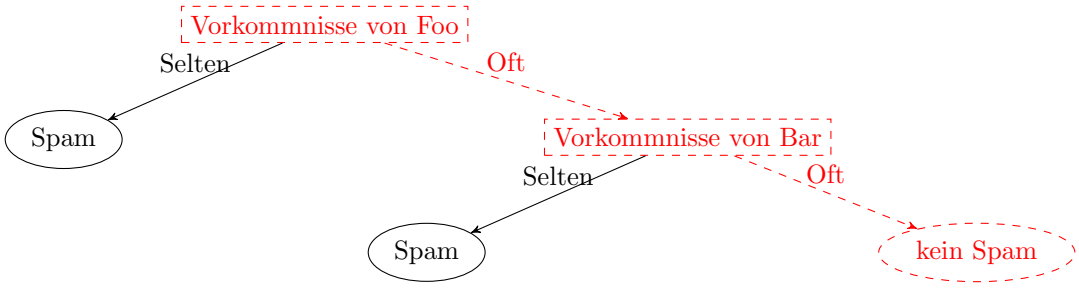

Figure 4: Möglicher Entscheidungsbaum, Klassifizierung von ,,Foo → Oft, Bar → Oft"

geeinigt haben, dass wir als Attribute verwenden, ob die Worte "Foo" bzw "Bar" oft oder selten vorkommen, dann würden die E-Mail-Datenbanken transformiert werden zu einem Vektor von Abbildungen von Foo und Bar auf oft oder selten:

Vorkommnisse Foo	Vorkommnisse Bar
Oft	Oft
Selten	Oft
Selten	Selten

Ein möglicher Entscheidungsbaum ist in Abbildung 4 dargestellt, der die Mails in Spam und Nicht Spam klassifiziert. Es ist hier ein Baum aufgezeichnet, der prinzipiell nur Mails als ,,Nicht Spam" klassifiziert, wenn sie oft Foo und Bar enthalten. In der Grafik ist zudem illustriert, welche Knoten und damit welche Kanten bei der Auswertung betrachtet wurden.

Ausgehend von dem ausgeteilten Paper gibt es ein privates 2 Parteien Protokoll, welches ein Lernverfahren, den ID3-Algorithmus, durchführt. Der ID3-Algorithmus ist ein heuristisches Verfahren, um aus einer Menge vorklassifizierte Zuordnungen einen Entscheidungsbaum zu produzieren. Damit kann man das in Abbildung 5 gezeigte Phasendiagramm aufstellen. Es muss somit festgelegt werden, wie die Attribute entstehen und wie die E-Mails diskretisiert werden müssen, um dann den ID3-Algorithmus verteilt und privat auszuführen und somit den Klassifikator zu berechnen.
Der ID3-Algorithmus ist ein heuristischer Greedy-Algorithmus. Er nimmt an, dass es optimal ist, lokal die Entropie zu minimieren und so den Informationsgewinn zu maximieren.
Formal definieren wir für die Ausgaben $K_1, K_2, \ldots, K_k$ und eine Datenmenge D die funktion $elements(D, K_i)$, welche die Menge von Werten in D liefert, die Ausgabe K_i haben. Analog definieren wir fuer das Attribut A_i mit Werten $v_i^1, v_i^2, \ldots v_i^w$ die Funktion $elements(D, A_i, v_i^j)$, welche die Elemente von D liefert, die für das Attribut A_i den Wert v_i^j haben. Dann definieren wir die Entropie einer Menge D als

$$Entropy(D) := \sum_{K_i} \frac{|elements(D, K_i)|}{|D|} \cdot \log \frac{|elements(D, K_i)|}{|D|} \tag{3}$$

Davon ausgehend definieren wir nun die Information, die wir erhalten, wenn wir eine Datenmenge D an einem Attribut A_i aufteilen:

$$Gain(D, A_i) := Entropy(D) - \sum_{v_i^j} Entropy(elements(D, A_i, v_i^j)) \qquad (4)$$

Danach wählt der Algorithmus bei der Erzeugung eines Astes durch vollständige Suche das Attribut aus, bei dem $Gain(D, A_i)$ maximal ist. Die einzelnen Partitionen der Datenmenge, gegeben durch die Werte des Attributes, werden dann rekursiv behandelt.
Dieser Algorithmus wurde dann verteilt und privat berechnet. Dazu ist es notwendig, $D = E_A \cup E_B$ zu definieren und die einzelnen Schritte des Algorithmus' (alle Ausgaben gleich? Maximum Gain?) als private Zwei-Parteien-Protokolle zu implementieren.
Es bleibt zu untersuchen, wieviele Informationen durch den ausgegebenen Klassifkator verloren werden. Prinzpiell kann aus dem Klassifikator immer nur entnommen werden, dass ein bestimmtes Attribut eine kleinere Entropie hatte als andere Attribute. Es ist damit nicht möglich, auszusagen, wie gross diese Entropie war, oder um wieviel grösser sie war als die anderen Entropien. Anhand der Blätter kann geschlussfolgert werden, dass zu diesem Zeitpunkt alle Attribute verbraucht waren (formal: dass die Sequenz der annotierten Attribute auf dem Pfad von der Baumwurzel zum Blatt alle Attribute enthält) und somit in der dann resultierenden Menge die Annotation des Blattes am häufigsten war oder dass nun die Daten „einstimmig" waren. Es ist damit nicht offensichtlich, wie über die Attribute hinaus einfach auswertbare Informationen preisgegeben werden.
Der berechnete Klassifikator gibt somit nicht auf offensichtliche Weise wichtige Informationen preis und laut dem Paper ist das Protokoll ebenfalls privat, d.h., das Protokoll gibt nur soviele Informationen preis wie die Ausgabe über die Eingabe preisgibt.

2.2 Festlegen der Attribute, Transformieren der E-Mails

Da wir keine Attributsvektoren als Eingaben haben, sondern nur Texte ist es notwendig, Möglichkeiten der Transformation zu untersuchen.

2.2.1 Berechnung einer Wahrscheinlichkeit und Diskretisierung

Eine gute Möglichkeit, derartige Vektoren zu erzeugen ist es, Vorkommnisse irgendeiner Form zu zählen, in Relation mit einer Gesamtanzahl zu setzen und danach dieses Verhältnis zu diskretisieren. Damit gibt es zwei weitere Unterprobleme: Wie bekommen wir diese Wahrscheinlichkeit, und wie diskretisieren wir sie?
Prinzipiell gibt es für die Diskretisierung zwei grosse Fragen: Wieviele Wertebereiche verwenden wir und wer bestimmt die Schwellwerte zwischen den Wertebereichen? Dies ist eine Entscheidung, die der Kunde treffen muss. Es muss

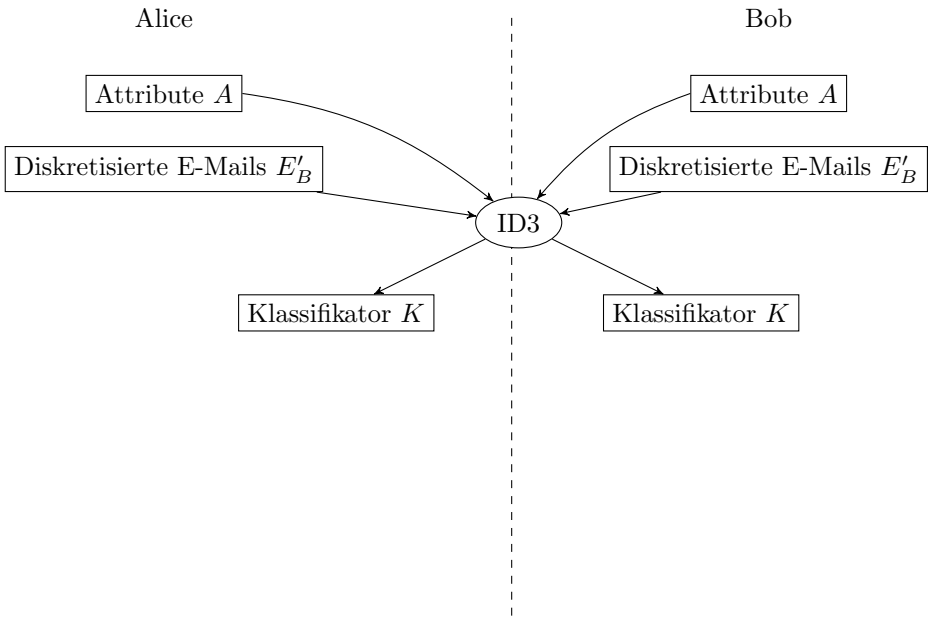

Figure 5: Phasendiagramm bei Verwendung von Entscheidungsbäumen und dem ID3-Algorithmus

hier bedacht werden, dass zuwenige Wertebereiche sehr grob sind (bei nur einem Wertebereich gibt es nur die Aussage, dass das Attribut betrachtet wurde, bei nur zwei Wertebereichen gibt es nur eine binaere Aussage) und dass zuviele Wertebereiche Probleme haben zu abstrahieren (Bei 1024 Wertebereichen abstrahiert man kaum noch von den eigentlichen Daten, weil (etwas dramatisiert) fast jeder Datenpunkt seinen eigenen Wertebereich hat. Damit wäre der Klassifikator dann praktisch nur auswendig gelerntes Wissen).
Für das Festlegen der Schwellwerte gibt es mehr Freiraum. Die erste Variante besteht darin, *den Nutzer die Werte eingeben zu lassen.* Dies erfordert dann eine Syncronisierung der Werte beider Anwender, ist dann jedoch maximal flexibel, da der Anwender alle Möglichkeiten hat, diese Schwellwerte zu bestimmen (neuartige Mustererkennungsalgorithmen, stundenlanges tunen der Werte).
Die zweite Variante besteht darin, dass *das Programm diese Werte selbst bestimmt.* Eine Möglichkeit hierzu besteht darin, das Histogramm der Wahrscheinlichkeitsverteilung zu betrachten und dann die Werte so zu platizieren, dass die Integrale in den Teilintervallen möglichst gleich gross sind. Das sollte dann die Information der einzelnen diskreten Werte maximieren. Der Vorteil hier besteht darin, dass dann die Werte nicht vom Anwender eingegeben werden müssen. Der Nachteil besteht darin, dass so keine Möglichkeit besteht, einen anderen, besseren Algorithmus zu verwenden. (Man beachte auch, dass man mit Variante 1 implementieren kann und dann Variante 2 durch Variante 1 im-

plementiert mitliefern kann).
Fuer die Syncronisierung der Werte kann man praktisch beliebige Funktionen verwenden, die zwei Sequenzen von reellen Zahlen auf eine Sequenz reeller Zahlen abbildet. Damit ist es etwas aufwändig, alle Möglichkeiten aufzuzählen. Eine Variante, die sich jedoch anbietet, ist das punktweise Arithmetische Mittel zu bilden, da hier die von beiden Nutzern verwendeten Schwellwerte zu gleichen Teilen beruecksichtigt werden. Zudem ist die Implementierung denkbar einfach, da der punktweise Mittelwert zweier streng monoton steigender Folgen streng monoton steigend ist und somit der punktweise Mittelwert von zwei Vektoren von Hysteresepunkten wieder ein Vektor von Hysteresepunkten ist.
Für die Berechnung von Wahrscheinlichkeiten gibt es ebenfalls mehrere Möglichkeiten. Die erste Klasse von Berechnungen drehen sich alle um die Anzahl der Vorkommnisse von Wortmengen. Es wird hierzu eine Menge von Mengen von Strings definiert und für jede dieser Stringmengen untersucht, welcher Prozentsatz der Worte in einer E-Mail in jeder dieser Stringmengen liegt. Wenn der Inhalt einer E-Mail beispielsweise "Foo Bar Baz" ist, und die Wortmengen $\{"Foo","Bar"\}$, $\{"Bar","Baz"\}$, $\{"Baz"\}$, dann sind die Vorkommnisse $\frac{2}{3}, \frac{2}{3}, \frac{1}{3}$. (Dies ist eine Generalisierung des vorherigen Ansatzes der Wortliste. Die Wortliste vom letzten Mal ist im Endeffekt eine solche Stringmenge, in der alle Mengen nur ein einzelnes Element enthalten). $< TODO|Bewertung >$
$< TODO|Phasendiagramm >$

2.3 Berechnung der Wortmengen

Die einfachste Variante, die Wortmengen zu berechnen ist die *statische Wortliste*. Hier wird die Wortliste statisch in das Programm einkodiert und beide verwenden diese statische Wortliste. Der Vorteil dieser Variante ist, dass so keine Syncronisierung der Wortlisten erforderlich ist und die preisgegebenen Informationen durch die Attribute des Baumes sehr einfach abzuschätzen sind, da man Attributliste definitiv kennt.
Die zweite Variante, die Wortmengen zu berechnen ist die *Eingabe vom Benutzer*. Dies hat den Vorteil, dass die Lösung sehr flexibel ist, da der Anwender bei jeder Anwendung eine an die Situation angepasste Wortliste verwenden kann. Diese Variante erfordert jedoch eine Syncronisation der Wortlisten beider Anwender, damit eine eindeutige Liste von Attributen für den ID3-Algorithmus berechnet werden kann.
Die dritte Variante, die Wortmengen zu berechnen ist die *Verwendung eines Algorithmus'* innerhalb der Software. Es ist interresant zu bemerken, dass die erste Variante eigentlich nur ein Spezialfall dieser Variante ist, in der nur statisch eine Liste ausgegeben wird. Die zweite Variante kann durch externe Programme so erweitert werden, dass die dritte Variante erreicht wird. Soll dagegen die dritte Variante mit einem einzelnen Algorithmus implementiert werden, muss wieder untersucht werden, wie die beiden berechneten Wortlisten syncronisiert werden und welcher Algorithmus verwendet werden kann.

2.4 Syncronisierung der Wortmengen

Die Syncronisierung der Wortmengen besteht prinzipiell in einem Protokoll, welches als Eingabe 2 solche Wortmengen-Mengen erhält und garantiert, dass nach Ablauf des Protokolles beide Anwender die gleiche Wortmengen-Menge wissen. Danach kann man dann diese Wortmengen als Attribute verwenden.
Ein einfacher Ansatz ist, dass als Eingaben genau zwei Wortlisten gegeben sind und auf diese eine Mengenoperation angewendet wird, um die resultierende Wortmengen zu bestimmen. Verwendet man den *Schnitt* der Wortmengen, so hat dies den Vorteil, dass garantiert ist, dass die eingegebenen Wortmengen auf jeden Fall eine Obermenge der schlussendlich ausgegebenen Wortmengen sein werden. Dadurch kann die Geheimhaltung besser gewährleistet werden, da so jeder Anwender genau weiss, welche Attribute es maximal geben kann (nämlich seine eigene eingegebene Wortliste).
Verwendet man die *Vereinigung*, so kann dem Anwender nicht mehr garantiert werden, welche Attribute verwendet werden. Es kann dann jedoch garantiert werden, dass jeder Anwender alle Wortmengen, die ihm wichtig sind, in der Klassifizierung berücksichtigt sieht.
Man kann den Ansatz der Vereinigung noch etwas erweitern um das potentielle Problem der Geheimhaltung lösen, indem man die *Vereinigung mit Blacklist* verwendet. Hierzu kann jeder Anwender neben der anders bestimmten Wortmenge noch eine Liste von Wörtern angeben, welche auf gar keinen Fall in einer der Wortmengen auftreten darf. Diese werden dann nach der Vereinigung der Wortmengen in einem zweiten Schritt entfernt werden, um die schlussendliche Wortmenge zu erhalten.