Spezifikation des verteilten, privaten Lerners

Gudrun Amedick, Tim Kunold, Harald Krämer

Contents

1 Einleitung

Dieses Dokument spezifiziert die im Projekt zu implementierende Anwendung. Die Spezifikation definiert eine gewisse Anzahl von Akzeptanzkriterien, die bei der Abgabe demonstrierbar sein müssen. Als Demonstration ist eine Demo, eine glaubhafte Code-Inspektion oder automatisierte Tests tauglich.
Im ersten Teil des Dokumentes werden die verwendeten Begriffe festgelegt, sowie die Annahmen, auf denen die Geheimhaltung der Daten jedes Anwenders beruhen. Danach werden zunächst gewisse technische Grundlagen geklärt, um danach die einzelnen Schritte in der Anwendung festzulegen.

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 foo 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

Generell lassen sich Entscheidungsbäume wie folgt definieren: Es gibt eine 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 $evaluate : (A \mapsto Values) \times T \mapsto K$:

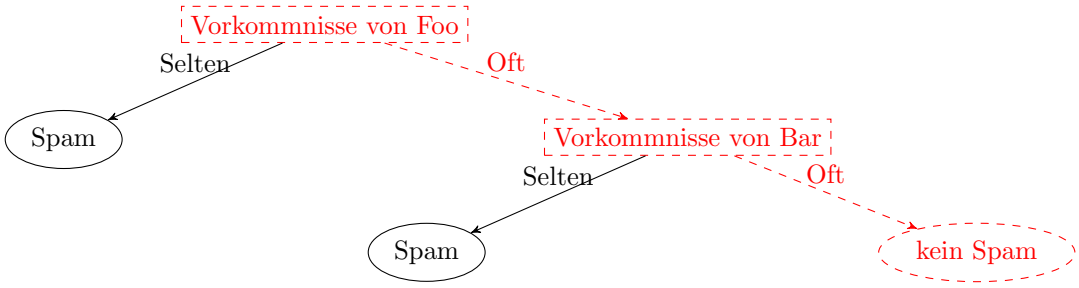

Figure 1: Möglicher Entscheidungsbaum, Klassifizierung von ,,Foo → Oft, Bar → Oft"

$$evaluate(s, (A_i, f)) = evaluate(s, f(s(A_i))) \quad (1)$$
$$evaluate(s, K_i) = K_i \quad (2)$$

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 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 1 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.

Definition: Attribut
Wir definieren eine Menge von Wahrscheinlichkeiten $P = [0, 1] \subset \mathbb{R}$ und eine Menge von Buchstaben $\Sigma = \{a, b, \ldots, z, A, B, \ldots, Z\}$. 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 boolsche 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 $\{i_1, i_2, \ldots, i_n\}$ 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 Verschlueselung, dann ist das daraus entstehende Element der entstellten 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

Wir nehmen im Folgenden an, dass die Anwender ehrlich sind. Das bedeutet, dass die Anwender zwar versuchen, aus den Informationen, die sie im Protokoll erhalten, möglichst viel zu lernen, allerdings werden sie sich an das Protokoll halten. (d.h., eine Annahme der Form ,,Jetzt sendet Alice diesen Wert erneut"

funktioniert).

2 Grundlagen der Anwendung

Gegeben diese Begrifflichkeiten ist dann die Vision der Anwendung, dass aus den gesamten E-Mails ein gemeinsamer Klassifikator erzeugt wird, wobei über die eigenen E-Mails jedes Anwenders möglichst wenig preisgeben möchte. Genauer gesagt, es soll nur der Klassifikator über die eigenen E-Mails preisgegeben werden.

2.1 Form der Benutzereingabe

Da wir nicht mit weiteren Informationen der E-Mail arbeiten, sondern nur mit dem textuellen Inhalt, wählen wir als Eingabeform der E-Mails einfache Textdateien. Um die E-Mails einfach nach Spam und Nicht Spam organisieren zu können, verlangt die Anwendung, dass in einem Verzeichnis, aus dem wir E-Mails einlesen sollen, zwei Verzeichnisse ,,spam" und ,,not_spam" existieren, welche dann die Dateien mit den E-Mail-Inhalten enthalten. Weitere Unterverzeichnisse in irgendeinem beteiligten Verzeichnis werden von der Anwendung ignoriert. Dadurch ist es nicht notwendig, die Metadaten aus einer separaten Datei einzulesen (und diese separate Datei auf dem neuesten Stand zu halten), sondern die Existenz einer Datei in einem der beiden Verzeichnisse schreibt sofort die Klassifkation der Datei vor.

Der Anwender kann dann über einen Parameter steuern, ob der Klassifikator auf der Standardausgabe ausgegeben wird oder in welche Datei der Klassifikator geschrieben werden soll.

2.2 Interaktion der verteilten Programme

Wir implementieren die Kommunikation der verteilten Programme dadurch, dass wir das Programm in zwei verschiedenen Modi ausführbar machen: Als Client und als Server. Der Server wird dann im Allgemeinen die Aufgabe von Bob erfüllen und auf Anfragen des Clients antworten, der dann natürlich die Rolle von Alice einnimmt. Der Client wird dann zudem die IP und den Port des Server vom Anwender erfahren, um mit dem Server kommunizieren zu können.

2.3 Übertragung der Gesamtanzahl der E-Mails

Aus technischen Gründen wird zu Anfang der Anwendung von jedem der beiden Anwender die Gesamtanzahl der eigenen E-Mails übertragen. Dies stellt kein Risiko für die Geheimhaltung dar, da die Anzahl der E-Mails keine Aussage über den Inhalt der E-Mails preisgibt.

2.4 Phasen der Anwendung

Die Anwendung durchläuft bei der Berechnung mehrere Phasen, die aufeinander aufbauen. Der Ablauf dieser Phasen ist in Abbildung 2 illustriert. In der ersten Phase bererechnen beide Anwender die gemeinsame Wortliste, aus denen dann

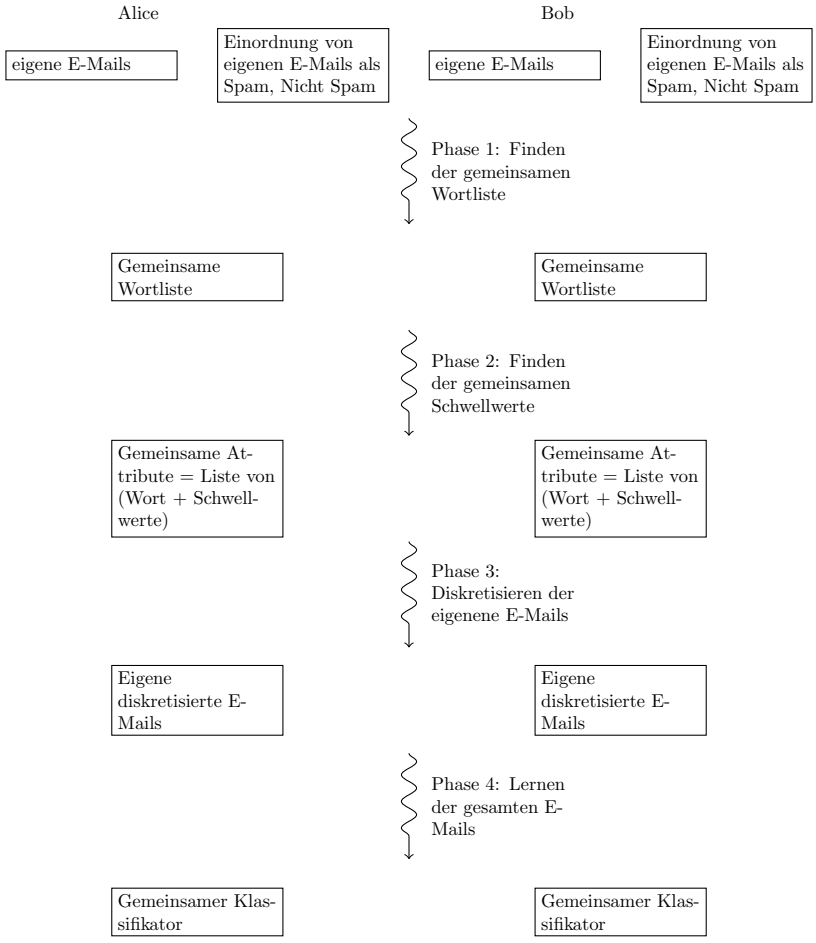

Figure 2: Phasen der Anwendung

in einer zweiten Phase Schwellwerte und damit Attribute für einen Entscheidungsbaum berechnet werden. In einer dritten Phase wandeln dann beide Anwender ihre eigenen E-Mails in die Attributwert-vektoren um und berechnen dann in einer finalen Phase den gemeinsamen Klassifikator.

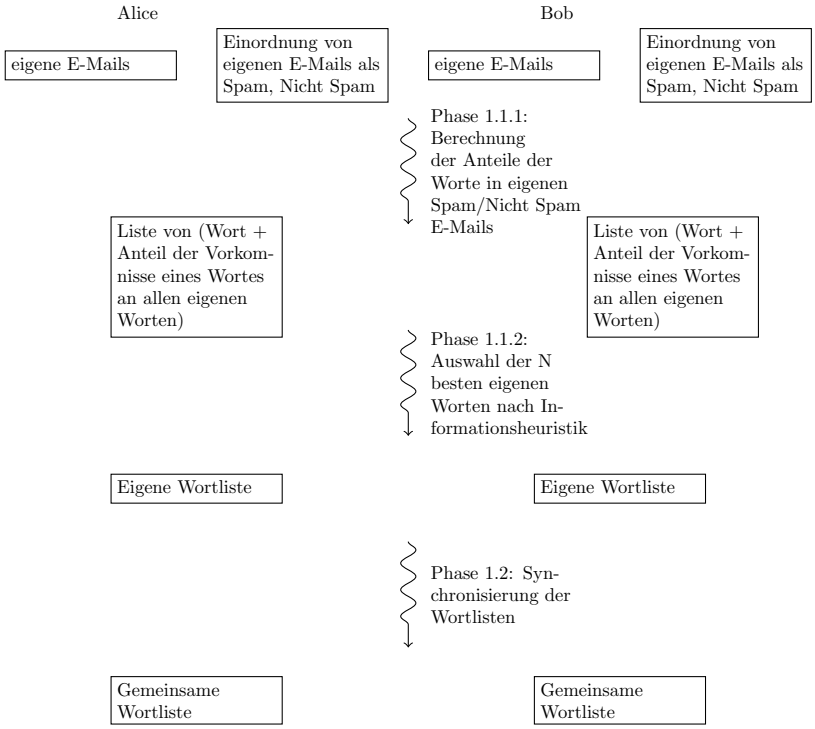

Figure 3: Ablauf der Phase zum Berechnen der gemeinsamen Wortliste

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 hat 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 3 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 aussagekräftigstenen 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 Vorkomnissen gegeben sind:

Wort	Vorkomnisse 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.

Abschliessend ergibt sich somit das Akzeptanzkriterium:
Requirement 4: Das Protokoll, in dem beide Anwender ihre Wortlisten einander zusenden und lokal die spezifizierte Syncronisierung durchführen, ist implementiert

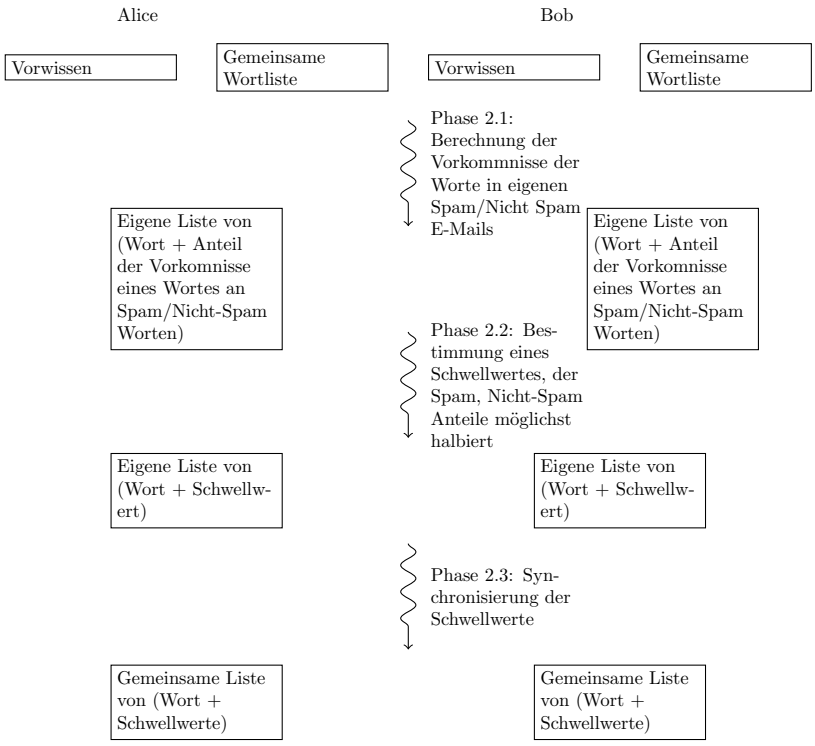

Figure 4: Ablauf der Phase zum finden der gemeinsamen Schwellwerte

4 Finden der gemeinsamen Schwellwerte

In dieser Phase müssen wir für jedes Wort in der gemeinsamen Wortliste Schwellwerte finden, wann das Wort besonders oft oder besonders selten vorkommt. Jeder Anwender berechnet dazu zunächst einen eigenen Schwellwert für jedes Wort und in einem zweiten Schritt werden diese eigenen Schwellwerte mit den jeweils anderen Schwellwerten vereint, um die gemeinsamen Schwellwerte und somit die gemeinsamen Attribute zu berechnen. Diese Schritte sind in Abbildung 4 illustriert.

4.1 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 5: 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.

4.2 Syncronisierung der Schwellwerte

Wir haben nun von Alice und Bob jeweils einen Schwellwert a und einen Schwellwert b, sodass Alice der Meinung ist, dass ein Wort w selten vorkommt, wenn der Anteil an Vorkomnissen unter a liegt und Bob der Meinung ist, dass das Wort selten vorkommt, wenn der Anteil an Vorkommnissen unter b liegt. Wir syncronisieren dann die Werte zu einem aufsteigend sortierten Paar von schwellwerten (a, b) bzw (b, a). oBdA sei $a \leq b$. Dann gilt, dass bei weniger Vorkommnissen als a beide Anwender sich einig sind, dass das Wort selten vorkommt, während bei mehr Vorkommnissen als b beide Anwender sich einig sind, dass das Wort oft vorkommt. Dazwischen herrscht uneinigkeit und es wird angenommen, dass die weiter Klassifikation durch einen Entscheidungsbaum diese Uneinigkeit auflösen kann.
Damit ergeben sich als Akzeptanzkriterien:
Requirement 6: Wenn a = 0.2 ist und b = 0.4, dann ist das Schwellwertpaar (0.2, 0.4)

Requirement 7: Wenn a = 0.5 ist und b = 0.4, dann ist das Schwellwertpaar (0,4, 0.5)

Requirement 8: Wenn a = b = 0.5 ist, dann ist das resultierende Schwellwertpaar (0.5, 0.5)

Da die Attribute am Ende öffentlich bekannt sind, implementieren wir dies, indem sich beide Anwender die eigenen Schwellwerte für die Worte zuschicken und dann lokal die Syncronisierung berechnen. Somit ergibt sich noch das abschliessende Akzeptanzkriterium:
Requirement 9: Das Protokoll, indem beide Anwender ihre eigenen Schwellwerte an den jeweils anderen Anwender senden und dann die bereits spezifizierte Merge-Operation ausführen ist implementiert.

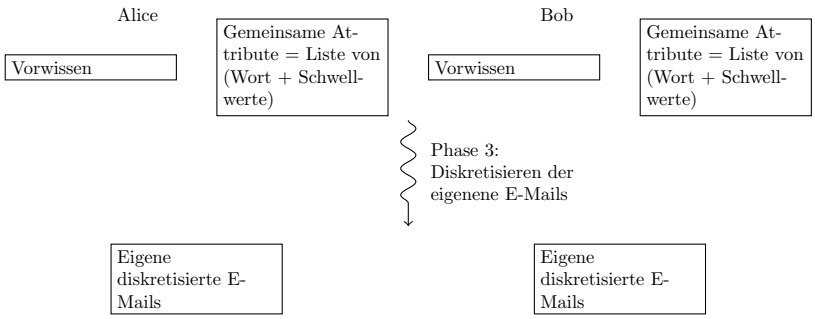

Figure 5: Ablauf der Phase zum diskretisieren der eigenen E-Mails

5 Diskretisieren der eigenen E-Mails

In dieser Phase muss die Anwendung die E-Mail-Inhalte, d.h., Texte ind die Attributvektoren umrechnen, auf denen der ID3-Algorithmus operiert. Dazu muss für einen E-Mail-Inhalt für jedes Wort in den Attributen der Anteil an den Worten in dem gesamten E-Mail-Inhalt berechnet werden und dann anhand des Schwellwertpaares als selten, oft oder mittel klassifiziert werden. Dies ist in Abbildung 5 illustriert. Da dies eine relativ einfache und vor allem lokale Phase ist, werden hier nur die Anforderungen aufgelistet:
Requirement 10: Wenn der E-Mail-Inhalt "A A A A B B C D" ist und die Attribute sind (A, 0.2, 0.3), (B, 0.1, 0.9), (C, 0.5, 0.8), dann wird diese E-Mail diskretisiert in den Vektor oft, mittel, selten

6 Lernen der gesamten E-Mails

Das verteilte Lernen der gesamten E-Mails ist bei weitem der komplexeste Schritt in der Anwendung. Auf einer sehr abstrakten Ebene ist dieser Schritt jedoch eine direkte Umsetzung des ID3-Algorithmus' auf den privaten, verteilten Fall. Das bedeutet, der Algorithmus arbeitet rekursiv auf einer Menge von Daten und einer Menge von verbleibenden Attributen und unterscheidet dann drei Fälle:

- Es gibt keine weiteren Attribute. In diesem Fall wird die ein Blatt mit der dominierenden Ausgabe erzeugt Dies ist in Abbildung 7 illustriert und in Sektion 6.2 genauer spezifiziert.
- Alle Datensätze sind sich bezüglich der Ausgabe einig. In diesem Fall wird ein Blattknoten mit der eindeutigen Ausgabe erzeugt. Dies ist in Abbildung 9 illustriert und in Sektion 6.3 genauer spezifiziert.
- Es herrscht noch Uneinigkeit bezüglich der Ausgabe und es gibt noch weitere verwendbare Attribute. In diesem Fall muss die Datenmenge anhand der Attributwerte eines Attributes partitioniert werden, sodass der Informationsgewinn maximal ist und die partitionen dann rekursiv behandelt werden. Dies ist in Abbildung 10 illustiert und in Sektion 6.5 genauer spezifiziert.

6.1 Yaos Protokoll

Generelles Vorgehen Der grobe Ablauf von Yaos Protokoll für die private zwei-Parteien Berechnung ist in Abbidlung 6 skizziert und basiert auf [3]. 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 Beeintraechtigung der Allgemeinheit Alice, muss nun aus diesem Schaltkreis einen enstellten 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 entstellten 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.

Verschleierung der Wahrheitstabellen 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 Markierung-

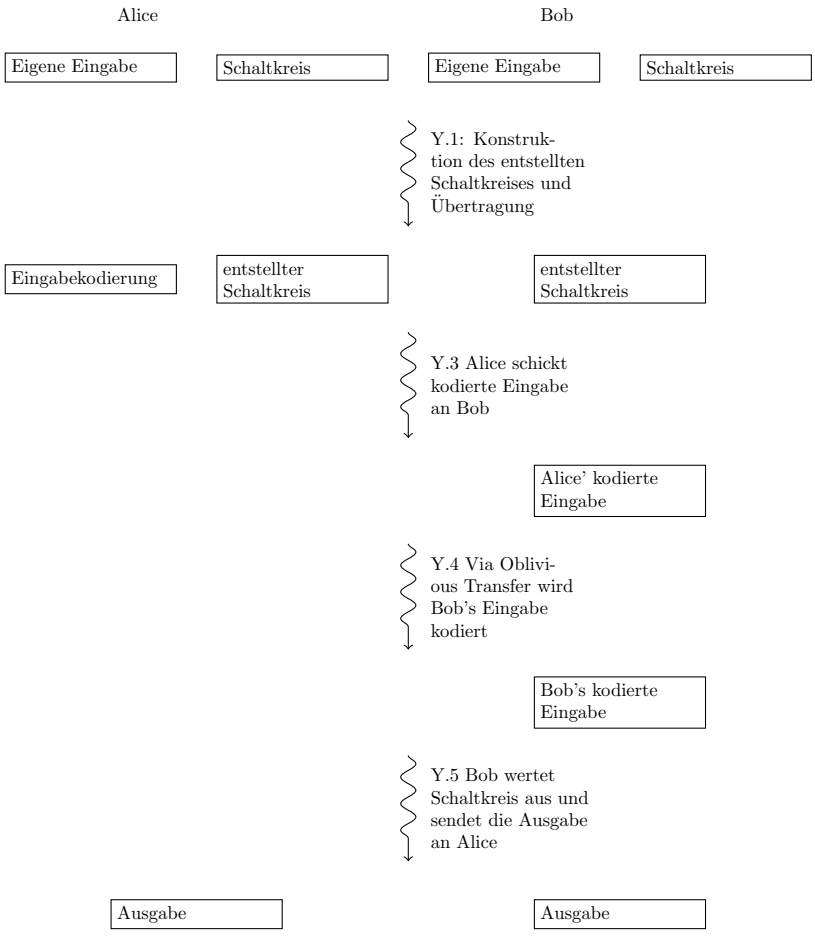

Figure 6: Ablauf von Yaos Protokoll

seintrag. 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.

Belegung der Eingangskanten

Alice Eingabe 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.

Bobs 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 $v = x_b + k' mod N$ 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

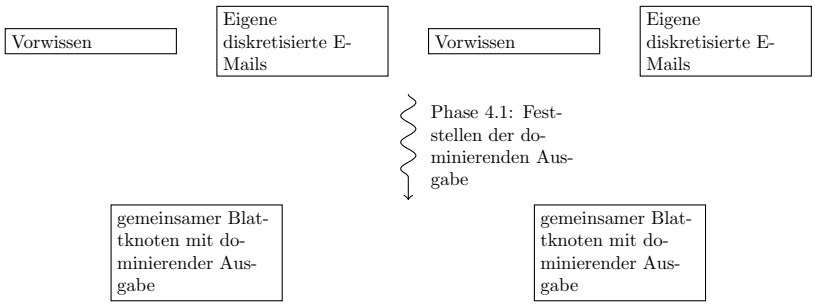

Figure 7: ID3-Algorithmus, Fall 1: Keine Attribute mehr vorhanden

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

Da die Menge der Attribute öffentlich ist und die Menge der bereits verwendeten Attribute ebenfalls öffentlich ist, können beide Parteien erkennen, dass sie nun einen Blattknoten mit der dominierenden Ausgabe konstruieren müssen. Dazu ist es notwendig, zu erkennen, ob in den verbleibenden E-Mails mehr Spam-E-Mails oder mehr Nicht-Spam-E-Mails vorhanden sind. Dazu muss ein Schaltkreis designed werden, welcher als Eingabe zwei Paare von Zahlen erhählt und als Ausgabe den Index der maximalen Summe liefert. Da hierzu die Summe gebildet werden muss, muss hierzu die verwendete Bitbreite bestimmt werden. Da jedoch beide Anwender die Anzahlen der von ihnen verwendeten E-Mail-Mengen preisgegeben haben, kann jedoch die Summe der verbleibenden Teilmengen abgeschätzt werden durch die Summe der Anzahlen der gesamten E-Mails und somit kann die geasmte Bitbreite der zu addierenden Zahlen abgeschätzt werden als Logarithmus dieser Gesamtanzahl.
Dadurch verbleibt es dann, einen Schaltkreis zu designen, welcher zwei Zahlen bekannter Bitbreite addiert und 0 bzw 1 ausgibt, wenn die erste bzw zweite Zahl grösser ist. Da Effizienz erst einmal sekundär ist, kann die Addition der Zahlen durch einen einfachen Riple-Carry-Adder vollzogen werden.
Die Maximumsbestimmung von zwei binären Zahlen ist dann die Frage, welche der beiden Zahlen die hoechstwertige 1 hat. Wenn wir zudem definieren, dass die Ausgabe 0 bedeutet, dass die erste Zahl in den Paaren die kleinere ist und die Ausgabe 1 die zweite, dann vereinfacht sich die Maximumsbestimmung noch weiter zum finden der höchstwertigen Stelle, die sich unterscheidet und dem zurückgeben des Bits in der ersten Zahl. Dies ist in Abbildung 8 angedeutet.
Formal gilt also: $lt(a,b) = b_i \Leftrightarrow a_i \neq b_i \wedge \forall k = i+1, \ldots, n : a_k = b_k$ Der Allquantor ist als Kaskade von Und-Gattern implementierbar und die Gleichheit bzw Ungleichheit als Negation bzw direkte Verwendung des XOR-Gatters implementierbar, damit kann diese Formel einfach in einen Schaltkreis überführt

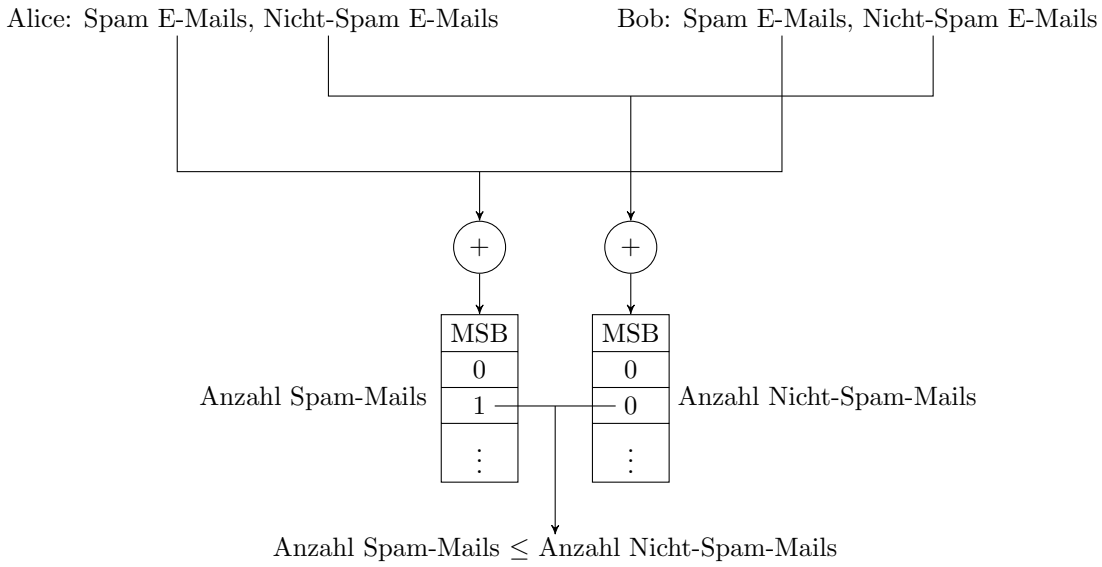

Figure 8: High-Level Struktur des Dominierende-Ausgabe-Schaltkreises

werden.

Requirement 11: Dieser Schaltkeris für die dominierende Ausgabe kann generiert werden

6.3 Feststellen ob Ausgabe eindeutig

Es muss ein Schaltkreis designed werden, der Feststellt, ob alle Elemente einer Menge eindeutig sind und dann entweder das Klassenlabel oder ein Fehlersymbol ausgibt. Die Eingabe ist somit für jede E-Mail eine Kodierung für Spam, Nicht Spam oder Abwesenheit, die Anzahl der Eingaben ist bekannt und durch die Gesamtanzahl der E-Mails beschränkt.

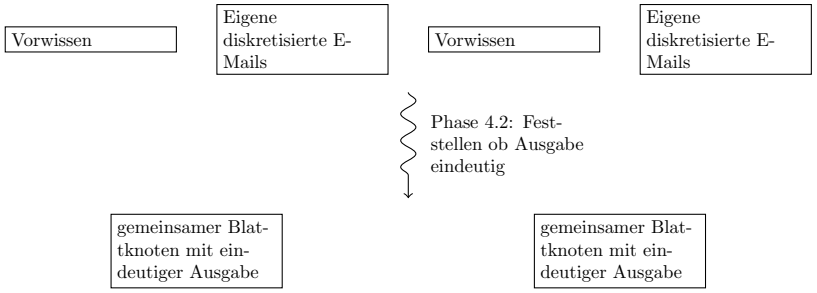

Figure 9: ID3-Algorithmus, Fall 2: Ausgabe eindeutig

Wir implementieren diesen Schaltkreis dann als Zustandsmaschine, die die internen Zustände ,,Alles bis hier Spam", ,,Alles bis hier kein Spam" und ,,Bereits uneindeutig" hat. Wenn dann T: State $\times$ Mailtype ist, muss folgende Zustandsübergangstabelle implementiert werden:

T	Alles bis hier Spam	Alles bis hier kein Spam	Bereits uneindeutig
Spam	Alles bis hier Spam	Uneindeutig	Bereits Uneindeutig
Nicht Spam	Uneindeutig	Alles bis hier kein Spam	Bereits Uneindeutig
Abwesenheit	Alles bis hier Spam	Alles bis hier kein Spam	Bereits uneindeutig

Zudem wäre es sehr angenehm, wenn die Kodierung von Spam und ,,Alles bis hier Spam" und von ,,Nicht Spam" und ,,Alles bis hier kein Spam" gleich sind, da dies den Anfangsfall vereinfacht.
Wenn wir nun ,,Bereits uneindeutig" durch 11 kodieren und ,,Alles bis hier Spam" mit 01 kodieren und ,,Alles bis hier kein Spam" mit 00 kodieren und davon ausgehend ,,Spam" als 01 kodieren und ,,Nicht Spam" als 00 und ,,Abwesenheit" als 11, dann muss folgende Zustandsübergangstabelle gelten:

T	01	00	11
01	01	11	11
00	11	00	11
11	01	00	11

Durch Anwenden von KV-Diagrammen ergeben sich als Zustandsübergangsgleichungen somit, mit s_0, s_1 als Bits des vorherigen Zustandes, e_0, e_1 als Bits der E-Mail Kodierung und s'_0, s'_1 als Bits des neuen Zustandes:

$$\begin{aligned} s'_0 &= s_0 \vee \overline{s_1 e_0} e_1 \vee s_1 \overline{e_0 e_1} \\ s'_1 &= s_1 \vee \overline{e_0} e_1 \end{aligned}$$

Der Anfangszustand des Automatens ist die Kodierung der Klasse der ersten E-Mail. Es muss dann nur noch für jede E-Mail das beschriebene Zustandsübergangsnetz von den vorherigen Zuständen und der Kodierung der Klasse der aktuellen E-Mail implementiert werden. Das Fehlersymbol, das ausgegeben wird, ist 11, andernfalls ist die Klasse im zweiten Ausgabebit Kodiert, wobei Spam als 1 kodiert ist und Nicht Spam als 0.

6.4 Das Entropien-Protokoll

Es muss in diesem Teilschritt für jeden Anwender ein Share von $x \cdot \ln x$ berechnet werden, sodass die Summe der Shares ca $x \cdot \ln x$ ist. Um den ersten Schritt mit Yaos Algorithmus zu berechnen, müssen wir einige Teilprobleme lösen. Es muss zum einen ein Weg gefunden werden, durch den man mit Yaos Protokoll für beide Anwender verschiedene Ausgaben produziert werden können und zum zweiten muss ein Weg gefunden werden, eine Zahl in zwei Shares zu zerlegen, sodass die Summe der Shares wieder die Zahl ist.

6.4.1 Erweiterung von Yaos Protokoll: Separate Ausgaben

Betrachten wir das erste Teilproblem, separate Ausgaben für die Anwender zu erzeugen. Dies ist ein Problem, da Yaos Protokoll fuer die private zwei Parteien berechnung einer Funktion eben eine Funktion berechnet, d.h. genau einen Wert, der dann beiden Parteien bekannt gegeben wird. Um dies zu lösen, nehmen wir an, dass i und j die Eingaben von Alice und Bob sind und der evaluierte Schaltkreis $f(i, j) = A\&B$ berechnet, wobei & die Konkatenation von Binärsequenzen bezeichne. Falls die Ausgabe nicht derartig sortiert ist, muss dies in einem separaten Schaltkreis vor der Ausgabe passieren. Wir erweitern dann die Eingabe i von Alice um $\|A\|$ Zufallsbits zu $i\&C_A$ und analog die Eingabe j von Bob um $\|B\|$ Zufallsbits zu $j\&C_B$ und erweitern den Schaltkreis am Ende, sodass er $f'(i\&C_A, j\&C_B) = A \oplus C_A\&B \oplus C_B$ berechnet, wobei $\oplus$ das bitweise XOR von zwei Binärsequenzen beschreibe. Da C_A und C_B im eigentlichen Schaltkreis zur Berechnung von A und B nicht verwendet werden und zufällig sind und dem jeweils anderen Anwender unbekannt sind, ist dies also äquivalent zu einer One-Time-Pad-Verschlüsselung der Ergebnisse mit einem Schüssel, den nur der Anwender kennt, der diese Ausgabe wisssen soll.
Requirement 12: Es ist möglich, einen Schaltkreis so zu erweitern

6.4.2 Erweiterung von Yaos Protokoll: Zerlegung der Ausgabe in zufällige Shares

Die Problemstellung Zum zweiten muss ein binärkodierter Integer S in zwei zufällige Shares S_A, S_B zerlegt werden, sodass $S_A + S_B = S$ ist. Da unsere Schaltkreise deterministisch sind, bedeutet das, dass wir Zufallsbits als Eingabe hinzugeben müssen, um eine zufällige Aufteilung zu erreichen. Wir lösen dieses Problem dann so, dass wir einen Verteilungsvektor V berechnen mit $\|V\| = \|S\|$ und kopieren das Bit i in Alice' Share S_A, falls $V^i = 1$ ist, andernfalls kopieren wir das Bit i in Bobs Share S_B. Das Kopieren kann implementiert werden, indem $S_A^i = S^i \wedge V^i$ und $S_B^i = S^i \wedge \overline{V^i}$ implementiert wird.

Unsere Lösung Die Summe der beiden Shares ist dann wieder S, weil die Shares den Wert von S als zwei disjunkte Mengen von Zweierpotenzen darstellen, sodass die Addition keinen Carry erfordert, sondern einfach eine Veroderung der beiden Shares ist. Somit wird S wieder entstehen, wenn jedes Bit in S_A oder in S_B zu finden ist. Das ist nach Konstruktion aequivalent dazu, dass jedes Bit in V entweder 0 oder 1 ist. Bleibt die Frage, wieviele Informationen aus dem Share gewonnen werden kann. Es gilt, dass eine 0 eine Unsicherheitsquelle ist, da die 0 entweder eine 0 sein kann, die enstanden ist, weil der Verteilungsvektor an dieser Position ungünstig war oder weil wirklich im Share eine 0 war. Wenn wir eine 1 sehen, wissen wir auf jeden Fall, dass in S an dieser Position 1 war. Da wir jedoch als Menge aufzuteilender Werte zum einen den Wert $2^N \cdot n \cdot \ln 2$ haben für eine Konstante N und zum anderen $\epsilon \cdot 2^N$ für ein beiden Parteien unbekanntes Epsilon, sind die vorkommenden Bitmuster so überlappend, dass

nur aus der Position einiger 1en nicht geschlossen werden kann, welcher Wert aufgetreten ist. Wenn zudem der Verteilungsvektor unbekannt ist, ist somit aus dem Share auf jeden Fall nicht ausreichend Informationen abzuleiten, um das Geheimnis zu lernen.

Berechnung des Verteilungsvektors Es bleibt nun V zu berechnen. Da die Bitbreite von S abgeschätzt werden kann, definieren wir, dass Alice und Bob zwei weitere Vektoren V_A und V_B eingeben und definieren $V = V_A \oplus V_B$. Dadurch kann keiner der beiden Anwender alleine den Wert von V zu seinem Vorteil beeinflussen.
Im $x \cdot \ln x$ Entropienprotokoll wird nun eine Taylorreihe aufgestellt, anhand derer festgestellt werden kann, das zwei Werte berechnet werden müssen: $2^N \cdot n \cdot \ln 2$ und $\epsilon \cdot 2^N$, wobei x die Summe der Anwendereingaben ist und $x = 2^n \cdot (1+\epsilon)$ mit $-\frac{1}{2} \leq \epsilon \leq \frac{1}{2}$ und N eine obere Schranke für n ist. Der erste Wert ist kein Integer, kann jeoch auf einen Integer gerundet werden, ohne dass die Genauigkeit einbricht, da er nur als Anfangsannäherung dient, der zweite Wert ist ein Integer.
Requirement 13: Es ist möglich, einen Schaltkreis so zu erweitern

6.4.3 Berechnung der Anfangsapproximation

Der erste Wert wird in einem Schaltkreis berechnet, indem das n berechnet wird und die möglichen Werte des Ausdruckes in einer hardkodierten Tabelle nachgesehen werden. Das n wird berechnet, indem der Index des signifikantesten Bits von x berechnet wird. Der zweite Ausdruck wird berechnet, indem $\epsilon \cdot 2^N = 2^(N-n) \cdot \epsilon \cdot 2^n = 2^(N-n) \cdot (x - 2^n)$ festgestellt wird. Das bedeutet, man muss das signifikanteste Bit von x löschen und das Ergebnis davon um N-n bit nach links shiften. Der Wert von N wird vorher berechnet und hardkodiert.Die Lookuptable und der parametrierte Shift werden wieder über vollständige Suchen implementiert, da so nur der (relativ einfache) Vergleich implementiert werden muss. Danach sind diese beiden Werte berechnet und ausgehend von der Vorüberlegung muss noch die Aufteilung beider Werte in Shares sowie die Verschleierung der Ausgaben für die beiden Anwender orthogonal hinzugefügt werden.
Requirement 14: Dieser Schaltkreis zur Berechnung der Shares der ersten Approximation kann generiert werden

6.4.4 Verbesserung der ersten Approximation

Vorgehensweise Um dann die Anfangsapproximation zu verbessern, wird die Taylorreihe bis auf das k. Element berechnet, wobei k ein Praezisionsparameter ist. [1] wählt hierzu eine zufällige Zahl z_1 und definiert das Polynom:

$$Q(z) = \mathrm{lcm}(2, \ldots, k) \cdot \sum_{i=1}^{k} \frac{(-1)^{i-1}}{2^{N \cdot (i-1)}} \cdot \frac{(\alpha_1 + z)^i}{i} - z_1 \tag{3}$$

Für die Implementierung wird der binomische Term aufgelöst und die Formel etwas umgestell:

$$Q(z) = \text{lcm}(2,\ldots,k) \cdot \sum_{i=1}^{k} \sum_{x=1}^{i} \frac{(-1)^{i-1}}{2^{N\cdot(i-1)} \cdot i} \cdot \binom{i}{x} \cdot \alpha_1^{i-x} \cdot z^x - z_1 \qquad (4)$$

In dieser Form kann prinzipiell ein Array mit den Koeffizienten als Einträge verwendet werden und immer an der Stelle x wird der entsprechende Faktor addiert.

Geheime Auswertung von Q

Die Problemstellung Die Auswertung von Q im Punkt α_2 ergibt dann ein zweites Share z_2 mit der Eigenschaft, dass $z_1 + z_2 + lcm(2,\ldots,k) \cdot (\beta_1 + \beta_2) \approx lcm(2,\ldots,k) \cdot 2^N \cdot \ln x$. Die Gleichheit gilt nur ungefähr, da die Taylorapproximation nur teilweise berechnet wurden, jedoch kann die Genauigkeit durch den Parameter k (zu Lasten der Laufzeit, da k der Grad des auszuwertenden Polynoms ist) beliebig verbessert werden.
Da Q einen geheimen Share von Alice enthählt und α_2 ein geheimer Share von Bob ist, brauchen wir einen Algorithmus zur geheimen Auswertung von Polynomen, sodass Bob zwar $Q(\alpha_2)$ lernt, aber nicht Q und dass Alice nichts über α_2 erfährt.

Das verwendete Protokoll Das dazu verwendete Protokoll basiert auf [4] arbeitet wie folgt: Alice wählt ein zufälliges bivariates Polynom $R(x, y)$ mit den Eigenschaften: (1) Der Grad d_y von y ist der Grad von Q (2) $R(0, \cdot) = Q(\cdot)$, (3) Der Grad von x in Q ist ein Parameter $d_{R,x}$. So ein Polynom kann gewählt werden, indem wir mit dem Polynom Q anfangen und dann Zufallskoeffizienten für Terme, die den Parameter x enthalten, wählen, bis der Grad in x gross genug ist. Dadurch kann durch setzen von x = 0 direkt das Polynom Q rekonstruiert werden, da nur die Faktoren aus Q keinen Anteil am x haben. Wenn wir desweiteren keine zu grossen Potenzen von y einfügen ist die 1. Bedingung ebenfalls direkt erfüllt. Die dritte Bedingung ist nach Konstruktion erfüllt.
Bob wählt ein weiteres univariates Polynom S mit der Eigenschaft, dass $S(0) = \alpha_2$ ist und der Grad des Polynoms ist $\frac{d_{R,x}}{d_Q}$, wobei d_Q der Grad von Q ist. Dies kann konstruiert werden, indem der konstante Faktor des Polynoms auf α_2 gesetzt wird und die restlichen Faktoren zufällig gewählt werden.

Der Plan ist nun, ein Polynom $T(0) = R(0, S(0)) = P(S(0)) = P(\alpha_2)$ auszuwerten, damit Bob den gesuchten Wert lernen kann. um T rekonstruieren zu können muss Bob dementsprechend $d_T + 1 = d_{R,x} + d_Q \cdot d_s + 1 = 2 \cdot d_{R,x} + 1$ Punkte von T erfahren, um T zu rekonstruieren und $T(0)$ auswerden zu können. (Die verschiedenen d_p bzw $d_{p,v}$ bezeichnen den Grad des Polynoms p, eventuell in der Variablen v). Um dies geheim durchzuführen, wählt Bob $n = 2 \cdot d_{R,x} + 1$ verschiedene Punkte verschieden von 0. Für jeden Punkt x_i dieser Punkte sendet

Bob eine Liste von m-1 zufälligen Punkten und dem Wert $S(x_i)$ in zufälliger Reihenfolge an Alice sowie den Punkt x_i selbst. Via 1-m-Oblivious-Transfer kann nun Alice den Wert $R(x_i, \cdot)$ für jeden der zufälligen Punkte anbieten und Bob kann – da er die Reihenfolge der Punkte und damit die Position von $S(x_i)$ kennt – den Wert $R(x_i, S(x_i))$ lernen. Alice lernt dabei die Position nicht und deswegen den Wert $S(x_i)$ nicht und Bob erfährt keinen der anderen Werte. Danach interpoliert Bob T und berechnet $T(0)$ **Requirement 15:** Dieses Protokoll zur Polynomauswertung kann ausgeführt werden.

1-out-of-N Oblivious Transfer

Die Problemstellung Es verbleibt nun ein Protokoll für den 1-out-of-N Oblivious Transfer zu wählen. Um zu rekapitulieren, Alice kennt N Werte $I_A^1, I_A^2, \ldots, I_A^N \in \{0,1\}^m$ und Bob kennt einen Index i_B. Das Ziel des Protokolles ist nun, dass Bob $I_A^{i_B}$ lernt, ohne dass Bob I_A^j für $j \neq i_B$ lernt und ohne dass Alice i_B lernt.

Das verwendete Protokoll Es sei $\{F_k : \{0,1\}^m \mapsto \{0,1\}^m\} | K \in \{0,1\}^t$ eine Familie von pseudozufälligen Funktionen. Wir verwenden nun diese Funktionen und 1-out-of-2-Oblivious Transfer (für den bereits eine Spezifikation existiert), um die 1-out-of-N Oblivious Transfer zu implementieren.

Alice berechnet nun l Schlüsselpaare $(K_1^0, K_1^1), (K_2^0, K_2^1), \ldots, (K_l^0, K_l^1))$ mit $N = 2^l$, sodass alle Schlüssel Schlüssel für die pseudo-zufällige Funktion F sind. Desweiteren berechnet Alice $Y_s = i_A^s \oplus \bigoplus_{j=1}^{l} F_{K_j^{s \rhd j}}(s)$, wobei $s \rhd j$ das Bit mit Index j von s ist. Bob wählt nun aus jedem der l Paare via 1-out-of-2-Oblivious-Transfer eines aus. Wenn Bob den Wert mit Index i_B lernen will, dann muss Bob im Schritt j den Eintrag $i_B \rhd j$ wählen. Desweiteren sendet Alice die Strings Y_s an Bob. Dann ist Bob in der Lage, $i_A^{i_B}$ zu rekonstruieren, indem er $Y_{i_B} \oplus \bigoplus_{j=1}^{l} F_{K_j}^{i_B \rhd j}(i_B)$ berechnet, denn:

$$
\begin{aligned}
& & Y_{i_B} \oplus \bigoplus_{j=1}^{l} F_{K_j}^{i_B \rhd j}(i_B) \\
= & & i_A^{i_B} \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_B \rhd j}}(i_B) \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_B \rhd j}}(i_B) \\
= & & i_A^{i_B}
\end{aligned}
$$

Requirement 16: Dieses Protokoll zum 1-out-of-N-Oblivious Transfer kann ausgeführt werden.

Die Familie pseudozufälliger Funktionen Es verbleibt die Familie F_k von Funktionen zu spezifizieren. Ausgehend von [2] funktioniert dies wie folgt:

Wir wählen einen kryptographisch sicheren Zufallsbit-generator G, der einen Seed der Länge k auf $2 \cdot k$ Bit erweitert. Wir bezeichnen dann mit $G^0(k)$ die erste Hälfte von $G(k)$ und mit $G^1(k)$ die zweite Hälfte von $G(k)$. Ist nun i ein Bitstring der Länge l, dann definieren wir $G^i(x) = G^{i \triangleright 1}(G^{i \triangleright 2}(\dots G^{i \triangleright l}(x) \dots))$, d.h. der Zufallszahlengenerator wird immer wieder aufgerufen und es wird in der ,,richtigen" Hälfte weitergerechnet. Wir definieren dann eine Familie von Funktionen $\{F_k\}$ wobei wir $F_k(x) = G_x(k)$ definieren.
Requirement 17: Es ist so eine Funktionsfamilie implementiert.

Um zu rekapitulieren, Alice und Bob kennen nun beide Shares β_1, z_1 bzw β_2, z_2, sodass $(z_1 + lcm(2, \dots, k) \cdot \beta_1) + (z_2 + lcm(2, \dots, k) \cdot \beta_2) = lcm(2, \dots, k) \cdot 2^N \cdot \ln x$ ist.

6.4.5 Private Multiplikation

Die Problemstellung Wir definieren nun ein Protokoll, welches für die Eingaben i_A, i_B zufällige Shares o_A, o_B produziert, sodass $o_A + o_B = i_A \cdot i_B$ gilt. Dadurch kann $x \cdot \ln x = E_A \cdot \ln(E_A + E_B) + E_B \cdot \ln(E_A + E_B)$ berechnet werden, wenn E_A die Anzahl E-Mails von Alice ist und E_B die Anzahl E-Mails von Bob, was insgesamt dann die gesuchte Entropie ist.

Das verwendete Protokoll Alice definiert das Polynom $Q(z) = i_A \cdot z + r_A$ für einen zufällig gewählten Wert r_A. Beide wenden nun die geheime Polynomauswertung an, sodass Bob $Q(i_B)$ berechnet. Wir definieren dann $o_A = r_A$ und $o_B = Q(i_B)$. Es gilt dann direkt, dass $o_A + o_B = r_A + i_A \cdot i_B + r_A = i_A \cdot i_B$ ist und o_A, o_B sind wirklich zufällig und geheim, da zum Berechnen von (beispielsweise) o_B aus i_A und o_A eine Gleichung mit zwei Unbekannten $Q(a_2), a_2$ gelöst werden müsste, was nicht eindeutig möglich ist und der Privatheit der Polynomauswertung.
Requirement 18: Dieses Protokoll zur Multiplikation kann ausgeführt werden

6.4.6 Das gesamte Protokoll

Das komplette Protokoll zum berechnen für $x \cdot \ln x$ mit Eingaben i_A, i_B und $x = i_A + i_B$ besteht nun aus drei Phasen: Wir berechnen zuerst die Shares l_A, l_B von $\ln x$ nach dem ersten Teilprotokoll. Wir wenden dann das zweite Teilprotokoll zur Multiplikation zweimal an, einmal auf die Eingabe l_A, i_B und einmal auf der Eingabe l_B, i_A und erhalten die Shares m_A^1, m_A^2, m_B^1 und m_B^2 mit $m_A^1 + m_B^1 = l_A \cdot i_B$ und $m_A^2 + m_B^2 = l_B \cdot i_A$. Wir definieren dann Alice Ausgabe des gesamten $x \cdot \ln x$-Protokolls als $o_A = m_A^1 + m_A^2 + l_A \cdot i_A$ und Bobs Ausgabe

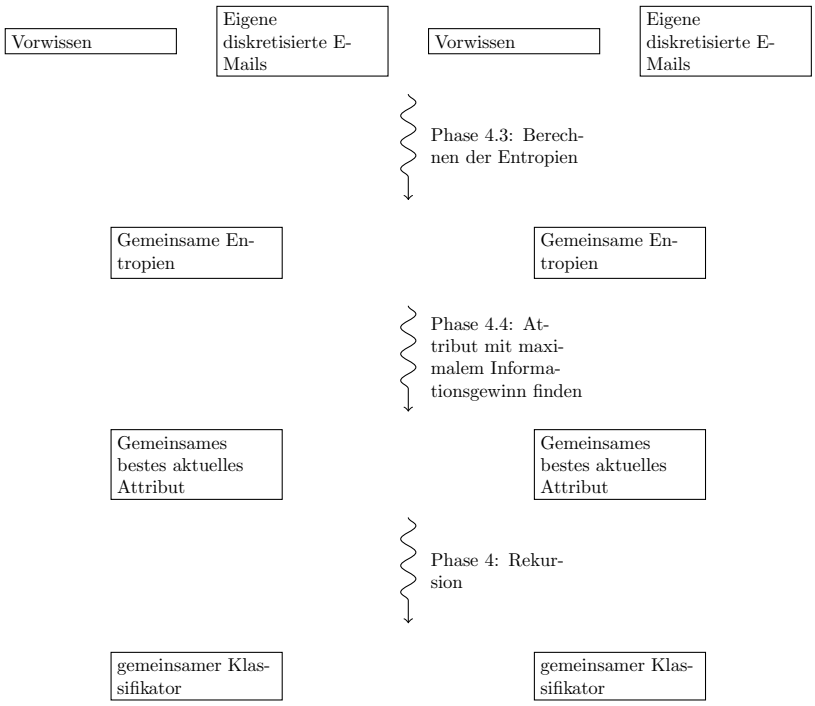

Figure 10: ID3-Algorithmus, Fall 3: Erzeugung eines Astes

analog als $o_B = m_B^1 + m_B^2 + l_B \cdot i_B$. Es gilt dann:

$$\begin{aligned} o_A + o_B = & \quad m_A^1 + m_A^2 + l_A \cdot i_A + m_B^1 + m_B^2 + l_B \cdot i_B \\ = & \quad l_A \cdot i_B + l_B \cdot i_A + l_A \cdot i_A + l_B \cdot i_B \\ = & \quad i_A \cdot (l_A + l_B) + i_B \cdot (l_A + l_B) \\ = & \quad (i_A + i_B) \cdot \ln(i_A + i_B) \end{aligned}$$

Zeile 2 folgt aus Zeile 1 durch ersetzen von $m_A^1 + m_B^1$ bzw $m_A^2 + m_B^2$, die restlichen Zeilen ergeben sich durch wiederholtes Ausklammern. Somit ist die Korrektheit des Protokolles gezeigt. Die Privatheit des Protokolles folgt aus der Privatheit der einzelnen Schritte sowie der Undurchsichtigkeit der berechneten Shares.
Requirement 19: Die Entropie-Shares können anhand diesse Protokolles berechnet werden

6.5 Attribut mit maximalem Informationsgewinn finden

Problemstellung Gegeben das obige Protokoll kann nun die Berechnung des Attributes mit dem maximalen Informationsgewinn einfach durchgeführt werden. Es wird für jedes Attribut das Entropienprotokoll angewendet. Wir erhalten dadurch einen Vektor von Shares $s_A^{a_1}, s_A^{a_2}, \ldots, s_A^{a_k}$ für Alice und einen

Vektor von Shares $S_B^{a_1}, S_B^{a_2}, \ldots, s_B^{a_k}$ für Bob, wobei $a_1, a_2, \ldots, a_k$ die Attribute sind. Wir müssen nun in diesem Vektor den Index mit der minimalen Summe finden, denn wenn die Reihenfolge der Attribute fix und öffentlich ist, kann durch diesen Index bestimmt werden, welches Attribut die geringste Entropie und damit den grössten Informationsgewinn hat.

Unsere Lösung Wir verwenden erneut einen Automaten, der die einzelnen Shares sequentiell abarbeitet und dessen interner Zustand der Index und der Wert des aktuell besten Shares ist. Die Eingabe des Automaten sind immer ein Paar von Shares für das gleiche Attribut sowie der Index dieses Attributes. Wenn S_m, i_m der interne Zustand des Automaten ist, wobei S_m die bisher minimale Summe von Shares ist und i_m der Index dieser Shares und S_A, S_B die aktuellen Eingabeshares sind und i der Index der aktuellen Eingabe ist, dann muss der Automat folgenden Zustandsübergang implementieren:

$$\begin{aligned} S'_m = & \quad \text{if } S_A + S_B < S_m \text{ then } S_A + S_B \text{ else } S_m \\ i'_m = & \quad \text{if } S_A + S_B < S_m \text{ then } i \text{ else } i_m \end{aligned}$$

Wenn wir dies als einen Schaltkreis implementieren können, können wir diesen Zustandsübergangschaltkreis für alle eingegebenen Shares wiederholen. Der Anfangszustand ist das erste Paar Shares.

Bedingte Zuweisung Gegeben die Zuweisung

$$x = \text{ if } b \text{ then } t \text{ else } f \tag{5}$$

und den Fakt, dass ein Schaltkreis B mit einer Ausgabe o_B existiert, sodass $o_B = 1$ ist genau dann wenn b wahr ist, dann kann die Zuweisung implementiert werden durch:

$$x = t \wedge o_B \vee f \wedge \overline{o_B} \tag{6}$$

Im Fall dass b wahr ist, ist $o_B = 1$ und somit gilt $f \wedge \overline{o_B} = 0$ und damit gilt insgesamt $x = t$. Im Fall, dass b falsch ist, ist $o_B = 0$ und somit gilt $t \wedge o_B = 0$ und insgesamt gilt $x = f$. Somit ist dieser Teilschaltkreis korrekt. Desweiteren werden wir die Indizes der Attribute hart in den Schaltkreis kodieren.

Vergleichen Für binärkodierte Integer a, b gilt, dass $a \leq b$ gilt, wenn im ersten unterschiedlichen Bit von a und b in der Stelle von b eine 1 und in der Stelle von a eine 0 steht. Damit kann dies erneut einfach als Automat implementiert werden, der die Zahlen a, b vom signifikantesten Bit zum wenigsten signifikanten Bit liest und die Zustände ,,unentschieden", ,,Ergebnis positiv" und ,,Ergebnis negativ" besitzt. Die Eingabe des Automaten sind die 2 Bits von a, b. Die Zustandsübergangstabelle ist somit:

$a_i, b_i \rightarrow$	1, 1	1, 0	0, 1	0, 0
unentschieden	unentschieden	Ergebnis negativ	Ergebnis positiv	unentschieden
Ergebnis positiv	Ergebnis positiv	Ergebnis positiv	Ergebnis postiv	Ergebnis positiv
Ergebnis negativ	Ergebnis negativ	Ergebnis negativ	Ergebnis negativ	Ergebnis positiv

Wir kodieren dann ,,unentschieden" mit 00, ,,Ergebnis negativ" mit 10 und ,,Ergebnis positiv" mit 11. Damit ergibt sich die binäre Zustandsübergangstabelle:

$a_i, b_i \rightarrow$	11	10	01	00
00	00	10	11	00
11	11	11	11	11
10	10	10	10	10

Wenn nun s_0, s_1 die Zustandsbits des Automaten sind, ergeben sich als Zustandsübergangsgleichungen die Gleichungen:

$$s_0' = s_0 \vee a_i \oplus b_i \tag{7}$$

$$s_1' = s_1 \vee \overline{s_0} \wedge \overline{s_1} \wedge \overline{a_i} \wedge b_i \tag{8}$$

$$\tag{9}$$

Diese Gleichungen machen Sinn, da s_0 kodiert, ob bereits ein Ergebnis gespeichert wird und wenn ein Ergebnis vorhanden ist oder a_i und b_i unterschiedlich sind, bleibt ein Ergebnis gesetzt. Im zweiten Zustandsbit ist kodiert, ob das Ergebnis positiv oder negativ ist. Es wird entweder einfach kopiert oder gesetzt, wenn der Zustand vorher ,,unentschieden" war und a_i 0 ist und b_i 1. Der Anfangszustand ist $s_0 = s_1 = 0$. Diese Zustandsübergangsgleichungen können nun durch ein einfaches Schaltnetz implementiert werden und einfach für alle Attribute wiederholt werden.

Summieren Wir berechnen die Summe wieder durch einen einfachen Ripple-Carry-Adder.

Requirement 20: Dieses Protokoll kann implementiert werden

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 ‘,’ probabilty ‘)’.
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 21: Der Klassifizierer weist die Eingabe Decide((Foo, 0.5,2), Output(Spam), Output(Spam)) wegen eines zu grossen Schwellwertes zurück
Requirement 22: Der Klassifizierer weist die Eingabe Decide((Foo, 2, 3), Output(Spam), Output(Spam)) wegen eines zu grossen Schwellwertes zurück
Requirement 23: Der Klassifizierer weist die Eingabe Decide((Foo, 1, 0), Output(Spam), Output(Spam)) wegen falsch sortierter Schwellwerte zurück
Requirement 24: Der Klassifizierer weist die Eingabe Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam)) wegen zuweniger Teilbäume zurück
Requirement 25: Der Klassifizierer weist die Eingabe Decide((Foo, 0.2, 0.3), Output(Spam), Output(Spam), Output(Spam), Output(Spam)) wegen zuvieler Teilbäume zurück
Requirement 26: 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 27: Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0, 0.5), Output(Spam), Output(Spam))
Requirement 28: Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0.5, 1), Output(Spam), Output(Spam))
Requirement 29: Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0.5, 0.5)), Output(Spam), Output(Spam))
Requirement 30: Der Klassifizierer akzeptiert die Eingabe Decide((Foo, 0, 0), Output(Spam))
Requirement 31: 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 32: 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 33: 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 34: 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 35: 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

8 References

References

[1] Yehuda Lindell Department, Yehuda Lindell, and Benny Pinkas. Privacy preserving data mining. In *Journal of Cryptology*, pages 36–54. Springer-Verlag, 2000.

[2] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[3] Yehuda Lindell and Benny Pinkas. A proof of yaos protocol for secure two-party computation. *Electronic Colloquium on Computational Complexity*, 11:2004, 2004.

[4] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, New York, NY, USA, 1999. ACM.