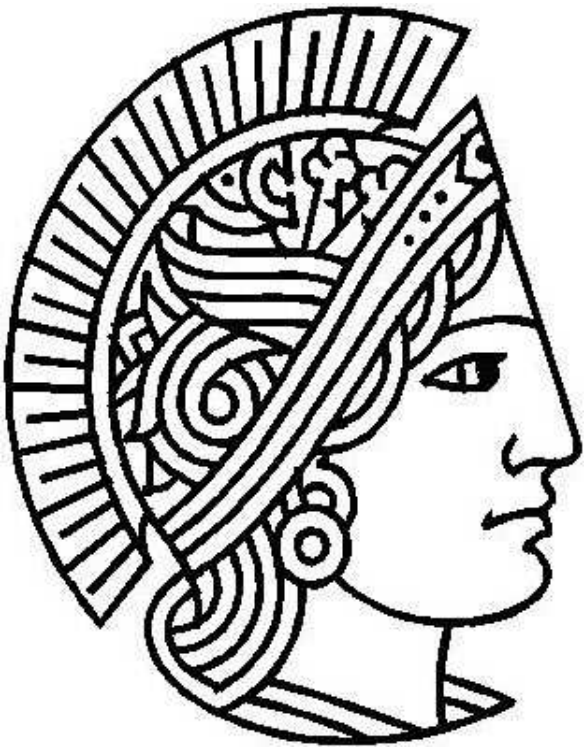
Diplomarbeit



Evaluierung des Mozilla Spamfilters

Kai Brodmann

September 2005

TU Darmstadt
Fachbereich Informatik – Fachgebiet Knowledge Engineering
Prof. Dr. Johannes Fürnkranz

Erklärung
Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weinheim, 19.9.2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Die Spam-Mail-Problematik

Diese Einführung vermittelt einen kurzen Überblick über die gesamte Spam-Mail-Problematik mit allen angrenzenden Themenbereichen.

1.1 Begriffsdefinitionen

Spam ist das Kurzwort für „Spiced Ham“ und heißt übersetzt „gewürzter Schinken“. Dass Spam nun für nervige Werbe-E-Mails steht, hat die Komikergruppe Monty Python zu verantworten. In einem ihrer Sketche bestellt eine Frau im Restaurant mehrere Speisen und bekommt immer „spiced ham“ dazu, obwohl sie das nicht mag. So wurde Spam (s. Abb. 1.1) zum Synonym für ungewollte Übersättigung. [38]

Die Definition von Spam-Mail ist nicht für jede Person oder jede Organisation die gleiche, stark abhängig von deren persönlichem Blickwinkel oder der Interessenlage. So wird die Einschätzung einer Person, die sich in unserer kommerzialisierten Gesellschaft geradezu von Werbung überflutet fühlt, eine andere

Abbildung 1.1: Spam ist das Kurzwort für „Spiced Ham“

UCE unsolicited commercial E-Mail (unverlangte/unerwünschte kommerzielle E-Mail)

SPAM die verbreitetste Bezeichnung für unerwünschte E-Mails

HAM eine verbreitete Bezeichnung für Nicht-Spam-Mails

BULK massenhaft versendete E-Mail

Tabelle 1.1: Kleiner Spam Glossar

sein als die des Vorsitzenden eines Direkt-Marketing-Verbandes. In diesem Zusammenhang fordern Kritiker „Höchststrafen“ für Spammer, ein Anzeichen dafür, dass sich die Internetgemeinde schon erheblich durch Spam-Mails gestört fühlt. Zur Einführung meine Definition von Spam-Mail:

> *Spam-Mail ist unerwünschte E-Mail eines zumeist unbekannten Absenders, deren Versender finanzielle Interessen kommerziellen bis betrügerischen und kriminellen Ausmaßes verfolgt.*

Oft ist diese Definition aus der Sicht vieler Nutzer zu eng gefasst und muss erweitert werden: auf Kettenbriefe, verschickte Witze in Form von Texten, Bildern oder Videos, Newsletters die manchmal sogar von der Person selbst angefordert wurden, sich aber als uninteressant herausstellten. Dabei ist der Weg, den Newsletter mit Hilfe eines Spam-Filters auszublenden, oft kürzer als der einer Abbestellung dieses Newsletters, falls diese überhaupt möglich ist.

Es zeigt sich, dass die Definitionen von Spam-Mail sehr unterschiedlich ausfallen mit „schwammigen“ Begrenzungen, wobei eine große Zahl der Spam-Mails wohl konform von allen als Spam-Mail bezeichnet wird. „Schwammig“ sind eben die Übergangsbereiche; darf beispielsweise eine Firma, bei der ein Buch bestellt wurde, ungefragt wöchentlich über neue Bücher informieren? Meiner Meinung nach nicht, es sei denn, man hätte sich auf der entsprechenden Internetseite mit einem auf „Nein“ voreingestellten Häkchen dafür entschieden.

Das Problem der Spam-Mails ist, wie von vielen angenommen, nicht ganz neu, sondern existiert schon seit den 70er Jahren, als in USENET-Foren bereits automatisch Werbung verbreitet wurde. Damals konnte aber einfach der Absender dieser automatisierten Mails gesperrt werden. Auch gibt es das Spam Problem nicht nur im Bereich E-Mail, sondern auch in Form von Spam-SMS, Spam-FAX, Instant Messenger Spam und, vielleicht nicht üblicherweise so bezeichnet, auch zunehmend ungewollte Spam-Anrufe per Fernsprecher.

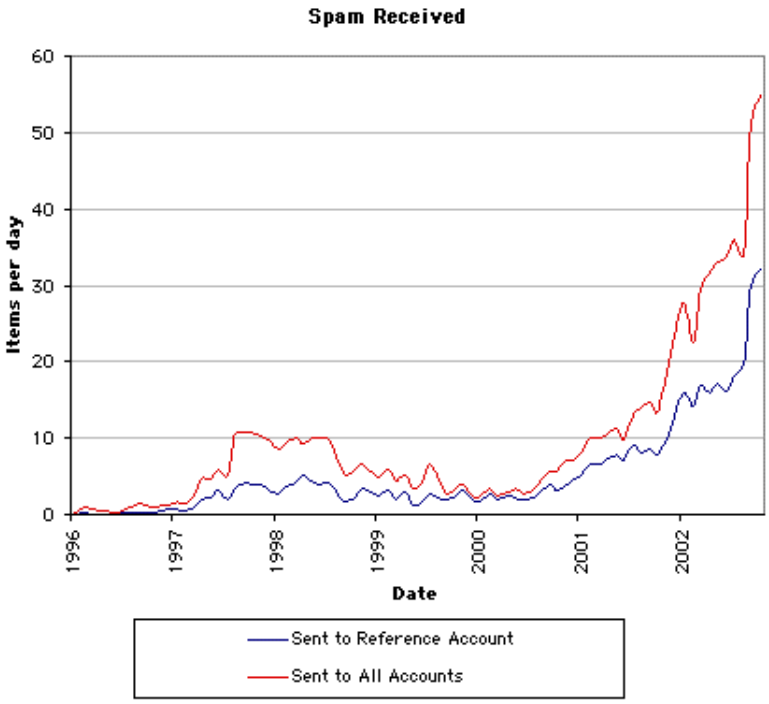


Abbildung 1.2: Spam Entwicklung ab 1996

1.2 Die Problematik

Zum globalen, auch wirtschaftlich sehr bedeutsamen Problem, entwickelte sich die Spam-Mail erst in jüngster Zeit, dramatisch etwa ab der Jahrtausendwende, als im Gegensatz zu früher auch weniger technisch interessierte Menschen (früher nur „Computerfreaks“) Zugang zu E-Mail-Systemen erhielten. Damit entstand ein interessanter Markt für Direkt Marketing Methoden.

Im Jahr 2003 überschritt der Anteil der Spam-Mails in Firmen und Verwaltungen erstmals die 50%-Marke. Dass Spam mittlerweile ein ernstzunehmendes wirtschaftliches Problem ist, belegen die Zahlen der Federal Trade Commission in den Vereinigten Staaten. So wird für Produktivitätsverluste und Ausgaben für Antispammaßnahmen eine Summe von zehn Milliarden Dollar geschätzt. Weitere zwölf Milliarden Dollar entstehen durch Zusatzkosten für das Speichern und Übertragen der Spam-Mails. Der Mailprovider AOL filterte nach eigenen Angaben im Jahr 2003 rund 500 Milliarden E-Mails. In der deutschen Wirtschaft und auch in der öffentlichen Verwaltung werden seit längerem Spamfilter eingesetzt. So benutzt z.B. die Commerzbank den Spamassassin Filter ebenso, wie die Stadtverwaltung von Göttingen. Es entsteht der Eindruck, dass Spam-Mails durch gute Filterlösungen zur Zeit noch in den Griff zu bekommen sind, die Probleme werden aber größer. [70]

Es stellt sich nun die Frage, warum das Konzept der Spam-Mails funktioniert.

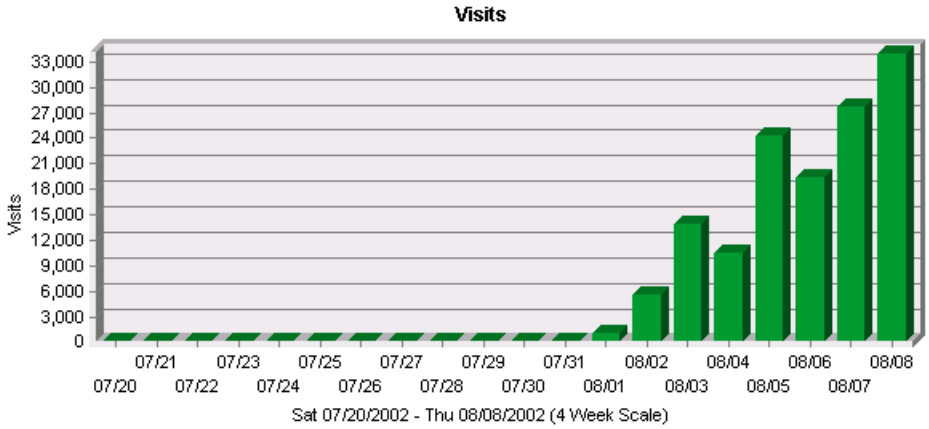


Abbildung 1.3: Zugriffe auf einen Pornoserver seit dem Versenden von Spam-Mails

Dass es funktioniert, zeigt sich daran, dass Spam-Mails in großer Anzahl existieren. Spam-Mail-Versender sind Geschäftsleute, die wie andere auch darauf aus sind, Profit zu erwirtschaften. Abbildung 1.3 [51] zeigt den Anstieg der Zugriffe auf einen Pornoserver seit dem Versenden von zugehöriger Spam-Mail-Werbung.

Das Versenden von Spam-Mails ist praktisch kostenlos, und es reichen minimale Antwort-Quoten dafür, dass sich das Konzept rentiert. Die großen Summen, die der Wirtschaft durch die Zeiteinbußen der Mitarbeiter verloren gehen, die Spam-Mails aussortieren, brauchen die Spam-Versender ja nicht zu bezahlen.

Es gibt Dienstleister, die ihre Versendemöglichkeiten von Spam-Mails Unternehmen anbieten, die hier gegen Bezahlung Werbung platzieren können. All das findet in rechtlichen Grauzonen statt. Um dieses Problem zu lösen, ist es nötig, die Anzahl der gelesenen Spam-Mails und damit die Antwort-Quoten zu senken. Mit Hilfe von Aufklärung, die Antwort-Quoten senken zu wollen, ist wohl unrealistisch. Die Antwort-Quote bedeutet die Anzahl derjenigen, die auf ein Angebot auch tatsächlich reagieren, im Verhältnis zu denen, die die E-Mail empfangen haben.

Verschiedene Geschäftsmodelle basieren auf der Anwendung der unerwünschten E-Mails. Vom kostenpflichtigen Angebot pornographischer Internetinhalte, von Versicherungen über pharmazeutische Produkte, von Angeboten aus dem Finanzwesen, Online Casinos, Reisen und fraglichen Produkten wie gestohlener Software oder gefälschten Zeugnissen, wird fast alles angeboten. Es wurde sogar versucht, mit Hilfe von Spam-Mail, einzelne Aktien Kurse zu manipulieren(siehe Anhang A unter [54]). Natürlich werden auch ganz konkrete Betrugsversuche mit dem Medium Spam-Mail transportiert, wie profitable (!) Investmentvorschläge aus Namibia, Links auf virenverseuchte Internetseiten, für die dann auch gleich der passende „Spezial“ Virenscanner vermarktet wird.

1.3 Strategien zur Bekämpfung von Spam im Überblick

Wie kann man nun diesem Problem entgegentreten? Da wäre zunächst die Unterscheidung zu machen, ob man versucht, bereits das Versenden der E-Mails zu unterbinden oder ob man nach dem Versenden verhindert, dass die E-Mail jemals das Ziel erreicht.

Zur ersteren Möglichkeit zählen rechtliche Bemühungen oder wirtschaftliche Konzepte, die das Versenden nicht mehr lohnenswert erscheinen lassen. Rechtliche Bemühungen könnten Gesetzesentwürfe sein, die das Versenden von Spam-Mail unter Strafe stellen und damit zur Abschreckung dienen - mit Sicherheit ein sinnvoller Teil des gesamten Maßnahmenpaketes, aber trotzdem sehr schwierig, da Regelungen in einzelnen Ländern kaum etwas verändern. Hier sind internationale, rechtliche Vereinbarungen nötig, die schwer durchsetzbar sein werden. Außerdem ist eine internationale Zusammenarbeit in der Strafverfolgung erforderlich.

Es gibt verschiedene Ideen, die das Problem marktwirschaftlich lösen sollen. Wenn E-Mails etwas kosten würden und sei es nur ein kleiner Betrag, würde sich der Break-Even-Punkt, ab dem das Spam-Versenden lohnenswert erscheint, in eine Höhe verschieben, die durch die Antwort-Quoten niemals abgedeckt werden könnte. Es gibt Ansätze, bei denen nur dann gezahlt werden muss, wenn der Empfänger die E-Mail als nicht erwünscht markiert. Somit würden im Normalfall E-Mails praktisch kostenlos bleiben.

Diese Konzepte halte ich für relativ unrealistisch. Falls E-Mails auch nur einen Eurocent kosten sollten, gäbe es meiner Einschätzung nach sofort zahlreiche kostenfreie E-Mail-Systeme. Gerade in unserem „Discount-Zeitalter“ glaube ich nicht, dass jemand bereit ist, für etwas zu bezahlen, das bisher kostenlos war, selbst wenn es sachlich sinnvoll ist. Wahrscheinlich würden sich auch die relativ starken Interessengruppen aus den Bereichen Open Source Software dagegen wehren. Selbst der Missbrauch dieser Modelle durch irgendwelche Lockangebote, auf die man sich per E-Mail melden soll, ist vorprogrammiert. Der Aufwand für die Verrechnung mit „echtem“ Geld wird zu hoch sein. Wesentlich mehr Chancen haben dagegen die „Pricing via Processing“ Ansätze, bei denen man kein Geld für das Zustellen einer unerwünschten E-Mail, sondern Rechenaufwand aufbringen muss(siehe Abschnitt 2.8).

Die zweite oben angesprochene Möglichkeit versucht mit verschiedenen Mitteln, das Zustellen von unerwünschten E-Mails zu vermeiden. Dazu gehören Filter für Endbenutzer, Filter für ganze Unternehmen, die sämtliche eingehenden E-Mails filtern, Vertrauensnetzwerke, Authentifikationsmechanismen/ Spurenrückverfolgung bei E-Mails, Vorverarbeitungsmethoden für die Filter, Black-Whitelists und die Untersuchung der E-Mailströme.

Mit Lernverfahren sind Verfahren gemeint, die ohne Hilfe von Menschen anhand von Beispielen induktiv Klassifikatoren lernen. Im Knowledge-Engineering-

Ansatz werden Regeln von menschlichen Experten erstellt, im maschinellen Lernen benötigt man nur die bereits richtig klassifizierten Beispieldaten. Ganz allgemein werden im Bereich Spam-Mail-Bekämpfung verschiedene Verfahren aus den Bereichen künstliche Intelligenz, Data Mining und stochastische Verfahren eingesetzt, dazu gehören:

- Naive-Bayes Klassifizierer und deren Verwandte [1][11][16][32]
- Neuronale Netze
- Support Vector Maschinen [29]
- Random Forests [29]
- Mustererkennungsverfahren, ähnlich dem Einsatz in der Bioinformatik [23]
- Signaturvergleichsverfahren (im einfachen Fall nur Hash-Funktionen für die E-Mails) [24]
- Genetische Algorithmen
- Spezielle Inferenzverfahren für Vertrauensnetzwerke [7]
- Kryptographische Verfahren zur Authentifizierung
- Bayesian Noise Reduction als Vorstufe für Bayes Klassifizierer [21]
- Lexikographische Abstandfunktionen [22]
- Clusteringmechanismen/Ähnlichkeitskonzepte

Bei allen inhaltsbasierten Filtern lassen sich folgende Schritte verallgemeinern:

1. Feature Engineering bzw. Tokenization: Bevor ein Text von maschinellen Lernverfahren verarbeitet werden kann, muss er zunächst in eine passende Form gebracht werden. Das bedeutet, der Text muss in irgendeiner Art und Weise so in Features umgewandelt werden, dass diese zum trainieren und validieren von Klassifikatoren geeignet sind. Mit Tokenization/Feature Engineering ist gemeint, in welcher Weise ein zusammenhängender Text in Wörter oder Sinnbausteine auseinandergenommen wird. Im Fall eines Bayesschen Klassifizierers wird z.B. von der unrealistischen Annahme ausgegangen, dass verschiedene Wörter keinen Kausalzusammenhang aufweisen. Das ist nicht der Fall, denn wenn in einem Text von „George“ und „Bush“ die Rede ist, dann ist natürlich das Auftauchen der Wörter „verfehlte“ und „Politik“ wahrscheinlicher. Diese Vereinfachung lässt sich schrittweise auflösen, in dem man z.B. immer zwei oder mehrere Wörter als Sinneinheit auffasst, was natürlich die Komplexität erheblich aufbläht. Das ist alles eine Frage des Feature Konzeptes und der verwendeten Lernverfahren.

2. Methoden zur inhaltlichen Analyse: Hier kommen maschinelle Lernverfahren zum Zuge, die aus Beispielen induktiv lernen. Man könnte z.B. aus einer Dokumentensammlung die häufigsten N Worte bestimmen. Damit hat man einen Featurevektor mit N Einträgen. Will man jetzt eine Dokumentenmenge trainieren, wählt man beispielsweise ein neuronales Netz mit N Eingangsneuronen und einem Ausgangsneuron. An die Eingänge legt man dann jeweils für ein Dokument die Anzahl der Vorkommen, eben dieser N Wörter, an den Ausgang legt man Eins oder Null, je nachdem ob es sich um Spam oder Ham handelt. Danach kann man jeden Text anhand der Vorkommen dieser N Wörter versuchen zu klassifizieren. Problematisch ist bei diesem Ansatz aber, dass die Textlänge gar nicht berücksichtigt ist. Somit ist in einem längeren Text die Wahrscheinlichkeit einfach höher, dass viele der Worte enthalten sind. Um nun die Generalisierungsleistung eines Klassifikators zu bestimmen, bedarf es noch der

3. Evaluierung:

   Hier ist es vor allem wichtig, einheitliche Evaluierungskonzepte zu verwenden, die sich wiederholen lassen. Zur grafischen Verdeutlichung von Vergleichen zwischen Spam-Filtern eignen sich beispielsweise ROC-Kurven und Recall-Precision-Diagramme(siehe Kapitel 5).

Weitere Studien beschäftigen sich mit der Vergleichbarkeit verschiedenster Spam-Filter-Ansätze, der Vergleichbarkeit von Spam-Corpora und der Problematik in der Verwendung von Spam-Corpora. Darüberhinaus gibt es Studien, die sich mit dem Beschaffen und Weiterverwenden von E-Mail-Adressen, also potentiellen Spam-Adressen beschäftigen. Diese Studien gehen folgenden Fragen nach:

- Wie bekommen die Spam-Versender ihre E-Mail-Adressen?
- Wie schnell werden E-Mail-Adressen auf Internetseiten von sogenannten „Erntemaschinen“ der Spammer eingesammelt?
- Wie werden diese E-Mail-Adressen weiterverwendet?
- Wie geographisch weiträumig findet der Handel mit potentiellen Spam-Adressen statt?
- Welche Möglichkeiten gibt es für Spammer, die Filter zu umgehen, wie effizient sind diese?
- Wie kann man die Filter der Zukunft resistent gegen Angriffe machen?

Das Projekt aus Abschnitt 2.10 versucht, Antworten auf diese Fragen zu geben.

1.4 Die Tricks der Spammer zur Verwirrung der Filter

Seit es Spam-Mails gibt, existiert auch eine Spam-Evolution, die daraus entsteht, dass Spammer versuchen, die aktuellen Maßnahmen gegen Spam-Mails zu umgehen. Im folgenden werden die bekanntesten Tricks in sprachlicher, verständlicher Kurzform zusammengefasst. Die ausführliche Dokumentation mit Beispielen in HTML findet sich in [67].

So ist es z.B. möglich, dass die gesamte E-Mail nur aus einem Hyperlink auf ein Bild besteht, in der die Werbebotschaft enthalten ist. Bilder zu dekodieren ist bei weitem schwieriger als Texte und fällt in den Bereich des maschinellen Sehens. Das maschinelle Sehen ist dabei unter Umständen mit unvertretbarem Rechenaufwand verbunden, vor allem für Server-seitige Spam-Filter, die sehr viele E-Mails in kurzer Zeit verarbeiten sollen.

Ein weiteres Problem: Wenn das Bild im Internet auf dubiosen Servern liegt, müsste zunächst ein Web-Crawler diese Seite aufsuchen, das Bild herunterladen, mittels maschinellem Sehen verarbeiten und auf Werbebotschaften überprüfen. Das Problem dabei wird sein, dass auch hier Tricks verwendet werden könnten wie das Einfügen von anderen Hyperlinks, eventuell mit „unsichtbarer Tinte“ geschrieben, also mit identischer Schrift- und Hintergrundfarbe. Falls es den Spammern z.B. möglich wäre, Links von offiziellen Seiten einzufügen und der automatische Web-Crawler des Spam-Filters alle diese Seiten aufsucht, wären diese Seiten wahrscheinlich restlos überlastet. Das Ganze käme einer Denial-of-Service-Attacke in Form von Überlastung gleich.

Beliebt ist auch der Täuschungsversuch mit unsichtbarer Tinte, z.B. mit weisser Schriftfarbe auf weissem Hintergrund. Dabei sollen dann für den Menschen unsichtbare zusätzliche Wörter eingefügt werden, die z.B. Bayessche Klassifizierer verwirren. So könnten extrem wenig spammige Wörter zur Verschleierung einer Werbebotschaft eingefügt werden, wie zufällig ausgewählte Nachrichtentexte, oder Wörter aus einem Lexikon. Aus diesem Grunde verwenden manche Spam-Filter eine Art optischen Filter, der die Spam-Mail zunächst so erscheinen lässt, wie sie auch vom menschlichen Auge wahrgenommen wird. Außerdem werden Wörter gerne durch bedeutungslose HTML-Tags aufgebrochen, um die Erkennung zu erschweren. Wörter können in Tabellen angeordnet sein, die man von oben nach unten lesen soll, oder in jeder Form von HTML, die vom Menschen lesbar ist, aber bei der die spammigen Wörter nicht direkt als ganzheitliche Zeichenfolgen im HTML-Quelltext vorkommen.

1.5 Überblick über die Arbeit

Ziel der Diplomarbeit ist es, eine Übersicht über das Phänomen Spam-Mail zu geben, die Grundbegriffe zu erklären, um daraufhin den aktuellen Ansatz von

Paul Graham, „A Plan for Spam“, zu evaluieren und mit einem wissenschaftlich fundierten Algorithmus, einem Naive-Bayes-Textklassifikator, zu vergleichen.

In Kapitel 1 werden die Grundbegriffe der Spam-Problematik eingeführt und eine Übersicht vermittelt. Kapitel 2 beschreibt die wichtigsten bereits vorhandenen Ansätze zur Spam-Mail-Bekämpfung. Kapitel 3 geht explizit auf Bayessche Spam-Filter und deren Verwandte ein, um in diesem Zusammenhang Grahams „Plan for Spam“ mit einem Naive-Bayes-Textklassifikator zu vergleichen und deren Leistungsfähigkeit bezüglich der Spam-Problematik zu untersuchen. Dabei werden die Gemeinsamkeiten und Unterschiede zwischen den beiden Algorithmen beschrieben und diskutiert. Kapitel 4 befasst sich mit dem im Mozilla-Paket enthaltenen Mail-Programm und mit Open Source Software. Dabei wird vor allem auf praktische Aspekte, die bei der Implementierung der Algorithmen von Bedeutung waren, Gewicht gelegt. Wenn implementierungsspezifische Details nicht von Interesse sind, kann dieses Kapitel übersprungen werden. Im Mozilla-Mail-Programm wurden beide Algorithmen aus Kapitel 3 in einer Implementierung getestet. Kapitel 5 zeigt die Testmethodologie der Testreihen aus Kapitel 6 zusammen mit der Datenbeschaffung und der verwendeten Evaluierungstechnik. Kapitel 6 geht konkret auf die durchgeführten Testreihen ein, und die Ergebnisse werden vorgestellt. Kapitel 7 enthält eine Zusammenfassung der gesamten Arbeit und einen Ausblick mit Anregungen für weitere Forschung.

Kapitel 2

Vorhandene Ansätze des Bekämpfens von Spam-Mail

In diesem Kapitel wird auf die wichtigsten vorhandenen Maßnahmen zur Bekämpfung von Spam-Mail eingegangen. Die sehr häufig in der Realität implementierten Bayesschen Klassifizierer werden in Kapitel 3 behandelt. Nach den wichtigsten Ansätzen werden noch die juristische Lage, ein spezielles Projekt zum Erforschen der Spam-Mail-Versender und ein zusammenfassendes Modell behandelt, das versucht, alle aktuellen Spam-Filter-Ansätze zu abstrahieren. Dieses Modell findet dann auch beim Vergleich der beiden Algorithmen in Kapitel 3 Verwendung.

2.1 Vorverarbeitung: Bayesian Noise Reduction

Eine weitere Möglichkeit zur Verbesserung der Spam Klassifikation sind Algorithmen, die das „Rauschen“ in den Texten reduzieren sollen. Mit „Rauschen“ sind bei der Textklassifikation die Worte gemeint, die nicht wirklich in ihre Umgebung passen. Wenn sich also wenig spammige Wörter mitten in einer strahlenden Werbebotschaft befinden, lassen sich solche Tricks unter Umständen umgehen, in dem das wenig spammige Wort ausgeblendet wird. Somit handelt es sich bei Bayesian Noise Reduction um eine Vorverarbeitung der Texte, bevor sie den Klassifizierern (oder auch dem Trainingsalgorithmus) präsentiert werden.

Hierbei können Parallelen zu Gewöhnungskonzepten aus der Biologie und der künstlichen Intelligenz gezogen werden. Als Beispiel sei hier erwähnt, dass das Auge nach zwanzig bis dreißig Sekunden gleichbleibende Objekte nicht mehr wahrnimmt, um nur die wesentlichen Änderungen des Bildes zu bemerken. Ähnlich verhält es sich auch beim Hören. Wenn Musik läuft und man anderseits Stimmen von Passanten verstehen möchte, kann man die Musik ansatzweise durch das Gehirn ausblenden lassen [21].

2.2 Vorverarbeitung: Verwendung lexikographischer Abstände

Spammer versuchen Spamfilter zu umgehen, indem sie Wörter vorsätzlich so verändern, dass der Leser sie zwar noch lesen kann, sie den Spam-Filter aber verwirren. Dazu zählt beabsichtigt falsche Rechtschreibung wie „Vlagra“ anstatt „Viagra“ oder das Verwenden von buchstabenähnlichen Symbolen wie „Vi@gra“. Mit Hilfe dieser Mutationen ergeben sich 600,426,974,379,824,381,952 Möglichkeiten „Viagra“ als Begriff darzustellen. Zweitens ist es möglich, zufällige wenig spammige, Wortkombinationen zur E-Mail hinzuzufügen (siehe Abschnitt 2.1). Mit Hilfe von Algorithmen, die Lexikographische Abstände zwischen den verschiedenen Schreibweisen ausrechnen, ist es möglich, Mutationen auf ihren Stamm zurückzuführen und damit die Effizienz der nachgeschalteten Bayesschen Klassifizierer zu erhöhen [22].

2.3 Chung-Kwei: musterorientierte Ansätze aus der Biotechnologie

Mehr kombinatorisch und weniger stochastisch mutet dieser Ansatz an, der auf einer großen Testmenge von 87.000 Mails trainiert und auf einer etwa gleich großen Menge von 88.000 Mails validiert wurde. Dabei wurde eine Spamerkennung von 96.56% und eine *False Positiv Rate* von nur 0.066%, also eine Mail in sechstausend, erreicht. Der Ansatz beruht auf einem in der Biotechnologie erfolgreich und oft angewendeten Algorithmus namens Teiresias. Das Prinzip besteht darin, dass man eine Trainingsmenge von Spam-Mails als einen einzigen Text betrachtet, also einfach die Zeilenumbrüche entfernt und aus diesem Text mittels des Algorithmus Teiresias eine Art Spam-Vokabular gewinnt.

Das heißt, der Algorithmus sucht jede Sequenz beliebiger Länge heraus, die mehr als einmal in der Menge der Spam-Mails vorkommt. Optional ist es nun möglich, genauso mit den vorhandenen Ham-Mails zu verfahren und dann vom Spam-Vokabular das Ham-Vokabular abzuziehen.

Das verkleinert die Rate der eventuell vorhandenen *false positives*, kann aber zu einer geringeren Spam-Erkennungsrate führen. Bei einer Anfrage für eine zu klassifizierende E-Mail wird nun der Schnitt des Spam-Vokabulars mit dieser E-Mail gebildet. Dabei erhält man alle Muster aus dem Spam-Vokabular, die in der E-Mail vorkommen, natürlich inklusive der vorhandenen Untermuster. Nun wird einfach gezählt, wie oft die einzelnen Spam Vokabeln in einer E-Mail vorkommen und auf welchen Positionen sie sich befinden. Daraus wird später mittels zweier Schwellenwerte ermittelt, ob es sich bei der Anfrage um Spam handelt oder nicht. Ein Schwellenwert gibt an, wie viele Muster aus dem Spam-Vokabular in der Mail vorkommen müssen; der andere gibt an, wieviel Prozent der Länge der Mail

mit Spam Vokabeln abgedeckt sein muss. Man kann natürlich für den Kopf und den Körper der E-Mail zwei verschiedene Vokabelverzeichnisse anlegen. In den Experimenten dieses Berichtes wurden nur die Körper der E-Mails betrachtet, mit den oben genannten erstaunlich guten Ergebnissen.

In den ersten Versuchen wurde mit 65.175 Spam-Mails trainiert. Dann wurde das oben genannte Negativ Training mit 21.355 Ham E-Mails durchgeführt. Dabei wurde die eine Hälfte der Ham-Mails zum Negativ-Training verwendet, die andere Hälfte dafür, die beiden oben genannten Schwellenwerte einzurichten. Sie wurden so eingerichtet, dass in der benutzten Hälfte der Ham-Mails eine falsch klassifizierte Ham-Mail unter 10000 auftrat. Die Schwellenwerte wurden auf 26 und 19% eingestellt. Das bedeutet, dass mindestens 26 der Muster aus dem Spam-Vokabular überhaupt vorkommen müssen und die E-Mail zu mindestens 19% ihrer Länge durch diese Vokabeln abgedeckt sein muss. Das durch Negativ-Training verkleinerte Spam Vokabular bestand aus 6.660.116 Mustern. Das Training auf den 86.530 Mails dauerte auf einem 2.2 GHZ Intel Pentium PC 17 Minuten, die Klassifizierungsgeschwindigkeit lag bei 214 Nachrichten pro Sekunde[23].

2.4 Hashing und Erkennung von Duplikaten

Eine weitere Methode zur Bekämpfung der Spam Flut sind Hashing-Methoden, die dazu dienen, Duplikate im E-Mail Strom zu erkennen. Auf Grund des immensen Anwachsens der Datenmengen und des Internets wurde in den 90er Jahren zunehmend die algorithmische Greifbarkeit von Duplikaten von Textdokumenten wichtig, z.B. im Einsatz bei Suchmaschinen. Wenn ein Benutzer nach einem bestimmten verbreiteten Dokument sucht, soll er natürlich nicht alle Links auf Kopien dieses Dokumentes in jedweder Form erhalten, weil dadurch der Sinn von Suchmaschinen verloren ginge. Die Übersicht wäre verloren. Statt dessen möchte man dem Benutzer die Ursprünge oder die wichtigen oder ersten Auftreten dieser Dokumente anzeigen. Dabei ist es nützlich, Duplikate auszublenden.

Zur weiteren aktuellen Anwendung dieses Prinzipes gehört es, gestohlene Designs oder Konzepte im Internet zu finden und die Diebe ausfindig zu machen. Duplikaterkennung kann auch in der Spam-Mail-Bekämpfung sinnvoll eingesetzt werden, weil ja Spam-Mails immer in sehr großen Volumen versendet werden. So wäre es wünschenswert, bereits am äußersten Eingang eines E-Mail Systems in einem Unternehmen sämtliche Duplikate von einer als sicher geltenden Spam-Mail zu eliminieren. Hierfür finden Ähnlichkeits-, Signatur- oder Hashing Konzepte Anwendung. Die Definition von Duplikaten ist wie die Definition vom Spam-Mails sehr schwammig. Ist ein Duplikat wirklich nur eine exakte binäre Eins-Zu-Eins Kopie des Originals oder sind auch Vertauschungen der Wörter oder kleine Veränderungen im Text zulässig?

Im Bereich der Spam-Mails ist natürlich erwünscht, diesen Schwellenwert der Definition „Duplikat“ so zu setzen, dass alle Mails mit zur Verwirrung der Filter

bewusst eingefügten Änderungen der Spammer noch als Duplikate gelten. Dagegen darf es wie üblich nicht passieren, dass eine Ham-Mail so sehr einer schon bekannten Spam-Mail ähnelt, dass sie als *false positive* aussortiert wird.

Es kommen verschiedenste Ähnlichkeitskonzepte zum Einsatz. Eine häufig erwähnte Methode des Fuzzy Hashings scheint I-Match zu sein. Hierbei wird zunächst ein Vokabular aus allen vorhandenen Trainingsdokumenten gebildet, wobei für jeden Eintrag dieses Vokabulars die „inverse document frequency“ (idf) berechnet wird. Dieses Lexikon dient dann dazu, den Hashwert aus der Schnittmenge der Tokens eines zu klassifizierenden Dokumentes mit der Menge der Tokens im Lexikon zu berechnen.

Die inverse document frequency (idf) ist definiert als

$$t_x = \log(N/n) \tag{2.1}$$

wobei N die Anzahl aller Dokumente und n die Anzahl der Dokumente ist, die den gesuchten Term enthalten. Somit klassifiziert I-Match ein Dokument, in dem es das Dokument in Tokens aufspaltet, die Hash-Funktion berechnet, dann aus der Schnittmenge dieser Tokens und dem verwendeten Lexikon einen Hash-Wert ermittelt.

Falls für zwei Dokumente eine Kollision im Hashtable entsteht, gelten diese Dokumente als Duplikate. Leider ist das normale I-Match Verfahren sehr anfällig für kleine Änderungen in den zu klassifizierenden Dokumenten. Das Verfahren ist invariant gegen Vertauschung von Wortreihenfolgen, aber keineswegs gegen das Entfernen oder Hinzufügen von Worten. Es wurden Ansätze unternommen, diese Empfindlichkeit von I-Match gegen von Spammern gezielt eingesetzte Mutationen in ihren Spam-Mails einzudämmen[5]. Dabei wurde versucht, die Stabilität der I-Match-Signaturen dadurch zu erhöhen, dass aus der normalen I-Match-Signatur ein n-dimensionaler Signaturen-Vektor wird, wobei sich jede Koordinate im Vektor auf ein anderes Lexikon bezieht. Die Lexika entstanden aus randomisierten Versionen des Ursprungslexikons. Somit wirkt sich eine Änderung in einer zu klassifizierenden Nachricht eher nur auf eine Koordinate des Signaturen-Vektors aus. Wenn man z.B. ein Duplikat so definiert, dass sich der Signaturen-Vektor mindestens in einer Stelle gleicht, wird I-Match mit Signaturenvektor invarianter gegenüber kleinen Veränderungen an den E-Mails[24].

2.5 Personalised, Collaborative Spam Filtering (P2P Technologie)

Spam-Filter, die auf dem Training und der anschließenden Klassifizierung von Mails beruhen, können nur vom Endbenutzer oder von Gruppen trainiert und benutzt werden. Persönlich trainierte Spam-Filter haben den Vorteil, dass sie die eigene Vorstellung von dem, was Spam-Mails sind und was nicht, gut repräsentieren. Wenn z.B. eine Firma einen Spam-Filter benutzt, der für alle Benutzer

klassifiziert, tritt oft das Problem auf, dass die Definition von Spam für jeden Benutzer unterschiedlich ist und somit *false positives* für einzelne Benutzer entstehen. Der Vorteil ist, dass nicht jeder Benutzer sich um das Training kümmern muss, was ohnehin unter dem Gesichtspunkt der Benutzbarkeit oder Bereitschaft zum Benutzen ziemlich unrealistisch erscheint.

In dem Artikel „Personalised, Collaborative Spam Filtering“ von Alan Gray und Mads Haahr [6] wird eine Lösung vorgeschlagen, die beide Welten mittels eines P2P (Peer-to-Peer) Netzwerkes kombiniert. Die Idee ist, dass jeder Benutzer einen Spam-Filter benutzt, der gleichzeitig ein Peer eines P2P Netzwerkes ist. Das Ziel ist, das System sehr allgemein zu formulieren und nicht auf einen bestimmten Algorithmus festzulegen. Wichtig ist nur, bestimmte Kommunikationsstandards auf XML Basis innerhalb des P2P Netzwerkes einzuhalten. Dabei tauschen die Peers per SMTP Nachrichten aus, die zur Aufrechterhaltung und Verbesserung des P2P Systems dienen. Somit möchte man erreichen, dass sich Peers und damit Benutzer zusammenschließen, die eine ähnliche Vorstellung von der Definition von Spam-Mails haben. Weiterhin sollen die Konzepte „räumliche Entfernung und Vertrauen zum jeweiligen Peer“ mit intergriert werden.

Man möchte eben genau mit den Peers kommunizieren, die gute Klassifizierer besitzen, die der eigenen Vorstellung von Spam-Mails am besten entsprechen. Gleichzeitig achtet man natürlich darauf, von wem man Klassifizierer bekam, die möglichst selten oder noch nie einen false positive verursacht haben. Somit ist es möglich, auch wenn man selbst gar nicht viele E-Mails bekommt, mit denen man trainieren kann, sich mit Peers zusammenzuschließen, die die eigene Definition von Spam-Mails erfüllen und für den anderen Benutzer mittrainieren. Man tauscht sich sozusagen aus.

Peers, die besonders gut klassifizieren können, werden sehr bekannt, weil man natürlich auch den guten Ruf der anderen Peers seinen Nachbarn mitteilt. Besonders fleißige Peers rutschen deshalb weiter ins Zentrum des Netzwerkes, während faule oder sinnlose Peers automatisch an den Rand gedrängt werden, so z.B. Benutzer, die nur profitieren, aber selbst keine guten Klassifikatoren beisteuern. Vorteilhaft erscheint auch, dass P2P Netzwerke schwer zu attackieren sind und im Falle von Spam-Filter-P2P-Netzwerken wird das natürlich von Spammern versucht werden. Durch die P2P Struktur ist eine geringe Angreifbarkeit und hohe Zuverlässigkeit garantiert, außerdem können Ressourcen der einzelnen Rechner wie Speicherplatz oder CPU Zyklen sinnvoll im Überlappungsnetzwerk verteilt werden. In dem Artikel wurde eine Implementierung der Ideen vorgestellt, die auch im kleinen Rahmen mit wenigen Benutzern getestet wurde. Die Ergebnisse sind erfolgversprechend, allerdings hauptsächlich unter dem Gesichtspunkt, dass das System sein wirkliches Können erst dann zeigt, wenn im realen Betrieb große P2P Netzwerke entstehen. Die Autoren versprechen aber, weitere Tests in grösserem Umfang durchzuführen[6].

2.6 Vertrauensnetzwerke zur Spam Bekämpfung

Whitelists und Blacklists repräsentieren die „Guten“ und die „Bösen“ in Form von E-Mail-Absendern. Diese zwei Ansätze werden oft als Ergänzung verwendet, um schon einmal eine gewisse Vorauswahl fast sicher als Spam oder Nicht-Spam zu klassifizieren. Man kann z.B. eine Whitelist pflegen, indem man alle Bestimmungsadressen der Mails, die vom Benutzer abgesendet werden, in die Whitelist einpflegt. Denn man kann normalerweise davon ausgehen, dass man von dem, dem man E-Mails sendet, auch bereit ist, E-Mails zu empfangen.

Blacklists gibt es in verschiedenen Formen. Da die Mail-Köpfe aber sehr oft gefälscht werden, haben Blacklists mit einer Reihe von Problemen zu kämpfen. In Blacklists sollen normalerweise die Absender stehen, von denen man auf keinen Fall Mails empfangen möchte. Anstatt dieser binären Approximation des Vertrauens, das man dem Absender entgegenbringt, versuchen Vertrauensnetzwerke dem Absender wirklich kontinuierliche Vertrauenswerte zuzuordnen. Das bedeutet, jeder Benutzer bewertet seine eingehenden E-Mails bzw. die Absender mit einer Zahl, die das Vetrauen gegenüber diesem Absender ausdrückt, z.B. eine Zahl von 1 bis 10. Somit bewertet jeder Benutzer seine Nachbarn (oder den Kanten im Graph) mit einem gewissen Vertrauen, das nun über Inferenz im sozialen Netzwerk für alle nutzbar gemacht werden kann. Es gilt das Prinzip: der beste Freund meines Freundes ist wahrscheinlich auch für mich vertrauenswürdig. Hierbei kann man sich einer lokalen oder globalen Vertrauensmetrik bedienen. Bei einer globalen Vertrauensmetrik bekommt jeder Benutzer im Netz genau einen Vertrauenswert. Bei einer lokalen Vertrauensmetrik können verschiedene Teilnehmer einen anderen, je nach Sicht der Dinge, sehr unterschiedlich bewerten, was im Zusammenhang mit Spam-Vermeidung angebrachter zu sein scheint.

Beispiele für eine Metrik und eine reale Implementierung in Form von einem E-Mail Programm namens „TrustMail“ finden sich in [7]. Der praktische Sinn dieser Methode ist dabei, dass alle E-Mails, die in der Inbox ankommen, automatisch nach Vetrauenswerten geordnet werden. Das heißt, die Mails der wichtigsten oder vertrautesten Personen stehen ganz oben und können zügig bearbeitet werden. Nicht durch andere Spamfilter-Technologien aufgehaltene Nachrichten sollten kein Vetrauen bekommen und ganz unten zu lesen sein.

2.7 Turing Test im Einsatz gegen Spam-Mail

Manche Forschungsrichtungen setzen darauf, dass der Sender einer E-Mail sich in irgendeiner Art und Weise als Mensch authentifizieren soll, so dass massenhaftes Versenden von E-Mails nicht mehr möglich ist. Ähnlich der rein wirtschaftlichen Ansätze, bei denen man einen kleinen Betrag pro E-Mail zahlt und damit das Spammen unwirtschaftlich wird, kann man diesen Preis einer E-Mail auch dadurch simulieren, dass ein Mensch ein paar Sekunden seiner Zeit darauf verwen-

det, einen kleinen Turing-Test zu lösen. Unter Turing-Test versteht man einen Test, der es ermöglicht, bei einem Experiment herauszufinden, ob die andere Seite, mit der man kommuniziert, ein Mensch ist oder ein Computer. So könnte man z.B. per Textchat kommunizieren, und der Computer könnte versuchen, sich als Mensch auszugeben. Diese Fähigkeit besitzen Computer allerdings noch nicht, auch wenn manche auf einfacher Logik basierende Programme einen Menschen schon kurzzeitig glauben lassen können, es säße ein Mensch auf der anderen Seite. In der Thematik der Spam-Bekämpfung könnte man verlangen, dass jemand, der einem anderen eine Mail schreiben will, einen kurzen Test macht (im Bereich von Sekunden) und somit beweist, dass es sich bei ihm um eine Person und nicht um Softwareroboter handelt. Ähnlich der Kryptographie gäbe es eine Funktion, die eine bestimmte Gruppe ausrechnen kann (die Menschen) und eine bestimmte Gruppe nicht (die Computer). Wichtig ist dabei, das es technisch kein Problem sein sollte, diese Aufgaben zu stellen und die eindeutigen Lösungen dazu zu kennen.

Beispiele für solche Tests:

- Feststellung des Geschlechts eines Menschen anhand von Gesichter-Fotos
- Erkennung des Gefühlsausdruckes von menschlichen Gesichtern anhand von Fotos von Gesichtern
- Anklicken von bestimmten Körperteilen eines Menschen oder eines Tieres, z.B. das Auge
- Erkennung, ob Personen bekleidet sind oder nicht [8]
- Erkennung von einfachen Skizzen (Haus, Auto, Schiff, Person)
- Erkennung handgeschriebener Zeichen
- Spracherkennung
- Lückentext ausfüllen, hierbei muss also in einem Satz beispielsweise ein Verb aus einer Liste eingefügt werden, Kenntnisse über die Semantik sind somit vorrausgesetzt

Nach meiner Einschätzung ist das Durchführen solcher Tests, falls die Spam-Problematik nicht anders in den Griff zu bekommen ist, auf jeden Fall realistisch, jedenfalls deutlich realistischer als wirtschaftliche Modelle, bei denen ein Preis für eine E-Mail bezahlt wird. Dieses Konzept hätte dann gute Chancen, wenn nicht jede E-Mail auf diese Art und Weise authorisiert werden müsste. Das lässt sich erreichen durch die Kombination mit anderen Verfahren, die nur dann eine Authorisierung erfordern, wenn die E-Mail wirklich aus einer fraglichen Quelle stammt. Somit ist eine Kombination mit Whitelists oder Vertrauensnetzwerken

sinnvoll. Man könnte den Turing-Test auch nur dann durchführen lassen, wenn ein beliebiger vorgeschalteter Spam-Filter die E-Mail ansonsten nicht zulässt. Somit wäre dem Versender auf jeden Fall eine Auslierung seiner E-Mail garantiert. Trotz alledem ist der Ansatz drastisch, denn man könnte eine E-Mail-Antwort, in der man seine versendete E-Mail authentifizieren soll, auch im Sinne von „Meine Zeit ist wichtiger als Ihre“ empfinden. Somit muss die Gegenseite Sonderaufwand betreiben, wenn sie jemanden wirklich erreichen will. Gerade in der Anfangszeit, wenn diese Methode noch nicht überall verbreitet ist, könnten Missverständnisse auftreten[18].

Bei weiterem Interesse an Turing-Tests sei auf den Loebner Preis verwiesen, eine Instanz des vom britischen Mathematikers Alan Turing 1950 vorgeschlagenen gleichnamigen Tests, die jährlich durchgeführt wird. Dieser von Hugh Loebner 1988 ins Leben gerufene Wettbewerb soll die Weiterentwicklung der künstlichen Intelligenz in Richtung Niveau des menschlichen Intellektes fördern und die Thematik bekannter machen[36].

2.8 Pricing via Processing, Rechenleistung als Zahlungsmittel

Wie schon in der Einführung erwähnt halte ich Modelle, die E-Mails kostenpflichtig machen für nicht realistisch. Völlig anders sieht es dagegen aus, wenn diese Kosten nur in Form von zu erbringender Rechenleistung vorhanden sind. Damit erzielt man einen ähnlichen Effekt wie bei Bezahl-Ansätzen, es weist aber nicht die vielen zum grossen Teil psychologisch bedingten Nachteile auf. Die Grundidee ist, dass man eine E-Mail nur dann ins eigene Postfach einsortiert, wenn der E-Mail ein Schlüssel beigefügt wurde, der zur E-Mail passt. Dieser Schlüssel muss vom Versender berechnet werden und sollte mit einem mittelgrossen Rechenaufwand verbunden sein[9].

Das bedeutet z.B., dass der Rechner des Versenders 10 Sekunden rechnen muss, um den Schlüssel zu bestimmen. Das ist für die normale Verwendungsweise von E-Mails kein Hindernis, insbesondere, wenn der Prozess der den Schlüssel ausrechnet, als Prozess mit niedriger Priorität läuft, so dass der Benutzer eigentlich nichts davon merkt. Für Spam Versender dagegen wird dies zum unlösbaren Problem, da möglichst viele E-Mails pro Zeiteinheit abgesetzt werden sollen. Wenn aber jede E-Mail 10 Sekunden Rechenzeit kostet, ist das Spammen nicht mehr gewinnbringend möglich.

Aus diesem Grunde ist es wichtig, dass die Algorithmen die zur Berechnung des Schlüssels erforderlich sind so geartet sind, dass beim Berechnen von vielen Schlüsseln kein Rechenvorteil gegenüber dem Berechnen einzelner Schlüssel möglich ist. Der Rechenaufwand sollte sich linear gegenüber der Anzahl der zu berechnenden Schlüssel verhalten. In einigen Modellen ist es vorgesehen, die Uhr-

zeit oder das Datum mit in den Schlüssel einzubeziehen, so dass nicht Schlüssel mehrfach verwendet werden können. Würde man z.B. den Schlüssel nur aus der E-Mail-Adresse des Empfängers berechnen, gäbe es wahrscheinlich bald regen Handel mit E-Mail-Adressen plus der dazugehörigen Schlüssel. Durch Hinzunahme von Uhrzeit und Datum wird dies unmöglich. Wichtig ist auch bei diesem Konzept, dass das Verifizieren eines Schlüssels leichter ist als das Berechnen des Schlüssels, damit Authoritäten im Netz oder der Empfänger sehr schnelle viele E-Mails überprüfen können.

Weiterhin wäre es unter Umständen sinnvoll, einen Art Code an die Authoritäten im Netz zu verteilen, der eine schnelle Validierung der Schlüssel ermöglicht, aber nur für die Stellen, die eben diesen Code besitzen. Durch die rasante Weiterentwicklung der Rechenleistung von Computern ist es auch wichtig, dass der Rechenaufwand zur Berechnung der Schlüssel in einem akzeptablen Bereich bleibt. Ein Schlüssel der jetzt sehr viel Rechenzeit braucht, ist in zehn Jahren vielleicht selbst für Spammer schon wieder in vertretbarer Zeit zu berechnen, falls es in zehn Jahren noch Spammer gibt, was wir alle nicht hoffen wollen. Zu diesem Zwecke ist es auch sinnvoll, dass die Algorithmen ein „Schräubchen“ besitzen, an dem man die Komplexität der zu berechnenden Schlüssel beliebig verändern kann. Andere Ansätze konzentrieren sich anstatt auf Rechenleistung mehr auf Speicherbedarf, da der Speicher nicht so schnell anwachsen wird wie die Rechenleistung[10].

2.9 Juristische Bekämpfung der Spammer

Inzwischen wird weltweit über Spam-Bekämpfung diskutiert und angesichts der anwachsenden Computer-Kriminalität nach härteren Urteilen der Justiz verlangt. Die Rechtslage bleibt auch heute noch unklar. Anstatt Spammer mit harten Strafen zu belegen wird darüber diskutiert, ob es rechtlich überhaupt gestattet ist, Spam-Mail herauszufiltern. [69] Vereinzelt hört man von Verurteilungen. In den Vereinigten Staaten von Amerika wurde ein Spammer zu 9 Jahren Haft verurteilt. In Deutschland scheint das Recht sogar nach wie vor auf der Seite der Spammer zu sein. So wurde am 17. Februar 2005 im Deutschen Bundestag ein Gesetz verabschiedet, welches das Verschleiern einer Werbe-E-Mail durch irreleitende Betreffzeilen verbietet, aber nicht das Versenden selbst! Grundsätzlich ist es möglich, Unterlassungsansprüche gegen die Spammer aus dem Wettbewerbsrecht und dem Haftungsrecht herzuleiten. Zunehmend findet auch das Strafgesetz Eingang in die Diskussion. Aber es entsteht beim Lesen der juristischen Diskussionen im Internet der Eindruck, als müsste man in Deutschland höchstens mit einer Geldstrafe rechnen.

2.10 Projekt Honeypot

Einen weiteren Schritt zum Aufhellen des Spam-Mail Phänomens stellt das Projekt „honeypot“ dar, welches im November 2004 von Unspam ins Leben gerufen wurde. Bei diesem Projekt werden Köder auf Internetseiten ausgelegt, in denen E-Mail-Adressen stehen. Früher oder später besuchen Datendiebe diese Internetseiten und sammeln die E-Mail-Adresse, entweder, wenn sie selbst Spammer sind, um diese in ihre Datenbank aufzunehmen und Spam-Mails zu verschicken oder um mit den potentiellen Spam-Adressen zu handeln, sie an andere „Geschäftsleute“ weiter zu verkaufen. Bei jedem Zugriff auf den „honeypot“ wird die zugreifende IP Adresse und Datum/Uhrzeit mitgeloggt. Alle Nachrichten, die an die Köder-E-Mail-Adressen geschickt werden, werden zu den Servern des Projektes „honeypot“ weitergeleitet und dort ausgewertet. Bisher beschränken sich die meisten Untersuchungen auf eine Richtung des Spam Zyklus, nämlich auf das Versenden der Spam-Mails zum „Verbraucher“. Mit Hilfe von „honeypot“ ist es aber möglich herauszufinden, wie lange es dauert, bis eine gestohlene E-Mail-Adresse mit Spam-Mails belästigt wird. Folgende Fragestellungen sollen aufgehellt werden:

- Wieviele Spam-Mails wird diese Adresse erhalten, abhängig davon, wie oft sie gelesen wurde?
- Wie lange bleibt die Köder-Adresse auf der Liste der Spammer?
- Sind die Personen, die die E-Mail-Adressen sammeln, auch die, die die Spam Mails verschicken?
- Wie weiträumig wird mit den E-Mail-Adressen gehandelt?

Das Projekt schafft neue rechtliche Möglichkeiten, Spammer zu überführen. Auch wenn die Debatte um die Definition von Spam-Mails natürlich je nach Interessenslage sehr unterschiedlich ausfällt (Direkt Marketing Verbände), ist jedoch unumstritten, dass das Sammeln und Verwenden von E-Mail-Adressen, mit denen nie eine geschäftliche Verbindung bestand, rechtswidrig ist. „Honeypot“ arbeitet mit rechtlichen Institutionen zusammen, um E-Mail-Adressen-Erntemaschinen unschädlich zu machen[35].

2.11 A Unified Model Of Spam Filtration

In einer Zeit des raschen Anstiegs von Spam-Mail und während einer hohen Geschwindigkeit der Spam-Evolution steigt auch der Forschungsaufwand dementsprechend schnell. Das bedeutet, dass es viele parallel entstehende Ansätze gibt und nicht alle Arbeiten ineinandergreifen oder sich detailliert mit dem auseinandergesetzt haben, was bereits existiert. So gibt es zahlreiche Verfahren und

auch Implementierungen, die sich bis auf Feinheiten kaum voneinander unterscheiden, deren Performance deshalb auch vergleichbar ist. In [4] wurde versucht, ein möglichst allgemeines Spam-Filter-Modell zu entwickeln, mit dem sich durch Festlegen der freien Parameter des Modelles alle wichtigen Spam-Filter emulieren lassen sollen. So unterscheidet „A Unified Model Of Spam Filtration“ verschiedene Stufen in der Filtersystematik.

Dieses Modell dient in Kapitel 3 als Rahmen für die Einordnung der beiden zu vergleichenden Algorithmen Grahams *Plan for Spam* und einem Naive-Bayes-Klassifikator. Das Skript beginnt mit der *initialen Transformation* des eingehenden Textes. Diese initiale Transformierungsfunktion ist in vielen Fällen die Identität. In diese Stufe gehören Vorverarbeitungen wie Zeichensatz-Transformationen, die für den Endbenutzer am sinnvollsten erscheinen. Der typische Fall ist hier, alles in ASCII Code ohne Berücksichtigung von Akzenten umzuwandeln. Man kann Groß- und Kleinschreibung verändern oder normalisieren, MIME Kodierungen entschlüsseln und in eine vom Filter fassbare Form bringen. Weiterhin ist es z.B. möglich, Verwirrungstricks der Spammer zu entschlüsseln, die darauf beruhen, dass Zufallswörter eingestreut werden, die aber durch ihre Farbe nicht sichtbar werden. So haben die Zufallswörter, die nur den Bayes-Filter verwirren sollen, dieselbe Farbe wie der Hintergrund und sind damit unsichtbar. Die Wörter, die für den Leser bestimmt sind, haben eine Farbe, die man erkennt. Eine Art optischer Filter könnte hier die Sicht des Menschen simulieren, um nur den Teil in den Filter zu lassen, den der Mensch auch wirklich erkennt. Und das kommt der potentiellen Werbebotschaft näher als das, was er nicht sieht. Darüberhinaus kann man in diesem Schritt auch Ersatzzeichen, die ähnlich aussehen, wieder in die ursprünglichen Zeichen zurücktransformieren, wie „@“ zu „a“. Schließlich könnte man versuchen, in Bildformaten durch Schrifterkennung und maschinelles Sehen Werbebotschaften zu erkennen, was aber bisher nach dem Wissen des Autors noch nie implementiert wurde, eventuell, weil auch die Spammer es noch nicht nötig haben, auf solche Tricks zurückzugreifen.

Schritt zwei wird als *Tokenization* bezeichnet. Dieser Schritt beschreibt die Art und Weise, wie der Text in Zeichenketteneinheiten (Tokens) zerlegt wird. Danach wird in der *Tupel basierten Kombination* festgelegt, wie die Tokens zu Features umgewandelt werden. Hierbei werden aus dem Tokenfluss Features extrahiert, wobei zunächst der Textfluss im Schritt Tokenization mittels eines regulären Ausdrucks in einzelne semantische Buchstabenketten zerlegt wird. Diese werden dann durch eine Token-Pipeline geschoben, und dabei werden jeweils N Token gleichzeitig über eine Funktion zu einem Feature verarbeitet. Somit bildet sich aus dem eingehenden Token Strom ein ausgehender Feature Strom.

Bei den von mir betrachteten Bayesschen Ansätzen ist dieses $N = 1$, das bedeutet, die einzelnen Wörter sind auch gleichzeitig die Features, in unserem Fall nur repräsentiert durch die Häufigkeit, mit der sie in den Trainingsdaten vorkommen. Damit ist bei den Bayes-Ansätzen, die nur einzelne Wörter verwenden, die Wahrscheinlichkeit von „real“ und „estate“ gleich der Kombination der Einzel-

wahrscheinlichkeiten der beiden Wörter, wobei der Kontext außer Acht gelassen wird.

Im nächsten Schritt *Feature Weighting* werden die Gewichte für die ermittelten Features bestimmt, im Sinne von Bayesschen Verfahren normalerweise einfach die Anzahl der Vorkommen des entsprechenden Features in den Trainingsdaten. Danach folgt die *Weight Combination*, die Ermittlung des „Gesamtgewichtes“ einer E-Mail. Diese Bewertung bedeutet wie spammig eine E-Mail ist. Es kann sich dabei um eine Wahrscheinlichkeit handeln, aber auch um eine Zahl > 1, wie beispielsweise beim Spamassassin Filter. In einem *Final Thresholding* wird dann über das „Gesamtgewicht“ und einen Schwellenwert bestimmt, ob es sich um Spam, Ham oder einen unsicheren Fall handelt. In dem Artikel werden als Instantiierungen die verschiedenen Spam-Filter in einer Art Sprache dargestellt, die die freien Parameter des abstrakteren Modells festlegt.

Kapitel 3

Bayessche Verfahren zur Spam-Mail Bekämpfung

Dieses Kapitel enthält eine Zusammenfassung des Gebietes Textklassifikation mit Methoden des maschinellen Lernens. Der Bereich wird dann auf die Anwendung des Spam-Filterns eingegrenzt, und es werden die besonders schwierigen Anforderungen an die maschinellen Lernverfahren für das Spam-Filtern aufgezeigt. Dann wird der Algorithmus Grahams *Plan for Spam* mit einem Naive-Bayes-Textklassifikator verglichen und die Gemeinsamkeiten und Unterschiede werden diskutiert, um die spätere Interpretation der Testreihen zu ermöglichen. Dabei wird jeweils auf den Kern der Algorithmen eingegangen, der für die Unterscheidung wichtig ist. Abschnitt 3.5 vermittelt daraufhin eine Übersicht auch über die Algorithmen in ihrer Gesamtheit nach dem Modell aus Abschnitt 2.11.

3.1 Grundlagen der Textklassifikation mit Methoden des maschinellen Lernens

Unter maschinellem Lernen versteht man den Versuch, Techniken zu entwickeln, die es Computern ermöglichen, aus irgendeiner Art von Daten zu lernen. Diese Daten können je nach Anwendungszweck sehr verschieden sein. Typische Anwendungsfelder sind Suchmaschinen, Medizinische Diagnosen, Finanzanalyse, Klassifikation von DNA Sequenzen, Sprach- und Handschrifterkennung, intelligent handelnde Mitspieler in Computerspielen und Robotersteuerung. Im Bereich Information Retrieval wird versucht, Mengen von Dokumenten auf Grund von inhaltlichen Aspekten zu verwalten. Die Bedeutung von solchen Konzepten ist seit 1990 sprunghaft angestiegen, begründet durch die ebenso rasante Zunahme an digital verfügbaren Dokumenten. Ein Teilgebiet des Information Retrieval ist die Textklassifikation. Die Forschung in der Textklassifikation kann zum Teil direkt zur Spam-Mail Klassifikation verwendet werden, denn die Unterscheidung zwischen Spam- und Ham-Mails ist ja nichts weiter als die Klassifizierung von Texten

(E-Mails) in zwei Kategorien. Ebenso wie in allen anderen Feldern des maschinellen Lernens, ist es zunächst nötig, die Daten in eine für die Lernverfahren adäquate Form zu bringen. Das ist die Aufgabe des Feature Engineerings. In diesem Schritt wählt man eine der Effizienz des Lernverfahrens möglichst zuträgliche Repräsentation der Daten.

Nach einem Überblick und der Erklärung der Grundbegriffe werden Bayessche Filteransätze vorgestellt und Paul Grahams *Plan for Spam* mit einem Naive-Bayes-Klassifikator verglichen. Dabei dient das verallgemeinerte Spam-Filter Modell aus Abschnitt 2.11 als Rahmen. Der Überblicksartikel [15] diente als Orientierung für die Einführung in maschinelle Lernverfahren in der Textklassifikation im Allgemeinen und Spam-Filter im Speziellen, bevor näher auf die beiden zu untersuchenden Algorithmen eingegangen wird. Ziel ist, herauszufinden, ob Grahams *Plan for Spam* tatsächlich eine zeitgemäße und effiziente Antwort auf die Spam Problematik ist, bzw. ob Grahams heuristisch anmutender *Plan for Spam* einen theoretisch fundierten Naive-Bayes-Algorithmus in der Leistungsfähigkeit übertreffen kann.

Textklassifikation lässt sich als Funktion auffassen, die jedem Pärchen $(d, c) \in D \times C$ einen booleschen Wert zuweist. Dabei ist D die Menge der Dokumente und C die Menge der Kategorien. Es handelt sich also um eine Funktion

$$\Phi^* : D \times C \to \{T, F\} \tag{3.1}$$

Wobei ein Pärchen bestehend aus Dokument und Kategorie eben dann TRUE zugewiesen bekommt, wenn das Dokument dieser Kategorie angehört. Φ^* ist dann die Funktion, die beschreibt, wie die Klassifikation wirklich ist. Zu bedenken ist, ob es diese Funktion im Normalfall überhaupt gibt, denn die Einordnung von Texten in Kategorien ist sehr subjektiv, was auch das Phänomen der „inter-indexer inconsistency“ zeigt. Dieses Phänomen besagt, das selbst hochqualifizierte Experten einer Domäne sich in der Einordnung von Texten aus dieser Domäne in Kategorien häufig gänzlich uneinig sind. Eine formale Definition von semantischen Kategorien ist sehr schwierig. Es wird versucht, mit Hilfe von maschinellen Lernverfahren die Funktion Φ, die die Funktion Φ^* möglichst gut approximiert, zu lernen. Φ wird Klassifizierer, Hypothese oder Modell genannt.

Die Genauigkeit der Übereinstimmung zwischen Φ und Φ^* repräsentiert die Effektivität des Klassifizierers. Man sollte unterscheiden ob ein zu klassifizierendes Dokument nur in genau eine Kategorie (single-label) oder in mehrere Kategorien (multi-label) eingeordnet werden darf. Für Spam-Filter findet das Modell der binären Kategorisierung Anwendung, da eben nur in die beiden Klassen Spam und Ham eingeteilt wird. Mit binärer Kategorisierung lassen sich alle anderen Kategorisierungen aber darstellen. Ein Multi-Label Problem mit n Kategorien lässt sich mit n binären Kategorisierungsproblemen repräsentieren. Weiter lässt sich unterscheiden, ob man einem Dokument Kategorien zuweisen möchte (document-pivoted categorization), oder alle Dokumente finden möchte, die in eine Kategorie

gehören (category-pivoted categorization). Das Problem des Spam-Mail Filterns ist sinnvollerweise als document-pivoted categorization Problem einzuordnen, da eine E-Mail nach der anderen in chronologischer Reihenfolge auftreten und den zwei Kategorien zugeordnet werden müssen. In der Textklassifizierung allgemein ist auch von harter versus bewerteter Kategorisierung die Rede. Mit hart ist gemeint, dass ein Dokument einer Kategorie eindeutig zugeordnet wird. Bewertete Kategorisierung meint, dass jedem einzuordnenden Dokument für jede Kategorie ein Zahlenwert zugeordnet wird. Somit kann eine letztendliche Einordnung einem Menschen überlassen werden. Eine bewertete Kategorisierung kann durch Einführung eines Schwellenwertes oder einer Ordnung auf die Bewertungen in eine harte Kategorisierung umgewandelt werden. Beide Möglichkeiten treten in der Spam-Bekämpfung auf. So ist ein Bayes Klassifizierer eine bewertete Kategorisierung (Wahrscheinlichkeit) mit einer Ordnung, die Kategorie, die die höhere Wahrscheinlichkeit für das Dokument aufweist, wird verwendet. Man kann aber auch einfach alle E-Mails nach eben dieser Wahrscheinlichkeit sortieren, was den Vorteil für den Benutzer hat, dass die Ham-Mails mit höchster Wahrscheinlichkeit oben stehen.

Bis 1990 wurde im Information Retrieval vorwiegend der Knowledge-Engineering-Ansatz favorisiert, bei dem Domänen-Experten manuell Regeln erstellten, die die Dokumente sortieren. Dabei wurden oft boolesche Formeln in disjunktiver Normalform verwendet, die in irgendeiner Art die vorher generierten Features aus den Texten verwerten, um anschließend die Kategorisierung vorzunehmen. Der Vorteil von induktiven Lernern, die aus Beispielen lernen, liegt aber klar auf der Hand. Man braucht keine Domänen-Experten, die die Regeln erstellen, sondern man braucht nur eine Datenbasis, in der bereits klassifizierte Dokumente vorliegen. Anwender, die von Knowledge-Engineering-Ansätzen auf maschinelle Lernverfahren umsteigen, haben den Vorteil, dass sie meist über große Mengen an bereits klassifizierten Daten verfügen, die sie als Basis für die maschinellen Lernverfahren verwenden können. Doch selbst wenn keine vorhandene Datenbasis existiert, ist es meist einfacher, eine Datenbasis zu erstellen, in dem man einige Dokumente manuell klassifiziert, als explizite Regeln aufzustellen. Genau wie ein Mensch relativ leicht Wörter in einer Taxonomie anordnen kann, soll er aber durch explizite Regeln erklären, warum, wird es deutlich schwieriger. Das liegt unter anderem an der „schwammigen“ Denkweise des menschlichen Gehirns (fuzzy), die sich auch in der menschlichen Sprache abbildet.

Eine Datenbasis besteht aus Dokumenten und der richtigen Klassifizierung dieser Dokumente, die man entweder neu manuell vornimmt oder aus vergangenen Daten besitzt. Diese Daten teilt man zum Trainieren der Klassifizierer in Trainingsmenge und Validierungsmenge auf. Der Klassifizierer wird mit den Dokumenten aus der Trainingsmenge trainiert. Daraufhin wird die Validierungsmenge mit dem Klassifizierer kategorisiert. Das Ergebnis dieser Kategorisierung wird mit dem wirklichen gewünschten Ergebnis verglichen, aus der Abweichung lässt sich die Generalisierungsleistung des Klassifizierers ermitteln. Bei der Aufteilung

von Trainings- und Validierungsmenge hat sich oft 2/3 zu 1/3 als sinnvolle Heuristik herausgestellt. Manchmal werden auch aus der Gesamtmenge drei Mengen erstellt, eine zum Trainieren, eine um die Feineinstellung der Parameter des Klassifizierers vorzunehmen (z.B. Schwellenwerte) und eine um zu validieren. Durch die Änderung der Aufteilungen lassen sich die Grenzen zwischen erkannten Gesetzmäßigkeiten und Auswendiglernen der Daten variieren. Das hängt natürlich auch sehr stark mit den gewählten Features zusammen.

3.2 Spam-Filtering als Klassifikationsproblem

Die Problematik Spam-Filtern beinhaltet besondere Schwierigkeiten, die in den angewendeten Verfahren und Evaluationen des maschinellen Lernens berücksichtigt werden sollten. Wenn man das Problem lediglich als reines Textklassifikationsproblem im herkömmlichen Sinne betrachtet, löst man viele der Anforderungen bei weitem nicht.

In wissenschaftlichen Arbeiten über Spam-Filter Validierung finden sich Spam-Anteile von 16.6% bis 88.2%. Es ist nicht klar, wie diese Verteilung idealerweise aussehen sollte, auch auf Grund der Tatsache, dass jede E-Mail-Adresse verschieden viel Spam erhält. Der Spam-Anteil variiert auf Grund der Dauer der Existenz einer E-Mail-Adresse, der Häufigkeit ihrer Veröffentlichung im Internet und der eventuell vorgeschalteten Spam-Filter beim Mail-Provider. Damit ist eine Fixierung der A-priori-Wahrscheinlichkeiten einer Klassenverteilung gar nicht möglich, von der aber zahlreiche Verfahren ausgehen. Sowohl die Anzahl der Spam-Mails als auch der Ham-Mails schwanken stark von Tag zu Tag, ebenso stark schwanken die A-priori-Wahrscheinlichkeiten dass es sich bei einer E-Mail um Spam oder Ham handelt. Somit kann ein Klassifizierer mit einer angenommenen A-priori-Wahrscheinlichkeit schon auf Grund von starken Schwankungen in der Zahl der Spam-Mails in verschiedenen Zeiträumen völlig unterschiedliche Ergebnisse liefern. Solche Schwankungen entstanden z.B. in 2002 als eine große Menge offener Mail-Server in China und Korea online gingen. Daraufhin wurde ein dramatischer Anstieg der Spam Zahlen festgestellt.

Ebenso schwierig zu beurteilen sind die Fehlerkosten, die auch noch von der subjektiven Einschätzung der Benutzer abhängen. Dabei sind sich aber die meisten einig, dass bei *false positives* (Ham als Spam klassifiziert) viel höhere Fehlerkosten entstehen als bei *false negatives*. Das geht sogar soweit, dass der Sinn des Spam-Filterns für manche völlig verloren geht, wenn auch nur eine einzige wichtige Ham-Mail als Spam aussortiert wird. Somit ist die Fassbarkeit der Fehlerkosten in Zahlen kaum möglich. Die Angaben darüber, wie viel mal schlimmer *false positives* gegenüber *false negatives* einzuschätzen sind, schwanken zwischen zwei und hundert. Auch das erschwert eine Vergleichbarkeit und Validierung von verschiedenen Lösungen, denn wer legt im Endeffekt fest, welche false positive Raten zulässig sind?

Nicht nur die Anzahl von Spam-Mails schwankt, sondern auch die Inhalte verändern sich drastisch. So kommt jedes Jahr um die Weihnachtszeit das Wort „christmas“ deutlich häufiger vor als sonst. Das Phänomen „concept drift“ wurde bereits 1996 untersucht[34]. Neben solchen zyklischen Erscheinungen (deren Zyklusdauer auch sehr verschieden sein kann) gibt es auch neue Betrugsideen, die sich rasend schnell verbreiten und zahlreiche Nachahmer finden, bis die Empfänger durch Informationen und Aufklärung dagegen immun werden. So geschehen mit den Mails der „Nigeria Connection“. Es gibt auch Wörter und Themen, die sich sehr konstant über die Zeit verhalten. Für die Veränderungen der Spam-Mails über die Zeit gibt es noch keine einheitlichen Messmethoden.

Ebenso herausfordernd wie bisher in der Textklassifikation nicht üblich sind die adaptiven Gegner. Bei der Textklassifikation von Datenbeständen in Firmen ist es normalerweise nicht üblich, dass jemand versucht, die Klassifikation ständig zu erschweren. Meist tritt dies im Zusammenhang mit kriminellen Aktivitäten auf, wenn eine Partei versucht, den Zugang zu einer Ressource zu erhalten und die andere Partei versucht, dies zu verhindern.

3.3 Paul Grahams „Plan for Spam“

Paul Grahams erfolgreicher Anti-Spam-Algorithmus *A Plan for Spam* ist ein heuristisch erstellter Algorithmus, der mit an die Wahrscheinlichkeitstheorie angelehnten Ideen funktioniert. Dennoch arbeitet er nicht mit Wahrscheinlichkeiten, eher mit Verhältnissen und Approximationen. Deshalb ist er auch wenn in der Literatur oft anders beschrieben, nicht wirklich zu den Bayesschen Ansätzen zu zählen. Bayessche Spam-Filter-Ansätze sind solche, die auf aus den Wahrscheinlichkeitsaxiomen hergeleiteten Formeln basieren. Die Terminologie, die bei der folgenden Erklärung der Funktionsweise von *A Plan for Spam* verwendet wird, wird in erweiterter Form auch im nächsten Abschnitt bei der Erklärung des Naive-Bayes-Textklassifikators verwendet.

Zum Training von *A Plan for Spam* verwendet man zwei Mail-Corpora, einen Spam-Corpus D_{spam} und einen Nicht-Spam-Corpus D_{ham}. Dann verkettet man diese beiden Corpora zu einem Mail-Corpus D und extrahiert aus diesem Text alle unterschiedlichen Wörter bzw. Zeichenketten W, das Vokabular. Mit der Formel

$$P_w = \frac{\frac{n_{w,spam}}{|D_{spam}|}}{\frac{2n_{w,ham}}{|D_{ham}|} + \frac{n_{w,spam}}{|D_{spam}|}} \tag{3.2}$$

berechnet man die „Spam-Wahrscheinlichkeit“ für jedes Wort des vorhandenen Vokabulars W. Dabei ist $n_{w,spam}$ die Anzahl der Vorkommen des Wortes w im Spam-Corpus, $n_{w,ham}$ dementsprechend die Anzahl der Vorkommen des Wortes w im Ham-Corpus. $|D_{spam}|$ ist die Anzahl der im Spam-Corpus vorhandenen Spam-Mails, $|D_{ham}|$ die Anzahl der Ham-Mails. Damit sind $\frac{n_{w,spam}}{|D_{spam}|}$ und $\frac{n_{w,ham}}{|D_{ham}|}$

die Vorkommen dieses Wortes pro Ham-/Spam-Mail. Graham fand heuristisch heraus, dass es sich zur Vermeidung von *false positives* anbietet, die Anzahl der guten Vorkommen zu verdoppeln. Deshalb findet sich in Formel 3.2 die zwei im Nenner.

Damit hat man für jedes Wort des Vokabulars eine Pseudo-Wahrscheinlichkeit. Wenn man nun eine E-Mail klassifizieren möchte, liest man die E-Mail ein, trennt sie nach Tokens auf (siehe Abschnitt 3.5) und liest die berechneten Wahrscheinlichkeiten für jedes Wort/Token aus, dass mindestens fünfmal im Training vorkam. Wörter, die weniger oft im Training vorkamen, werden mit Wahrscheinlichkeit 0.4 angesetzt. Graham bezieht in die Berechnung einer finalen Gesamtwahrscheinlichkeit genau die fünfzehn Tokens ein, deren Wahrscheinlichkeiten am weitesten weg von 0.5 liegen. Diese Wahrscheinlichkeiten kombiniert man nun mit

$$P_{final} = \frac{\prod_{i=1}^{15} P_i}{\prod_{i=1}^{15} P_i + \prod_{i=1}^{15}(1 - P_i)} \tag{3.3}$$

zu einer Endwahrscheinlichkeit. Sollte diese nach Graham über 0.9 liegen, wird die Nachricht als Spam-Mail klassifiziert.

3.4 Naive Bayes zur Textklassifikation

$$P(a|b) = \frac{P(b|a) * P(a)}{P(b)} \tag{3.4}$$

Bei der Textklassifikation aus [3] handelt es sich um einen häufig verwendeten Naive-Bayes-Ansatz. Prinzipiell ist ein Naive-Bayes-Lerner/Klassifizierer ein auf der statistischen Entscheidungstheorie basierender Klassifikator, dem eine Menge von Beispielen präsentiert wird, sprich eine Menge von Merkmalen $\langle a_1, a_2, \ldots, a_n \rangle$ mit den dazugehörigen Ergebnissen der gesuchten Zielfunktion. Die möglichen Werte der Zielfunktion sind dabei aus einer endlichen Menge V. Beim bayesschen Ansatz allgemein versucht man den erwarteten Wert der Zielfunktion zu schätzen, indem man die Wahrscheinlichtkeit für jeden möglichen Zielfunktionswert $v \in V$ ausrechnet.

$$v_{MAP} = \underset{v \in V}{\operatorname{argmax}} P(v|a_1, a_2, \ldots, a_n) \tag{3.5}$$

beziehungsweise umgeformt nach Bayes-Theorem (3.4)

$$v_{MAP} = \underset{v \in V}{\operatorname{argmax}} \frac{P(a_1, a_2, \ldots, a_n|v)P(v)}{P(a_1, a_2, \ldots, a_n)} \tag{3.6}$$

und damit, da der Nenner keine Rolle für $argmax$ spielt

$$v_{MAP} = \underset{v \in V}{\operatorname{argmax}} P(a_1, a_2, \ldots, a_n|v)P(v) \tag{3.7}$$

Die Werte können anhand der Trainingsdaten geschätzt werden. $P(v)$ lässt sich einfach bestimmen, in dem man zählt, wie oft ein Zielfunktionswert in den gesamten Trainingsdaten vorkommt. $P(a_1, a_2...a_n|v)$ lässt sich nur bei extrem grossen Mengen von Trainingsdaten bestimmen, was in der wie im folgenden beschriebenen Textklassifikation unmöglich scheint. Wenn wir aber die vereinfachende Annahme treffen, dass die Merkmale untereinander konditional unabhängig sind, so können wir obige Formel auf die vereinfachte Variante des Naive-Bayes-Klassifikators ändern:

$$v_{NB} = \underset{v \in V}{\operatorname{argmax}} P(v) \prod_i P(a_i|v) \tag{3.8}$$

Dabei sind v_{MAP} und v_{NB} genau dann identisch, wenn die Merkmale wirklich alle konditional unabhängig sind.

Nun gehen wir anhand von Formel 3.8 konkreter auf den Anwendungsfall zur Textklassifikation von E-Mails ein. Im Falle von Spam-Mail-Filtern wäre also $P(v)$ der Prozentanteil von Spam-Mail und Ham-Mail an der gesamten Mail-Menge. Die Terminologie aus Abschnitt 3.3 wird nun noch etwas erweitert. $P(v)$ bezeichnet man auch als A-priori-Wahrscheinlichkeit:

$$P(v) = \frac{|D_v|}{|D|}, v \in \{spam, ham\} \tag{3.9}$$

Dabei ist $|D_v|$ die Anzahl aller Spam-/Ham-E-Mails und $|D|$ die Anzahl aller E-Mails zusammen. Nun fehlen uns zum berechnen mit Formel 3.8 noch die Werte $P(a_i|v)$. Diese werden mit dem *m-estimate of probability* [3] abgeschätzt.

Jetzt berechnet man wie in Abschnitt 3.3 die Spam/Ham-Wahrscheinlichkeit für jedes Wort des vorhandenen Vokabulars W mit den Formeln

$$P(w|spam) = \frac{n_{w,spam} + 1}{n_{spam} + |W|} \tag{3.10}$$

bzw.

$$P(w|ham) = \frac{n_{w,ham} + 1}{n_{ham} + |W|} \tag{3.11}$$

Wieder ist $n_{w,spam}$ die Anzahl der Vorkommen des Wortes w im Spam-Corpus, $n_{w,ham}$ die Anzahl der Vorkommen des Wortes w im Ham-Corpus. n_{ham} bzw. n_{spam} ist die Anzahl der *Wortpositionen* im Ham-/Spam-Corpus. $|W|$ ist die Anzahl der Worte im Vokabular.

Wenn man also eine E-Mail klassifizieren will, trennt man die E-Mail wieder in Tokens auf und fragt für jedes Token die vorher berechneten $P(w|spam)$ und $P(w|ham)$ ab. Dann berechnet man basierend auf der Formel 3.8 die Klasse, in die die E-Mail am wahrscheinlichsten gehört:

$$v_{NB} = \underset{v \in \{spam, ham\}}{\operatorname{argmax}} P(v) \prod_{j \in positions} P(w_j|v) \tag{3.12}$$

Dabei ist w_j das Wort an der Stelle j in der zu klassifizierenden E-Mail. Falls w_j nicht im Vokabular vorkommt, wird es nicht berücksichtigt. Anstatt einfach die wahrscheinlichere Zielfunktionsklasse zu berechnen (siehe Formel 3.12), kann man die beiden Wahrscheinlichkeiten für die Zielfunktionsklassen (siehe Formel 3.13) auch in einen Wert umrechnen (siehe Formeln 3.15 und 3.14), auf den man dann einen Schwellenwert anwenden kann. Die Formeln 3.15 und 3.14 stellen darauf folgend die normalisierten Wahrscheinlichkeiten dar.

$$P(v|d) = P(v) \prod_{j \in positions} P(w_j|v), v \in \{spam, ham\}, \qquad d \in D \tag{3.13}$$

$$P_{ham} = \frac{P(ham|d)}{P(spam|d) + P(ham|d)} \tag{3.14}$$

$$P_{final} = P_{spam} = \frac{P(spam|d)}{P(spam|d) + P(ham|d)} \tag{3.15}$$

3.5 Übersicht Paul Grahams „Plan for Spam“ versus Naive-Bayes-Textklassifikator

Der folgende Vergleich der beiden Algorithmen bezieht sich zum Teil auf die konkrete Implementierung der Algorithmen im Mozilla-Mail-Programm (siehe Abschnitt 3.5). Die Vorverarbeitungsschritte für beide Algorithmen in der Implementierung sind also die aus dem Mozilla-Mail-Programm (siehe Tabelle 3.1). Der Vergleich der beiden Algorithmen reduziert sich somit auf einen Vergleich der Schritte *Feature Weighting*, *Weight Combination* und *Final Thresholding*. Die Tabelle 3.1 stellt beide Algorithmen in der Übersicht dar. Zur Erläuterung der Terminologie in der Tabelle siehe Abschnitte 3.3 und 3.4.

Im Schritt *Feature Weighting* verwendet Graham für seine „Wortwahrscheinlichkeiten“ ein normalisiertes Maß was aus den Vorkommen eines Wortes pro Spam-/Ham-Mail berechnet wird. Da dabei verschiedene „Maßeinheiten“ verwendet werden, nämlich Worthäufigkeiten und Anzahlen der E-Mails in den Corpora, handelt es sich hierbei nicht um Wahrscheinlichkeiten. Trotzdem liegen diese Werte auf Grund der Normalisierung wie Wahrscheinlichkeiten zwischen Null und Eins. Man könnte Grahams Maß als direkte Schätzung der Wahrscheinlichkeitswerte $P(spam|w)$ ansehen, ohne die vom Naive-Bayes-Algorithmus verwendete Vertauschung der Kausalzusammenhänge nach Bayes Theorem (3.4).

Somit lässt sich durch Weglassen der A-priori-Wahrscheinlichkeiten in Form von $|D_{spam}|$ bzw. $|D_{ham}|$ und der heuristischen Zwei im Nenner folgende Vereinfachung erreichen:

$$P_w = \frac{\frac{n_{w,spam}}{|D_{spam}|}}{\frac{2n_{w,ham}}{|D_{ham}|} + \frac{n_{w,spam}}{|D_{spam}|}} \tag{3.16}$$

$$\approx \frac{n_{w,spam}}{n_{w,ham} + n_{w,spam}} \tag{3.17}$$

$$\approx P(spam|w) \tag{3.18}$$

Dabei ist Formel 3.17 der %-Anteil der Vorkommen dieses Wortes in Spam.

Der Naive-Bayes-Algorithmus verwendet hier echte Worthäufigkeiten als Wahrscheinlichkeiten in geringfügig abgewandelter Form, nämlich das *m-estimate of probability* [3].

Graham berechnet seine „Endwahrscheinlichkeit“ beim *Feature Weighting*, in dem er nur die 15 „Wortwahrscheinlichkeiten“ kombiniert, die am weitesten entfernt von 0.5 liegen. Der Naive-Bayes-Algorithmus leitet seine Klassenwahrscheinlichkeiten herkömmlich aus den Wahrscheinlichkeitsaxiomen her und verwendet die Einzelwahrscheinlichkeiten aller Wörter, die trainiert wurden.

Beim *Final Thresholding* verwendet Graham einen heuristisch ermittelten Wert von 0.9, der Naive-Bayes-Klassifikator sollte bei einer geraden Umverteilung der zwei Klassenwahrscheinlichkeiten normalerweise 0.5 als Schwellenwert verwenden. Wegen der schiefen Kostenfunktion, sprich, weil *false positives* teurer sind als *false negatives*, ist aber ein höherer Wert sinnvoller.

Was sind nun die entscheidenden Unterschiede, die die deutlichen Differenzen in der Performance der beiden Algorithmen im Spam-Umfeld erklären? Da wäre zunächst zu nennen, dass Graham nur die 15 „Wortwahrscheinlichkeiten“ verwendet, die am weitesten entfernt von 0.5 liegen. Im Naive-Bayes-Algorithmus werden aber die Wahrscheinlichkeiten aller Wörter kombiniert, die im Trainings-Corpus vorkamen. Somit ist es naheliegend, zu vermuten, dass zur Unterscheidung von Spam- und Ham-Mail nur wenige wichtige Indikatoren sehr ausschlaggebend sind. Somit wäre zu untersuchen, warum genau 15 Wörter im *Plan for Spam* verwendet werden. Sehr oft werden die freien Parameter in Grahams Artikel nach dem „Konzept Trial and Error“ (Herumprobieren) ermittelt. Hier wäre es sinnvoll, zeitgemäße Heuristiken über die freien Parameter laufen zu lassen und die Experimente mit hoher Rechenleistung oft zu wiederholen, um die optimalen Parameter zu bestimmen.

Es scheint einen Zusammenhang zu geben zwischen der Auswahl der Wörter, die man für die Klassifizierung überhaupt in Betracht zieht und der semantischen Separierbarkeit von Texten. Es ist schwer zu sagen, wieviele „Dimensionen semantischer Separierbarkeit“ es gibt und wie die optimale Konfiguration eines Klassifizierers dazu aussehen sollte. Vermutlich ist das Feature-Engineering und die

Auswahl der präsentierten Daten wichtiger als das maschinelle Lernverfahren, das darauf arbeitet. Somit scheinen die Performance-Unterschiede hauptsächlich darauf zurückzuführen sein, dass Graham das reale Modell der semantischen Unterscheidung besser abbildet als ein herkömmlicher Naive-Bayes-Algorithmus. Dies liegt meines Erachtens nach eben an der Begrenzung auf 15 Wörter und daran, dass bei Graham Wörter überhaupt nur eine Chance haben mit in die Berechnung einzugehen, nämlich in dem Sie mindestens 5 mal im Trainingscorpus aufgetaucht sind. Zusätzlich kann es möglich sein, dass das Feature-Engineering, in dem genau die Wörter die Features sind, ungeeigneter für den Naive-Bayes-Algorithmus ist, weil auch sehr selten auftauchende Wörter, die eher als „Rauschen“ aufzufassen sind, trotzdem in die Berechnung mit eingehen. Hier könnte auch eine Alternative zum *m-estimate of probability*[3] Verbesserungen bringen. Siehe dazu auch die Ausführungen in [4] unter *Feature Weighting*.

In Kapitel 6 werden die obigen Vermutungen im Zusammenhang mit den Testreihen diskutiert.

Schritt	PFS	NB
Initiale Transformation	alles in Kleinschrift transformieren, HTML-Kommentare und Zahl-Tokens entfernen	alles in Kleinschrift transformieren, HTML-Kommentare und Zahl-Tokens entfernen
Tokenization	alles außer `-'$ a-z` sind Trennzeichen	alles außer `-'$ a-z` sind Trennzeichen
Tupel basierte Kombination	jedes Token = ein Feature	jedes Token = ein Feature
Feature Weighting	$P_w = \frac{\frac{n_{w,spam}}{\lvert D_{spam}\rvert}}{\frac{2n_{w,ham}}{\lvert D_{ham}\rvert}+\frac{n_{w,spam}}{\lvert D_{spam}\rvert}} \approx P(spam\lvert w)$ nur mind. 5x aufgetretene Wörter	$P(w\lvert spam) = \frac{n_{w,spam}+1}{n_{spam}+\lvert W\rvert}$ $P(w\lvert ham) = \frac{n_{w,ham}+1}{n_{ham}+\lvert W\rvert}$
Weight Combination	$P_{final} = \frac{\prod_{i=1}^{15} P_i}{\prod_{i=1}^{15} P_i + \prod_{i=1}^{15}(1-P_i)}$	$P(v\lvert d) = P(v)\prod_{j\in positions} P(w_j\lvert v)$, $v \in \{spam, ham\}, \quad d \in D$ $P_{final} = \frac{P(spam\lvert d)}{P(spam\lvert d)+P(ham\lvert d)}$
Final Thresholding	Spam falls $P_{final} > 0.9$	Spam falls $P_{final} >$ X

Tabelle 3.1: Übersichtstabelle Paul Grahams „Plan for Spam“ versus Naive-Bayes-Textklassifikator

Kapitel 4

Implementierung der Algorithmen in Mozilla

In folgenden Ausführungen werden der Mozilla Browser und generell die Open-Source-Software abgehandelt. Danach werden Implementierungsdetails der untersuchten Spam-Filter in das Mozilla-Browser-Paket und die damit verbundenen praktischen Aspekte erläutert. Falls kein Interesse an Implementierungsdetails besteht, kann dieses Kapitel übersprungen werden.

4.1 Mozilla und die „Open Source Welt“

Netscape Communications Corporation gab im Frühjahr 1998 den Netscape Communicator Quellcode frei, um angesichts der steigenden Marktdominanz des Microsoft Internet Explorers die Weiterentwicklung der Netscape Suite als Open-Source-Projekt sicherzustellen. Zu diesem Zeitpunkt wurde auch Mozilla.org gegründet, eine virtuelle Organisation, die die Aktivitäten der Mozilla-Gemeinschaft organisieren sollte. Mehr und mehr entwickelte sich die Mozilla-Organisation zu einem echten Open Source Projekt.

In der Anfangszeit war das Projekt noch hauptsächlich mit Netscape und dessen Mitarbeitern verknüpft, später verselbständigten sich zahlreiche Mozilla-nahe Organisationen weltweit. Der Ausdruck „Open Source Software“ ist eng verwandt mit „freier Software“. In Internetforen wird (siehe [43]) sehr viel über genaue Definitionen und Begrifflichkeiten gestritten, wobei der Eindruck entsteht, als gäbe es nicht die eine Open Source Software, sondern viele verschiedene Formen und Vorstellungen von Konzepten, die alle der „Open-Source-Ecke“ angehören. So sei auf folgende wichtige Organisationen und die von ihnen vertretenen Lizenzarten hingewiesen: die Free Software Foundation (FSF) und die Open Source Initiative (OSI). Es gibt Standard Lizenzmodelle wie die GNU General Public Licence (GPL), die die Intention des Machers oder Gründers (z.B. von GNU) wiedergeben. So kann theoretisch jeder seine eigene Lizenzvereinbarung für seine Software

entwickeln, wobei es sinnvoll erscheint, Standardlizenzmodelle zu verwenden, die schon zahlreichen juristischen Verfahren standgehalten und sich bewährt haben. In diesen Lizenzen werden vor allem die freie Verfügbarkeit des Quellcodes, die freie Benutzung der Programme in privatem oder kommerziellem Sinne und die Weiterentwicklung geregelt. So gibt es Programme, die wirklich jeder zu jedem Zweck frei verwenden, völlig frei weiterentwickeln und sogar in kommerzielle Projekte mit einbringen und diese dann wieder vermarkten darf. In manchen Fällen muss jede Änderung des Quellcodes dem Autor des Originals mitgeteilt werden. In anderen Fällen wiederum darf man zwar den Code beliebig verändern, aber nicht in kommerziellen Projekten einsetzen. Das alles wirft komplexe und neuartige juristische Herausforderungen auf. Als wichtige Beispiele in der Entwicklung der Open Source Welt seien GNU [37], Linux in seinen verschiedensten Distributionen (SuSe, Red Hat, Debian...) ,die grafischen Oberflächen KDE und GNOME für Linux, die Mozilla-Organisation sowie der Webserver Apache [39] genannt. In der Entwicklung von Linux hat sich in den vergangenen Jahren viel getan. Das klassische Vorurteil der schwierigen Installation und Handhabung scheint angesichts von Suse & Co. nicht mehr zeitgemäß. Erfahrungsgemäß kann berichtet werden, dass sich eine Suse Linux Installation in der Version 9.0 im Schwierigkeitsgrad nicht mehr stark von einer Windows Installation unterscheidet. Schwierigkeiten gibt es noch mit den grafischen Oberflächen, die leider auch nicht seltener als Windows abstürzen. Open Source Software wird mehr und mehr in der Wirtschaft eingesetzt, Behörden und Firmen aller Branchen setzen Linux Systeme ein. Es gibt mittlerweile kommerziellen professionellen Support für zahlreiche Open Source Produkte, als Beispiel sei die Datenbank MySQL genannt[40].

4.2 Vor- und Nachteile von Open-Source-Software

Ein gewaltiger Vorteil ist die Offenlegung des Quellcodes. Damit kann Spionage jeder Form, die bei Produkten mit nicht offen gelegtem Quellcode zumindest theoretisch problemlos möglich ist, komplett unterbunden werden. In einer Art gut organisierter Anarchie in den Weiten des Internets, entstehen „lebende“ Organismen und virtuelle Softwareprojekte, die oft Fehler schneller beheben als kommerzielle Softwarefirmen. Man kann von einer Art Softwareevolution sprechen, denn jeder kann den Quellcode ändern und verbessern. Schlechte Versionen sterben automatisch aus. Trotz alledem bedarf es auch in wenig hierarchischen Open-Source-Projekten zentraler Instanzen, die letztendlich die Entscheidungsgewalt darüber haben, was nun in die Release Version eingebaut wird und was nicht. Typisch ist z.B., experimentelle Releases herauszugeben, bei denen die Benutzer wissen, dass sie praktisch nur eine Testversion installieren und oftmals bereitwillig die Fehler an die passende Organisation zurückmelden. Warum solche

nicht auf Profit ausgerichteten Organisationen trotzdem teilweise so gut funktionieren, lässt sich eventuell auch durch soziale Phänomene erklären. So gibt es eine nicht zu unterschätzende Anzahl „Hacker“ und „Coder“ weltweit, die zu einem nicht unerheblichen Teil ihr Leben dem Computer oder dem Entwickeln von Software verschrieben haben. Hier macht es eventuell auch dem einen oder anderen Spaß, „den Großen“ wie Microsoft eins auszuwischen, indem sie konkurrenzfähige Produkte in Eigenregie entwickeln. Manche Programmierer sind auch sicher jenseits von finanziellen Interessen auf Bestätigung aus, wenn ihr Modul in Mozilla weltweit eingesetzt wird. Dies konnte man in der Computerwelt schon im Mailboxzeitalter Ende der 80er Jahre beobachten, als viele Menschen sehr viel Zeit auf nicht kommerzielle Projekte verwendeten. Für Informationen rund um die Open-Source-Gemeinde sei die Lektüre von [41] und [42] empfohlen. Open-Source-Software spielt eine zunehmende Rolle in den verschiedensten Bereichen; ganz aktuell ist die billigste Möglichkeit, an viel Rechenleistung zu kommen, ein Linux-Cluster mit herkömmlicher Standard PC-Hardware im unteren Preissegment. Nachholbedarf scheint es im Bereich betriebswirtschaftlicher Standardsoftware und beim kommerziellen Software-Support zu geben.

Bei einem großen Projekt wie Mozilla sind zahlreiche Instanzen verschiedener Hierarchie-Ebenen nötig, um das Projekt zu kontrollieren. So ist *www.mozilla.org* die zentrale Internetseite der Mozilla-Organisation und dient zum Aufstellen der technischen und architekturellen Vorgaben für die Weiterentwicklung von Mozilla, zum Sammeln von Änderungen sowie zur Kommunikation zwischen den Entwicklern. Hauptsächlich dient *www.mozilla.org* als Drehscheibe für die Organisation der Entwickler und koordiniert somit die Entwicklung aller weltweit beteiligten Mitarbeiter. Je nach Kenntnissen bekommen mitarbeitende Entwickler mehr Verantwortung und werden eventuell zu Modulbesitzern ernannt, die als Leiter für einen bestimmten Teilbereich fungieren. Ihre Aufgabe ist es, den eingehenden Code zu prüfen und zu integrieren. Die letzte Entscheidungsinstanz liegt dann beim „gerechten Diktator“ *mozilla.org*, meistens jedoch klären sich die Probleme von selbst, da jeder der Beteiligten ja nur das Beste für das Projekt erreichen will. Weiter gibt es Mozilla-Diskussionsforen in aller Welt und in (fast) allen Sprachen, auf denen rege die Benutzung aber auch Weiterentwicklung des Systems diskutiert wird.

4.3 Programmierpraxis in Mozilla unter Linux

In diesem Abschnitt soll dem Leser ein konkreter Eindruck von der Umgebung Mozilla Suite 1.7.3 unter dem Betriebssystem Suse Linux 9.0 gegeben werden.

Beim Wunsch sich mit dem Mozilla-Quellcode auseinanderzusetzen und diesen zu compilieren, sollte man sich zunächst bei *www.mozilla.org* im Bereich Entwicklung informieren. Als wichtigste Quellen und Kontaktmöglichkeiten mit der Entwicklergruppe von Mozilla eignen sich die passenden Kanäle im IRC Chat und

www.mozillazine.org, ein Forum, in dem man oft sinnvolle Antworten von Entwicklern bekommt. Eine ausgiebige Suche nach Schlüsselwörtern empfiehlt sich in jedem Fall, da die Entwicklergemeinde um das Mozilla-Projekt sehr aktiv ist und viele Probleme schon im Detail beschrieben sind.

Es stand für den Autor die Entscheidung an, das E-Mail-Programm aus der Mozilla Suite oder Thunderbird, das neue E-Mail-Programm von Mozilla, zu verwenden, um Grahams *Plan for Spam* mit einem Naive-Bayes-Klassifizierer in der Eigenschaft der Spam-Tauglichkeit zu vergleichen. Da sich die Mozilla Suite mit geringeren Problemen compilieren lies, fiel die Entscheidung dementsprechend aus. Es ist empfehlenswert, die Voraussetzungen, die zum Compilieren der Mozilla Suite auf *www.mozilla.org* gegeben werden, genauestens zu beachten. Selbst eine Differenz in der zweiten Nachkommastelle der Versionsnummer einer benötigten Bibliothek kann zu Problemen führen. Die meisten geforderten Voraussetzungen wurden von Suse Linux 9.0 erfüllt, teilweise war das Problem sogar, dass Versionen zu neu waren und eine Abwärtskompatibilität nicht vorhanden war. In diesem Fall hilft das Suchen nach RPM Paketen im Internet, z.B. bei *rpmfind.com*. Die RPM Pakete unterscheiden sich für die verschiedenen Linux Distributionen. Ein Debian RPM Paket läuft also nicht unbedingt auf Suse Linux. Somit kommt man meist mit etwas Geduld und eventuell ein paar Tests mit verschiedenen Versionen einzelner Bibliotheken zu dem Ziel, Mozilla zu übersetzen. Mozilla ist fast komplett in C++ geschrieben. Die Compiliermechanismen sind sehr komplex, es wird empfohlen einen Objektpfad zu benutzen, so dass `.o` Dateien und `.cpp` bzw `.h` Dateien nicht im selben Ordner liegen. Das kann man in der `.mozconfig` Datei (siehe unten) einstellen.

Wenn alle Voraussetzungen an das Betriebssystem erfüllt sind, führt man folgende Schritte durch:

1. Im Homeverzeichnis des Benutzers, der den Übersetzungsvorgang durchführt, muss eine Datei namens `.mozconfig` abgelegt werden. In dieser Datei werden zahlreiche Details des Übersetzungsvorgangs festgelegt.

2. Man lädt das gezippte TAR-Paket der entsprechenden Mozilla-Version herunter.

3. Man entpackt das TAR Paket mit `tar -xvf *.tar`.

4. Nun begebe man sich in das entstandene Mozilla-Verzeichnis. Hier führe man folgende Befehle aus:

   ```
   gmake -f client.mk build
   ```

   ```
   gmake -C objdir/xpinstall/packager
   ```

   `objdir` wird in der Datei `.mozconfig` selbst festgelegt, steht also für einen Namen, den man vorher frei gewählt hat.

5. Wenn alles funktioniert hat, findet man unter `/mozilla/objdir/dist/bin` die ausführbaren Dateien der Mozilla Suite.

Der gesamte Mozilla-Quellcode ist sehr gut organisiert, aber trotzdem auf Grund seiner Größe (allein 4220 .cpp Dateien) sehr schwer zu überschauen. Alles, was das Mail-Programm betrifft, befindet sich im Verzeichnis `mailnews`.

Im folgenden finden sich einige Angaben zum Vorgehen des Autors, um sich einen Überblick über den Quellcode zu verschaffen. Es sei jedem selbst überlassen, ob er unter Suse Linux eine grafische Programmierumgebung für C++ mit einem grafischen Debugger benutzt. Die Integration der Mozilla Suite in solch eine Umgebung sei aber nur Linux Compiler Experten zu empfehlen.

Es existieren im Quellcode zahlreiche Meta-Mechanismen zum Compilieren, die das Compilieren auf verschiedenen Plattformen und unter verschiedenen Voraussetzungen möglich machen sollen.

Der Ausgangspunkt war die Datei `nsBayesianFilter.cpp` in `mailnews/extensions/bayesian-spam-filter/src/`.

Wenn man sie geändert hat und neu compilieren möchte, geht man in das Objekt Verzeichnis unter `mailnews/extensions` und führt dort ein make aus. In manchen Fällen eventuell auch ein `make clean` und dann ein `make`, falls aus irgendwelchen Gründen das `makefile` nicht bemerkt hat, dass sich einzelne Dateien geändert haben.

Im Laufe der Einarbeitungsphase stellte sich ein extrem pragmatisches Vorgehen als sinnvoll heraus; man sollte nicht im Rahmen einer Diplomarbeit versuchen, sämtliche interne Mechanismen des Quellcodes zu verstehen. Das wichtigste Hilfsmittel war die *Datei suchen* Funktion auf der Kommandozeile oder als Fenster. Als erstes wurden in zahlreiche Dateien in die Konstruktoren der einzelnen Objekte `printf` Kommandos eingebaut, die in dem Terminalfenster, in dem man dann das Mail-Programm startet, über die beteiligten instantiierten Objekte informieren. Mit dieser Methode kann man das Mail-Programm bedienen, z.B. alle Mails als Junk-Mail markieren und dabei das Terminalfenster beobachtend die Objekthierarchie begreifen. Damit bekommt man einen groben Überblick. Es existiert bereits ein Mozilla-weites internes Logsystem, das aber von den Funktionen her die Anforderungen maßlos überstieg. Wenn man das Logsystem eingeschaltet hat, stellt man fest, dass die Log Datei schnell mehrere hundert Megabyte groß, und damit unüberschaubar wird. Aus diesem Grund wurde ein eigenes Logsystem implementiert, dass die wichtigen Informationen in eine Datei schreibt. Sehr wichtig und scheinbar in der Objekthierarchie ganz oben schienen die Funktionen zum Klassifizieren und zum Trainieren von Mails, `nsBayesianFilter::classifyMessage` und `nsBayesianFilter::SetMessageClassification` im Modul `nsBayesianFilter.cpp` zu sein.

Im Vorgehen kann z.B. nach allen `.cpp` Dateien, die die Zeichenfolge

`->classifyMessage` enthalten, gesucht werden. Dort findet man die Syntax und die Einsatzart der Methoden. So bietet es sich an, den Quellcode (*bottom up*) zu durchforsten, ebenfalls sinnvoll kann es sein, sich *top down* von den Menüfunktionen z.B. bei *Wende Junk Mail Klassifikator auf Ordner an* nach unten zu hangeln. Demzufolge sucht man die Ereignis-Funktionen, welche von einem Klick auf einen Menüpunkt ausgelöst werden, und damit die aufgerufenen instantiierten Objekte.

Der Stil des Mozilla Codes scheint trotz der vielen unterschiedlichen Programmierer einigermaßen einheitlich und verständlich zu sein. Das erste Teilziel war das automatisierte Trainieren der Ordner *Spam* und *Ham*, das Validieren der Ordner *ValidierungSpam* und *ValidierungHam* und die Ausgabe dieser Daten in Form einer Tabelle mit *Message-ID*, *Ist-Spam-Attribut*, *Klassifiziert-Als-Attribut* und *Spam-Wahrscheinlichkeit*. Die Richtigkeit der automatisierten Fassung wurde überprüft, indem der Vorgang einmal manuell durchgeführt wurde und einmal in der automatisierten Form. Beide Male war die Datei, die die Hashtables enthält, gleich groß. Da es sehr viele E-Mails waren und die Datei 11545945 Byte groß war, kann das als relevant für die Richtigkeit der Implementierung betrachtet werden.

Kapitel 5

Testumgebung

In diesem Kapitel wird sowohl die experimentelle Methodologie und die verwendete Testumgebung beschrieben. Dabei werden zunächst die benötigten Grundbegriffe erklärt. Danach wird noch auf die grundsätzliche Problematik bei der Evaluierung von Spam-Testszenarien sowie auf die verwendeten Daten und ein Standard Mail Format eingegangen, das von Mozilla genutzt wird.

5.1 Grundbegriffe und Testmethodologie

Ziel der Testreihen ist es, viele verschiedene Endbenutzerszenarien möglichst gut zu simulieren und zu evaluieren. Das heißt, die verwendeten Mail-Corpora sollen im Idealfall Mails enthalten, die sowohl inhaltlich als auch zahlenmäßig die Gesamtheit des existierenden E-Mail-Verkehrs abbilden. Generell wird in den Testreihen 30% der Spam/Ham-Mails als Validierungsmenge, der Rest als Trainingsmenge verwendet. Diese Zahlen haben sich als Erfahrungswert im Data-Mining bewährt. Man kann natürlich auf den selben Daten validieren, mit denen man trainiert hat. Damit kann man grob einschätzen, ob ein Klassifikator überhaupt die Komplexität hat, dem Problem gerecht zu werden. Ein Entscheidungsbaum, der nur aus zwei Entscheidungsknoten besteht, wäre wohl kaum in der Lage, die selben Spam-/Ham-E-Mails, mit denen trainiert wurde, richtig zu validieren. Der Sinn von getrennter Trainings- und Validierungsmenge ist das Abschätzen zukünftiger Generalisierungsleistungen.

5.2 Evaluierungsmaße

Bei der Klassifizierung von E-Mail als Spam oder Nicht-Spam ergeben sich naturgemäß die Klassen aus Tabelle 5.1, die in dieser Arbeit durchwegs in dieser Terminologie verwendet werden. Dabei bedeuten

- TP (true positives): Spam-Mails, die als Spam-Mails erkannt wurden

ist in Wirklichkeit	erkannt als Spam	Ham
Spam	TP	FN
Ham	FP	TN

Tabelle 5.1: Klassifizierung von Spam und Ham

- FP (false positives): Ham-Mails, die als Spam-Mails erkannt wurden
- FN (false negatives): Spam-Mails, die als Ham-Mails erkannt wurden
- TN (true negatives): Ham-Mails, die als Ham-Mails erkannt wurden

In manchen Arbeiten sind die Begriffe positiv und negativ vertauscht. In dieser Arbeit wird für TP auch der Begriff Spam-Erkennungsrate verwendet.

Zum Vergleich der Leistungsfähigkeit von Klassifikatoren eignen sich die ROC-Kurve und das Recall-Precision-Diagramm. Darüberhinaus werden die Übersichtsmaße AUC zur ROC-Kurve und 11-Point-Average-Precision zum Recall-Precision-Diagramm verwendet.

ROC-Kurve (Receiver Operation Characteristic): Spamerkennungsverfahren weisen zumeist einer zu klassifizierenden E-Mail eine Art Wahrscheinlichkeit oder Punkte zu, die dann repräsentieren, ob eine E-Mail eher als Ham oder Spam-E-Mail klassifiziert wurde. So gehört zu solch einem Bewertungssystem immer auch ein Schwellenwert, der aussagt, ob eine E-Mail nun letztendlich Spam ist oder nicht. Es ist sinnvoll, Klassifikatoren unabhängig von ihrem Schwellenwert zu vergleichen. Die ROC-Kurve biete eine gute grafische Veranschaulichung der Leistungen eines Spam-Klassifikators.

Dabei wird auf der X-Achse die False-Positive-Rate eingetragen, sprich die Anzahl aller Ham-Mails, die fälschlicherweise als Spam-Mails deklariert wurde, im Verhältnis zu der Anzahl aller Ham-Mails. Auf der Y-Achse liegt die True Positive Rate, sprich die Anzahl aller Spam-Mails, die auch als solche erkannt wurden. Die perfekte Funktion wäre eine Treppenfunktion, die von der linken unteren Ecke über die linke obere Ecke in die rechte obere Ecke führt. Gemeint ist, dass alle Spam-Mails erkannt werden, ohne dabei eine einzige False-Positive-Mail zu erhalten. In Abbildung 5.1 kann man ablesen, dass man etwas über 80% Spamerkennung bei 20% *false positives* erreicht. Das ist natürlich ein sehr schlechter Klassifikator im Anwendungsfeld Spamerkennung. Wenn man mehrere ROC-Kurven in ein Schaubild einzeichnet, gilt generell: Auf Abschnitten, in denen eine Kurve höher verläuft als eine andere, ist der Klassifikator, zu dem die höhere Kurve verläuft, der bessere. Als Zusammenfassungsmaß bietet sich die Fläche unter der Kurve (AUC Area under the curve) an, die natürlich zwischen 0 und 1 liegen muss.

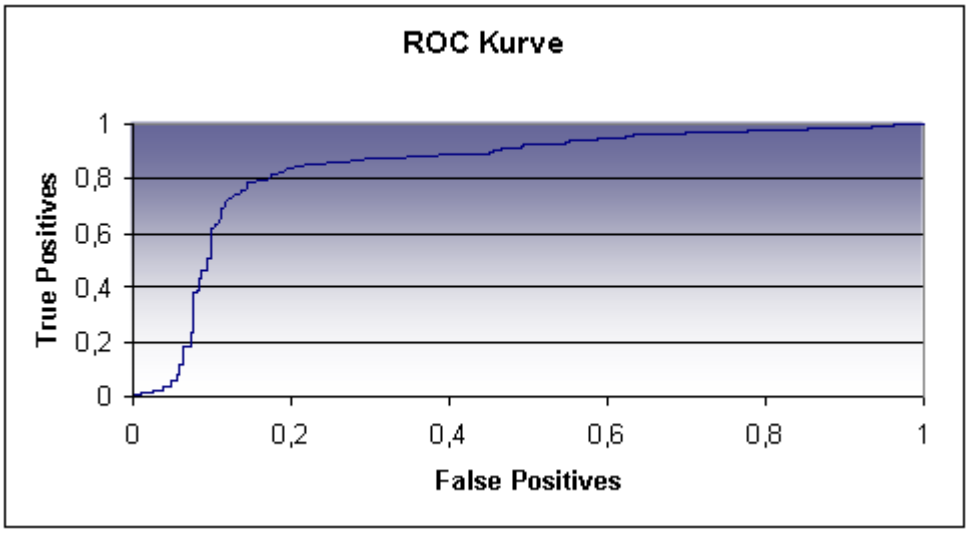


Abbildung 5.1: Erklärung ROC-Kurve

Recall/Precision Diagramm: Recall und Precision sind Kenngrößen aus dem Bereich des Information Retrieval. Auch sie eignen sich zur Bewertung von Spam-Klassifikatoren.

$$Recall = TruePositives/(TruePositives + FalseNegatives) \tag{5.1}$$

gibt darüber Auskunft, wie viel Prozent aller vorhandenen Spam-Mails auch als solche klassifiziert worden sind.

$$Precision = TruePositives/(TruePositives + FalsePositives) \tag{5.2}$$

besagt, wieviele aller als Spam erkannten Mails auch wirklich Spam sind.

Hier würde die perfekte Funktion von der linken oberen Ecke über die rechte obere Ecke und die rechte untere Ecke führen. Diesem Diagramm ließe sich beispielsweise entnehmen, dass wenn man mehr als 90% aller Spam-Mails als solche klassifizieren will, die Präzision rasch von etwa 90% auf bis 50% abnimmt. Man erreicht immer 100% Recall, wenn man einfach alles als Spam klassifiziert. Die Parallele beim Finden von relevanten Informationen generell ist die, dass man alle gesuchten Dokumente mit bestimmten Eigenschaften findet, indem man einfach alle Dokumente auswählt. Auch im Recall-/Precision-Diagramm lassen sich verschiedene Klassifikatoren vergleichen. Wer abschnittsweise bei gleichem Recall eine höhere Precision erreicht, ist der bessere Klassifikator.

Als Zusammenfassungsmaß für Recall und Precision bietet sich die 11-Point-Average-Precision an. Sie ist einfach der Durchschnitt der Precision Werte bei jeweiligen Recalls von 0%, 10%, 20%, ..., 100%.

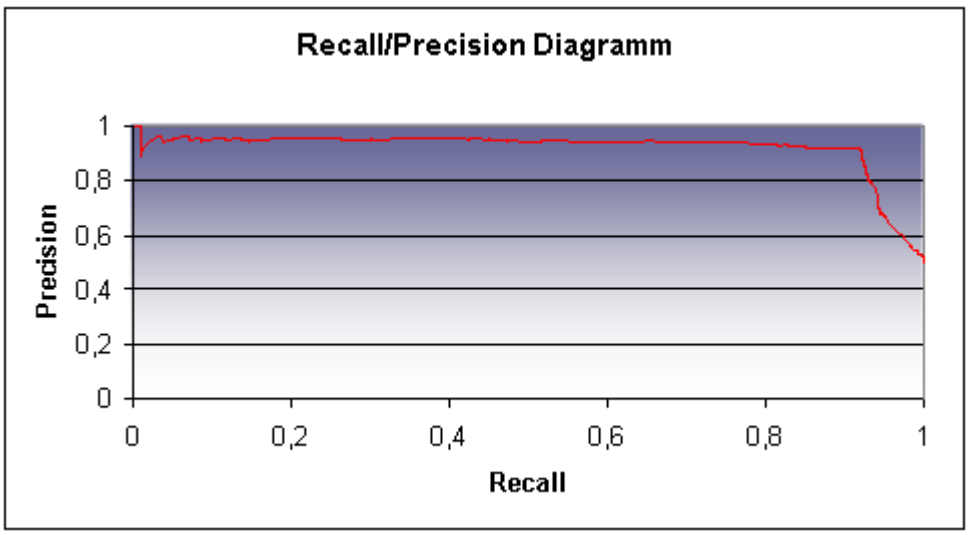


Abbildung 5.2: Erklärung Recall-Precision-Diagramm

Man erstellt ROC-Kurven und Recall-Precision-Diagramme, indem man die Liste der E-Mails mit realer Natur (Spam/Ham) und der Spam-Wahrscheinlichkeit nach der Wahrscheinlichkeit sortiert. Dann berechnet man die benötigten Kennzahlen (TP/FP/FN), in dem man über eine Schleife iteriert, so dass alle E-Mails bis zur Nr. X als Spam klassifiziert werden, alle darunter liegenden als Ham. Das entspricht einer Iteration über verschiedene Höhen des Schwellenwertes.

Wahrscheinlichkeitsdiagramm: Darüberhinaus enthalten die Testreihen ein Wahrscheinlichkeitsdiagramm. Dieses veranschaulicht die Datei in der die Wahrscheinlichkeiten aller klassifizierten Mails stehen. Kurz um gesagt, alle klassifizierten Mails werden nach Ihrer Wahrscheinlichkeit sortiert und in dem Diagramm aufgetragen. Somit ist unten erkennbar, dass die ersten 337 Mails in der sortierten Liste alle eine Wahrscheinlichkeit nahe bei Null besitzen.

5.3 Experimentelle Vorgehensweise

Grundsätzlich führte ich alle Testreihen in der veränderten Mozilla Suite durch. Dabei werden die beteiligten Spam-/Ham-Corpora in das lokale Mail-Verzeichnis kopiert und entsprechend der geplanten und dokumentierten Testreihen zufällig in Trainings- und Validierungsmengen aufgeteilt. Dabei hielt ich mich an die Faustregel „30% Validierungsmenge, 70% Trainingsmenge“. Die Verteilung Spam/Ham bewegt sich immer in einem Bereich, der der Verteilung in der Realität in den letzten Jahren entspricht, will heißen 30-60% Spam-Mails. Abweichungen werden in einzelnen Testreihen gesondert erläutert. Weiter startet man die eingebaute Testfunktion, die entweder mit Grahams *Plan for Spam* oder dem von mir implementierten Naive Bayes die Testreihen durchführt. Dabei werden zweispaltige

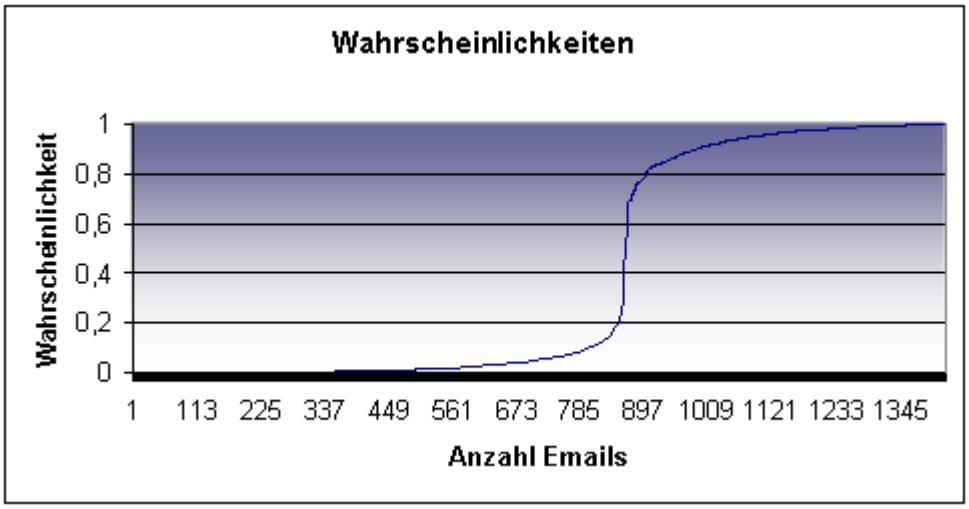


Abbildung 5.3: Erklärung Wahrscheinlichkeitsdiagramm

Dateien erzeugt, die in der linken Spalte der Datei pro E-Mail eine Null oder eine Eins für ihre wirkliche Natur Ham oder Spam enthalten und in der rechten Spalte die zugehörige Wahrscheinlichkeit des Klassifizierers. Diese Dateien werden dann von einem getrennten C Programm gelesen, um sie in Form von ROC-Kurven und Precision-Recall-Diagrammen auszuwerten. Alle Experimente wurden durchgeführt, indem zufällig die vor den Testreihen angegebenen Anzahlen von E-Mails aus dem Ursprungs-Corpus heraus kopiert und zum Testcorpus zusammengestellt wurden. Das zufällige Aufteilen der Mail Corpora ist sehr wichtig, da ansonsten die in Abschnitt 5.4 besprochenen Verzerrungen auftreten können, z.B. bei zeitlich sortierten Corpora.

5.4 Fehlerquellen bei der Evaluierung von Spam-Testszenarien

Die Evaluierung von Spam-Filtern und die Simulation realer Endbenutzerszenarien ist äußerst schwierig und nicht zu unterschätzen. Es zeigt sich, dass zwischen der rein wissenschaftlichen Evaluierung und Einschätzung von Spam-Lösungen und der Realität oft bedeutsame Unterschiede bestehen. Bei einer kleinen Umfrage, die ich per E-Mail im Freundeskreis durchführte, ergaben sich folgende, statistisch nicht unbedingt relevante, jedoch aus meiner Sicht in die richtige Richtung weisende Ergebnisse: Befragte Personen, die nicht Informatik studieren oder nicht sehr stark beruflich im Computer- Umfeld anzusiedeln sind, haben noch nie selbst irgendetwas an einem Spam-Filter eingestellt, weder Mails bewusst trainiert, noch den Spam-Filter überhaupt ein- oder ausgeschaltet. Das bedeutet, dass eben diese Nicht-Informatiker überhaupt nur Zugang zu den Spam-Filtern

der Mail-Provider oder mit etwas Glück ein MS Outlook-Plugin installiert haben. Bei den Filtern der Mail-Provider ergibt sich das Bild, nach dem schon etwa 1/3 der befragten Personen Probleme mit *false positives* hatten. Es scheint tatsächlich so zu sein, dass selbst das einfache Anklicken des Mülleimer-Symbols für Spam im Posteingang für die meisten Computernutzer schon zuviel ist, weil sie schon froh sind, wenn das E-Mail-Programm an sich funktioniert.

In diesem Zusammenhang kam ich etwas in Berührung mit Benutzbarkeitsstudien, die auf viele Bereiche des Marketings im Internet ein völlig neues Licht werfen, siehe dazu [44] oder [68].

Bei den Informatikern ist schon eine rege Benutzung von Spam-Filtern zu verzeichnen, selbst diese haben allerdings dann und wann Probleme mit *false positives* und der Unverständlichkeit der Filter. Den größten Anteil an allen Spam-Filtern scheinen die Filter der Mail-Provider und die Plugins für MS Outlook zu haben (schon allein auf Grund der Marktdominanz von MS Outlook). Je höher die Ausbildungsstufe im Bereich der Informatik, desto eher scheinen die Benutzer Open-Source-Lösungen wie Mozilla Thunderbird zu verwenden.

Jetzt aber zu den Tücken des Data Minings, die oft als Fehler bei der Evaluierung von Spam-Filtern auftreten. Grundsätzlich ist die Einschätzung der meisten Studien zu optimistisch. Die Zahlen zwischen den Papers und der Realität liegen oft deutlich auseinander. Das schwierige ist, gute Mail-Corpora zu beschaffen. Es gibt zwar viele Freiwillige, aber wie immer bei der Beschaffung von Daten für Data-Mining-Zwecke gibt es einige Tücken, die die Daten nur als schwache Approximation der Realität erscheinen lassen. So sind die Personen, die E-Mails freiwillig sammeln, um Spam-Corpora zu erstellen, meist keinesfalls Repräsentanten für die Allgemeinheit, sondern Informatiker oder Wissenschaftler, die gegen den Kommerz und die Verunreinigung des E-Mail Dienstes angehen wollen. Speziell, wenn gegen die eigenen Ham-Mails der Personen, die Spam Studien erstellen, abgeglichen wird, ergeben sich Probleme, weil deren Ham-Mail nicht repräsentativ ist, und weil, wie oben schon angedeutet, in der Wissenschaft öfter Open Source Software eingesetzt wird, die zu anderen Mail Köpfen führen kann. Auch die Themen in den Mails sind völlig andere, sehr wissenschaftlich und gespickt von Fachausdrücken. Auch E-Mail-Programme, die HTML E-Mails verfassen, dürften hier seltener anzutreffen sein.

Es ist schwierig bis unmöglich wegen der privaten Natur von Ham-Mails an gute Ham-Corpora heranzukommen, deshalb werden meistens Spam und Ham-Corpora aus völlig verschiedenen Quellen verwendet. So kann ein Filter z.B. lernen, dass der Name oder Vokabeln, die mit persönlichen Interessen zusammenhängen, immer Ham-Anzeichen sind. Die Spam-Mails können aus einer völlig anderen Zeitspanne kommen, so kann je nach Datumsformat z.B. das Wort „Januar“ ein besseres Anzeichen für Spam sein als alles andere zusammen. Weiter können die Ham-Corpora bereits durch einen Filter gelaufen sein, weshalb die härtesten Ham-Mails bereits aussortiert sind.

Nicht zu unterschätzen sind die psychologischen und soziologischen Faktoren.

Jemand, der ein Paper zu einer neuen Spam-Filter-Idee schreibt, möchte natürlich gute Zahlen und den Ruhm einheimsen. So scheinen sehr viele Papers zu optimistisch zu sein. Man liest selten über Filter, die überhaupt nicht funktionieren.

Dass die bisher unzureichende Evaluierung von Spam-Filtern ein aktuelles Thema ist, zeigt sich daran, dass in jüngster Zeit Ansätze gestartet wurden, dieses Problem in den Griff zu bekommen. Jüngst wurde von der Text Retrieval Conference (TREC) [64] unter Leitung von Gordon Cormack ein Projekt namens SPAM Track [71] ins Leben gerufen, das der Vergleichbarkeit und Evaluation von Spam-Filtern dient. Dabei handelt es sich um ein Toolkit, welches für die gängigen Betriebssysteme eine Software bereitstellt, die standardisierte Evaluationsmechanismen zur Verfügung stellt. Diese Software kann mit jeder Art Spam-Filter verknüpft werden, die die entsprechenden Schnittstellen zur Verfügung stellt. Dabei werden die benutzten Test-Corpora in chronologischer und einheitlicher Reihenfolge präsentiert, anstatt einfach Spam- und Ham-Corpora komplett zu trainieren und dann zu validieren. Nur so ist eine realistische Validierung möglich, da ja in Wirklichkeit ein Spam-Filter, der am Anfang untrainiert ist, schon während des Trainings validiert wird, sprich die E-Mails werden in chronologischer Reihenfolge empfangen und es ist ja nicht nur interessant, was nach tausenden von E-Mails validiert wird, sondern auch die Lernkurve am Anfang, die ja nun einmal in der Realität bei einem untrainierten Filter auftritt.

Wichtig ist den Mitarbeitern dieses Projektes die einheitliche Validierung und Wiederholbarkeit aller Tests, sowie eine große Anzahl von Teilnehmern, die sowohl viele Testcorpora als auch eine Menge verschiedener Filter ausprobieren. Die Ergebnisse sollen dann zentral gesammelt und der Öffentlichkeit zur Verfügung gestellt werden. Das soll im Oktober 2005 passieren, die Präsentation erfolgt dann auf der TREC Konferenz vom 15.-18. November 2005. Auf der Seite von Gordon Cormack finden sich weitere Links und Informationen zu Spam Track. [72]

5.5 Verwendete Mail-Corpora und deren Beschaffung

Zunächst einige Bemerkungen über die Beschaffung, Verwendung und Problematik von E-Mail-Corpora. Grundsätzlich finden sich mittlerweile zahlreiche E-Mail-Corpora, vor allem zum Zwecke der Spam-Filter-Evaluierung, im Internet. Diese werden teilweise von nicht kommerziell ausgerichteten Organisationen, teilweise von Unternehmen und auch privat gepflegt. Ein Hauptproblem ist, dass Ham-E-Mail privat ist, schließlich möchte niemand seine private E-Mail in einer öffentlichen E-Mail-Sammlung wiederfinden. Damit ist klar, dass jede Form von Ham-Corpora maximal eine Approximierung der Realität sein kann. Bei Spam-Corpora ist das unproblematisch und sie stehen in großer Zahl zur Verfügung. Aber auch hier ergeben sich Probleme wie der Zeitraum, aus dem die E-Mails

stammen. Es stellen sich diverse Fragen, die bei der Verwendung zu beachten sind.

- Wie sind die E-Mails sortiert?

  Es zeigt sich, dass die Corpora oft zeitlich oder nach Beschaffungsquellen sortiert vorliegen.

- Aus welchem Zeitraum stammen die E-Mails?

  Dies ist vor allem auf Grund der hohen Geschwindigkeit der Spam-Evolution von grundsätzlichem Interesse. Das die meisten Spam-Corpora aus der Zeit von 2000-2005 stammen, weist auf die steigende Bedeutung der Problematik hin.

- Wie ändern sich die Dynamiken in der Spam Evolution, lässt sich das anhand von unterschiedlich datierten Mail-Corpora belegen?

  Das ist schwierig, da sich in unterschiedlichen Zeiträumen auch die Anzahl der vorhandenen Spam-Corpora deutlich unterscheidet. Wirklich einheitliche Spam-Archive mit vergleichbaren Einträgen gibt es erst seit 2002, somit lässt sich die Spam-Evolution auch erst seit 2002 vernünftig untersuchen.

- Wie teile ich die Spam-Corpora in Trainings- und Testmenge auf?

  Es hat sich bewährt, 30% als Validierungsmenge und 70% als Trainingsmenge zu verwenden.

- Lasse ich sie dabei in der ursprünglichen Sortierung oder mische ich zufällig?

  In jedem Fall sollten die Mails zufällig gemischt werden, da fast alle Corpora in einer Weise sortiert sind, die die Ergebnisse verzerren würden.

- Wirkt sich die Weiterleitung von E-Mails auf die Klassifizierung aus, weil ja viele Mail Corpora auf diese Weise gesammelt werden?

  Ja, auch das kann sich stark auswirken, weshalb auch viele Corpora von solchen Einträgen bereinigt wurden.

- Ergeben sich dadurch Verfremdungen der Realität, da bestimmte Absender immer Beispiele für Spam und bestimmte andere Absender immer Beispiele für Ham schicken in einer unüberwachten Sammelstelle?

  Meiner Meinung nach ja, aber das wurde hier nicht untersucht.

Wenn man nur deutsche Ham-Mails und nur englische Spam-Mails hat, wird das Ergebnis einer Untersuchung sich bis zum völlig unrealistischen Ergebnis verzerren, da irgendwelche deutschen Wörter sofort ein hundertprozentiges Signal für eine Ham-Mail sind. Wenn man z.B. in einem Unternehmen Spam-E-Mails sammelt, ist ganz und gar nicht klar, ob dort ein realistischer Querschnitt von

Spam-E-Mails ankommt. Ähnlich verhält es sich mit dem geographischen Standort, an dem die Mail-Corpora erstellt werden (siehe dazu auch die Anmerkungen zur Sprache der E-Mails in 5.1 „eigene Forschungsanregungen“). Viele Untersuchungen basieren auf unzureichend großen E-Mail-Corpora und sind damit nicht oder kaum relevant.

5.6 Übersicht existierender, freizugänglicher und verwendeter eigener E-Mail-Corpora

Ich versuche im folgenden einen Überblick über verfügbare E-Mail-Corpora im Internet zu geben, mit Angabe der Quelle und Daten zur Größe und zum Format der Dateien, zusätzlich Mutmaßungen über die Qualität der Corpora. Teile dieser Sammlung werden dann später zur Evaluierung des Mozilla-Spam-Filters und des Naive-Bayes-Klassifikators verwendet.

- Spamassassin Public Mail Corpus [45]: Mit 514 Treffern bei www.google.de unter „spamassassin public corpus“ ein sehr verbreiteter Spam-Corpus, der in vielen Studien über Spamfilter Verwendung fand. In diesem Corpus wurden alle E-Mailköpfe gepflegt und bei Veränderung durch Weiterleitung wieder in den Originalzustand gebracht. Er besteht aus 6047 E-Mails, mit einer Spam-Rate von 31%. Die Mails liegen nicht im MBOX-Format vor. Hilfe bei der Konvertierung ins MBOX-Format findet sich unter [61]. Keine genaue Zeitangabe, gesammelt etwa ab dem Jahr 2000.

- Annexia Spam-Corpus [46][47]: Diese Sammlung besteht aus 25126 Spam-Mails und ist mit 33 Treffern unter www.google.de „annexia Spam-Corpus“ recht verbreitet. Älteres Archiv, E-Mails etwa ab 1997 bis 2003 gesammelt.

- Xpert Ham-Corpus[48]: Hier findet man neben Ling Spam, Spamassassin und Annexia auch den Ham-Corpus Xpert. Xpert ist ein Ham-Corpus aus einer Mailingliste, die sich mit Xfree86 beschäftigt. Zeitraum unbekannt.

- Ling Spam-Corpus[49]: 2412 E-Mails aus einer Mailing Liste mit dem Thema Linguistik, 481 Spam-Mails, Zeitraum unbekannt.

- Spamarchive Corpus[50]: Quantitativ wertvolle Sammlung von Spam E-Mail, mit zwei verschiedenen Varianten, automatisch gesammelte Spam-Mails und per Anhang eingeschickte Spam-Mails. Bei den automatisch gesammelten E-Mails wurden Tools an Freiwillige ausgegeben, die Spam-Mails automatisch an Spamarchive weitergeben. In dieser Sammlung befinden sich 222.000 E-Mails. Man beachte die vielen Duplikate.

- Paul Wouters Spam Archiv[51]: Hier finden sich neben einem Spam Archiv auch weitere nützliche statistische Informationen und Daten zur Spam Entwicklung. Das Spam Archiv enthält Spam sauber getrennt nach den Jahren 1997-2004, allerdings nimmt die Corpus Größe auch exponentiell zu. Somit enthält der erste 541 Spams, der letzte 95405 Spams.

- Bruce Guenters Archiv[52]: 1998-2005. Zeitlich sortiert 1998-2001 nach Jahren, 2002-2005 nach Monaten.

- Bloodgate Spam Archiv[53]: 1998-2005, Spam-Mail Archiv mit sehr interessanten und aktuellen Statistiken.

- Japanisches Spam Archiv[57]: keine weiteren Informationen, eventuell interessant für die Sprachproblematik.

- Privater Ham-Corpus des Autors: 2870 private E-Mails aus dem Leben eines Informatik-Studenten, darunter etwa 500 mit universitärem Bezug, 600 Mails aus einer (erwünschten) Mailing Liste mit dem Thema Datenbanken, 350 geschäftliche E-Mails aus der Börsen- und Bankenwelt und der Rest rein private E-Mails. Zeitraum 1999-2005.

- Geordnete Spamarchive Corpora: Aus den Spam-Archive-Daten habe ich zufällig fünf Spam-Corpora zu je tausend Mails zusammengebaut, jeweils aus verschiedenen Zeitabschnitten, die sich auf Quartale beziehen. Es existieren also

  01/03 erstes Quartal aus dem Jahr 2003

  03/03 drittes Quartal aus dem Jahr 2003

  01/04 erstes Quartal aus dem Jahr 2004

  03/04 drittes Quartal aus dem Jahr 2004

  01/05 erstes Quartal aus dem Jahr 2005.

Für die durchgeführten Testreihen wurden die Corpora Spamassassin, Annexia, Xpert, Spamarchive und der private Ham-Corpus des Autors verwendet. Die öffentlich zugänglichen Corpora wurden auf Grund der Kriterien von Reinheit und Vollständigkeit der Daten ausgewählt.

5.7 Das Mail Format „MBOX“

Zum Speichern von E-Mails auf der Festplatte hat sich in der Linux-Welt hauptsächlich das MBOX-Format durchgesetzt. Fast alle E-Mail-Programme unterstützen das MBOX-Format, von Kommandozeilenwerkzeugen wie PINE bis hin

Name	Zeitraum	Anz.SPAM	Anz.HAM	Verweis
Spamassassin Public Mail Corpus	2000–2005	1875	4172	[45]
Annexia Spam-Corpus	1997–2003	25126	0	[46][47]
Xpert Ham-Corpus		0	11000	[48]
Ling Spam-Corpus		2412	0	[49]
Spamarchive Corpus		222000	0	[50]
Paul Wouters Spam Archiv	1997–2004	100000	0	[51]
Bruce Guenters Archiv	1998–2005			[52]
Bloodgate Spam Archiv	1998–2005			[53]
Japanisches Spam Archiv				[57]
Privater Ham-Corpus	1999–2005	0	2870	

Tabelle 5.2: Übersicht Spam-Corpora

zu ausgereiften E-Mail-Programmen wie Thunderbird, dem Nachfolger des E-Mail-Teils aus der Mozilla Suite. Grundsätzlich werden im MBOX-Format einfach alle E-Mails in einer Datei hintereinander gespeichert. Man erkennt eine beginnende neue Nachricht an zwei Leerzeilen und einem „From“, auf welches ein Leerzeichen und dann eine Leerzeile folgt. Einzige Ausnahme ist die erste Nachricht, vor deren „From“ nichts weiter steht. Nachfolgend sind die beiden Möglichkeiten als reguläre Ausdrücke dargestellt.

```
Erste Nachricht
<From *\n>

Jede weitere Nachricht
<\n\nFrom *\n>
```

Um Schwierigkeiten zu verhindern, falls einer dieser regulären Ausdrücke in dem Inhalt einer E-Mail auftritt, wird „Character Stuffing“ verwendet. Damit ist gemeint, dass bei einem „From“ am Anfang einer Zeile in einer E-Mail ein „¿“ davorgesetzt wird, bevor die E-Mail in der MBOX-Datei gespeichert wird. Allgemein bezeichnet „Character Stuffing“ das Einfügen von Steuerzeichen, um Fehlinterpretationen des lesenden Parsers zu vermeiden (oft im Bereich von Netzwerktechnologie, „Character Stuffing“, „Bit Stuffing“).

In Mozilla-Mail werden alle E-Mails eines Ordners in einer MBOX-Datei abgelegt. Diese Datei hat keine Dateinamenserweiterung. Zusätzlich bildet Mozilla eine Index-Datei, die den selben Namen wie die MBOX-Datei trägt, selbst aber die Dateinamenserweiterung *.msf erhält. Die Größe der MBOX-Dateien wird nur noch von Betriebssystemvorgaben beschränkt, was aber bei den heutigen erlaubten Dateigrößen kein Problem mehr darstellen dürfte.

Kapitel 6

Auswertung der Testreihen

Es werden die Experimente besprochen, die der Evaluierung und dem Vergleich von Grahams *Plan for Spam* und einem Naive-Bayes-Klassifikator im Bereich der Spam-Klassifizierung dienen. Darüberhinaus wird der Vergleich der beiden Algorithmen im Kontext der Mail-Sortierung betrachtet.

6.1 Betrachtung und Konzept der Testreihen

Die Testreihen dieses Kapitels gliedern sich in drei Abschnitte. Private, deutschsprachige Ham-Mail (Testreihe 1-2), Analyse anhand von bekannten Ham- und Spam-Corpora (Testreihe 3-6) und Einsatz zur Mail-Sortierung (Testreihe 7-8). Dabei wurde auf verschiedenen Daten und mit unterschiedlichen Voraussetzungen getestet, um eine realistische Basis für den Vergleich von Grahams *Plan for Spam* mit einem Naive-Bayes-Klassifikator zu schaffen.

In Abschnitt 6.2 (*Private, deutschsprachige Ham-Mail*) wurden Tests mit den privaten hauptsächlich in deutscher Sprache verfassten E-Mails des Autors durchgeführt. Dabei steht die Zielgruppe derer im Vordergrund, die hauptsächlich deutschsprachige E-Mails empfangen.

In Abschnitt 6.3 (*Analyse anhand von bekannten Ham- und Spam-Corpora*) werden verbreitete Ham- und Spam-Corpora verwendet, um eine Vergleichbarkeit mit anderen Studien zu ermöglichen. Testreihen 3 bis 5 arbeiten ausschließlich auf englischen E-Mails, dabei werden die zwei von Spamassassin zur Verfügung gestellten Ham-Corpora „easy ham“ und „hard ham“ miteinbezogen. Für Testreihe 6 wurden Annexia und Xpert, zwei andere bekannte englisch-sprachige Corpora, hinzugezogen, die eine große Anzahl von E-Mails enthalten. Öffentliche Spam-Corpora haben den Vorteil, dass sie normalerweise aus verschiedenen Quellen gesammelt werden und deshalb eine allgemeingültigere Analyse ermöglichen als privat gesammelte Spam-Mails. Öffentliche Corpora sind Voraussetzung für eine Vergleichbarkeit von Studien. Bei öffentlichen Ham-Corpora ist allerdings nur eine Approximation der Realität möglich, weshalb für realistische Tests auf jeden

Fall privat gesammelte Ham-Mails vorzuziehen sind, hierbei geht aber leider die Vergleichbarkeit mit anderen Studien verloren. Auswertungen, die sich auf privat gesammelte Ham-Mails stützen, sind aber eventuell recht stark auf die Zielgruppe zugeschnitten, die dem Sammler der E-Mails ähnelt.

Abschnitt 6.4 (*Einsatz zur Mail-Sortierung*) widmet sich der Anwendung im Bereich der E-Mail-Sortierung. E-Mail-Sortierung bedeutet, ankommende E-Mails im Postfach nach verschiedenen Kategorien zu ordnen. Das kann bei Email-Postfächern mit hohem Empfangsvolumen sehr sinnvoll sein, um z.B. die Kategorien mit den höheren Prioritäten als erstes zu bearbeiten. Mehr noch als im privaten Bereich könnte solch eine Klassifizierung z.B. in einem E-Mail-Postfach sinnvoll sein, das sämtliche Beschwerden oder Kontaktformulare einer Firma empfängt.

Das Klassifizierungsproblem des Mail-Sortierens hat eine andere semantische Natur als das Spam-Filtern. Die Anforderung ist hierbei in mehr als zwei Kategorien zu ordnen. Da man aber jedes n-Kategorisierungsproblem als ein binäres Kategorisierungsproblem (siehe Abschnitt 3.1) darstellen kann, sind die Experimente relevant. Testreihe 8, welche intuitiv das schwerste Separierungsproblem darstellt, könnte ein reales Problem zur Mail-Sortierung sein. Es wird versucht, zwei Sorten von Ham-Mails aus dem Corpus des Autors zu klassifizieren. Dabei handelt es sich einerseits um eher offiziell geartete E-Mails aus dem universitären Umfeld, andererseits um E-Mails guter Freunde. Der Unterschied dieser beiden Klassen ist „offiziell“ versus „umgangssprachlich“ und semantischer Natur. Es sollte herausgefunden werden, ob der Vergleich von Grahams *Plan for Spam* und einem Naive-Bayes-Klassifikator unter diesen Anforderungen anders ausfällt.

Die Evaluationen der Testreihen werden in Form von ROC-Kurven, Recall-Precision-Diagrammen, Wahrscheinlichkeitsdiagrammen und als Tabellen dargestellt. Die Wahrscheinlichkeitsdiagramme beziehen sich immer nur auf den Naive-Bayes-Klassifikator, da Grahams *Plan for Spam* nur Wahrscheinlichkeiten direkt bei Null oder Eins ausgibt und damit das Diagramm immer nur eine gleich geartete Treppenfunktion beinhalten würde. Trotzdem sagt das Wahrscheinlichkeitsdiagramm auch viel für den Vergleich der beiden Algorithmen aus, da man mit dessen Hilfe den Schwierigkeitsgrad der Separierbarkeit der Daten einschätzen kann. Eine Übersicht aller Testreihen findet sich in Tabelle 6.1.

6.2 Private, deutschsprachige Ham-Mail

Diese Testreihe soll anhand der Ham-Mails des Autors aufzeigen, was *Plan for Spam* und der Naive-Bayes-Klassifikator zur Spam-Erkennung leisten und ob sie zum Einsatz als Spam-Filter geeignet sind. Dieser Test ist eine realistische Approximation der meisten E-Mail-Szenarien, die viele deutschsprachige Ham-Mails und vor allem englischsprachige Spam-Mails erhalten. In den Test geht die Lernkurve des Klassifizierers nicht mit ein, das heißt, es wird nicht chronologisch

NR	Spam-Corpus	Ham-Corpus	#Spam Training	#Ham Training	#Spam Val.	#Ham Val.
1	Spamarchive Spam	Privat Ham	750	750	250	250
2	Spamassassin Spam	Privat Ham	1329	2009	570	861
3	Spamassassin Spam	Spamassassin Easy Ham	1329	1786	570	765
4	Spamassassin Spam	Spamassassin Hard Ham	1329	175	570	75*
5	Spamassassin Spam	Spamassassin Hard+Easy Ham	1329	1961	570	75*
6	Annexia Spam	Xpert Ham	7000	7000	3000	3000
7	Privat Ham	Xpert Ham	2000	2000	800	800
8	Freunde Ham	Uni-Kollegen Ham	601	670	258	287

Tabelle 6.1: Übersichtsdarstellung aller Testreihen
* Validierung nur aus Hard-Ham

jede eingegangene E-Mail trainiert, sondern es werden alle Trainings-E-Mails auf einmal trainiert. Somit stehen die Ergebnisse eher für die Raten, die man nach einigem Training des Spam-Filters erhält.

Testreihe 1 (Privater Ham-Corpus des Autors versus Spamarchive Corpora): In dieser Testreihe (siehe Tabelle 6.2) wird ein Test durchgeführt, in dem tausend Ham-Mails aus dem privaten Mail-Corpus des Autors mit tausend Spam-Mails aus den jeweils quartalsweise geordneten Corpora des Spamarchive kombiniert werden. Die Zeitangaben in der Tabelle sind somit als Quartale des jeweiligen Jahres zu verstehen. Es werden jeweils 750 Trainingsmails und 250 Validierungsmails verwendet, in Spam und Ham. Dabei werden der in Mozilla integrierte Spamfilter nach Grahams *Plan for Spam*, bei einem von Graham festgelegten Schwellenwert 0.9, mit dem vom Autor implementierten Naive-Bayes-Filter, Schwelle 0.92, verglichen(siehe Abschnitt 3.5). Dabei bedeuten in Tabelle 6.2

- NB: Naive Bayes
- PFS: Plan for Spam
- TP: True Positive Rate
- TN: True Negative Rate
- Opt. SW: Der Schwellenwert, bei dem das Ergebnis optimal wird.
- Opt. TP/TN: Die durch Opt. SW erreichten Ergebnisse.

Der heuristisch ermittelte Schwellenwert von 0.92 für den Naive Bayes erweist sich als unbrauchbar, was an der Sparte bester Schwellenwert Naive Bayes deutlich zu erkennen ist. Somit entsteht der Eindruck, dass sich der Naive-Bayes-Algorithmus wesentlich empfindlicher gegenüber der Veränderung des Schwellenwertes zeigt. Die Bandbreite des optimalen Schwellenwertes beträgt beim Naive-Bayes-Klassifikator 0.87 bis 0.95. Grahams *Plan for Spam* zeigt sich sehr stabil mit durchwegs keinen *false positives*, wobei mit Naive-Bayes die *false positives* selbst bei perfekter Anpassung des Schwellenwertes im Quartal 3/2003 mit 17 von 250 also 6.8 % in einem inakzeptablen Bereich liegen. Beim *Plan for Spam* ändert sich in drei von fünf Quartalen nicht einmal etwas, wenn man den Schwellenwert im Bereich 0.01 bis 0.99 verändert. Testreihe 1 zeigt somit eine deutliche Überlegenheit von Grahams *Plan for Spam*.

Testreihe 2 (Privater Ham-Corpus des Autors versus Spamassassin Corpora):

Das Filterverhalten ist wie erwartet sehr gut. Die privaten E-Mails des Autors lassen sich leicht vom nur aus englischen Spam-Mails bestehenden Spamassassin Corpus trennen. Leider tritt bei den Auswertungen des *Plan for Spam* in

Quartal	1/2003	3/2003	1/2004	3/2004	1/2005
TP (NB)	0,940	0,840	0,872	0,504	0,960
TP (PFS)	0,920	0,980	0,980	0,980	0,984
TN (NB)	0,996	0,928	1,000	1,000	1,000
TN (PFS)	1,000	1,000	1,000	1,000	1,000
Opt. SW (NB)	0,950	0,870	0,990	0,940	0,950
Opt. SW (PFS)	0,01	0,01-0,13	0,01-0,99	0,01-0,99	0,01-0,99
Opt. TP (NB)	0,984	0,984	1,000	0,836	1,000
Opt. TP (PFS)	0,972	0,984	0,980	0,980	0,984
Opt. TN (NB)	0,996	0,932	1,000	1,000	1,000
Opt. TN (PFS)	1,000	1,000	1,000	1,000	1,000

Tabelle 6.2: Klassifikationsraten Testreihe 1

vielen Testreihen das Problem auf, dass aufgrund der nur wenigen verschiedenen Klassifizierungs-Endwahrscheinlichkeiten der E-Mails auch nur wenige Datenpunkte für die ROC-Kurve und das Recall-Precision-Diagramm vorhanden sind (siehe Ausführungen in Abschnitt 6.5). Somit ist es in manchen Testreihen zusätzlich zu den Diagrammen sinnvoll, die Ursprungsdaten zu besprechen, sprich die Liste mit den klassifizierten E-Mails aus der jeweiligen Testreihe und den zugehörigen Wahrscheinlichkeiten. Je schwieriger die Separierbarkeit einer Testreihe, desto mehr verschiedene Wahrscheinlichkeiten liefert der *Plan for Spam*.

Beim *Plan for Spam* werden bei 0.34% *false positives* 98.2% der Spam-Mails ausgefiltert. Beim Naive-Bayes-Klassifikator ist das Ergebnis fast exakt gleich (Abbildung 6.1).

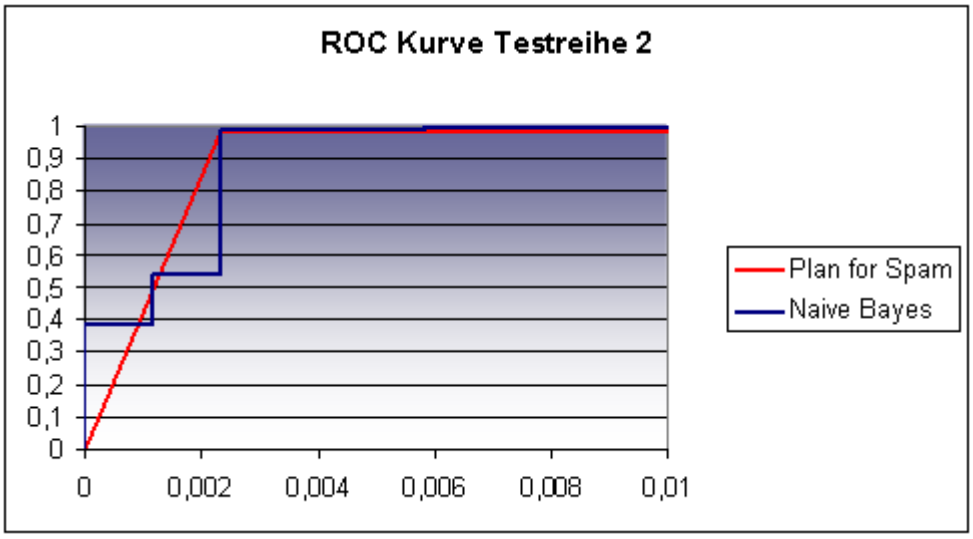


Abbildung 6.1: ROC-Kurve Testreihe 2

Beim Betrachten der Ursprungsdaten fällt auf, dass sich die gewünschte maximal akzeptable Anzahl *false positives* beim Naive-Bayes-Klassifikator feiner regeln lässt, mit sehr vergleichbaren Ergebnissen. Abbildung 6.2 zeigt das selbe Bild, und zwar etwa gleiche Performance der beiden Algorithmen bei deutlich weniger Datenpunkten des *Plan for Spam*.

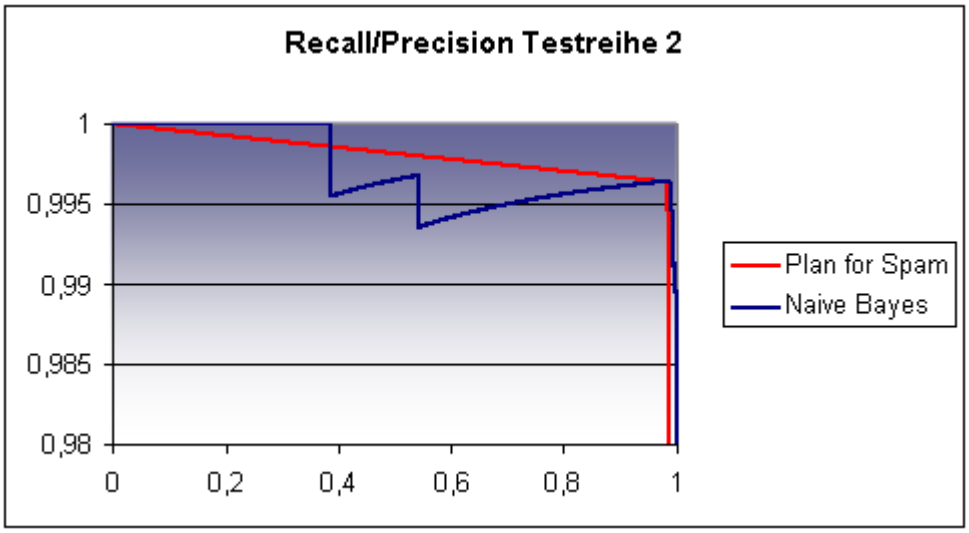


Abbildung 6.2: Recall-Precision-Diagramm Testreihe 2

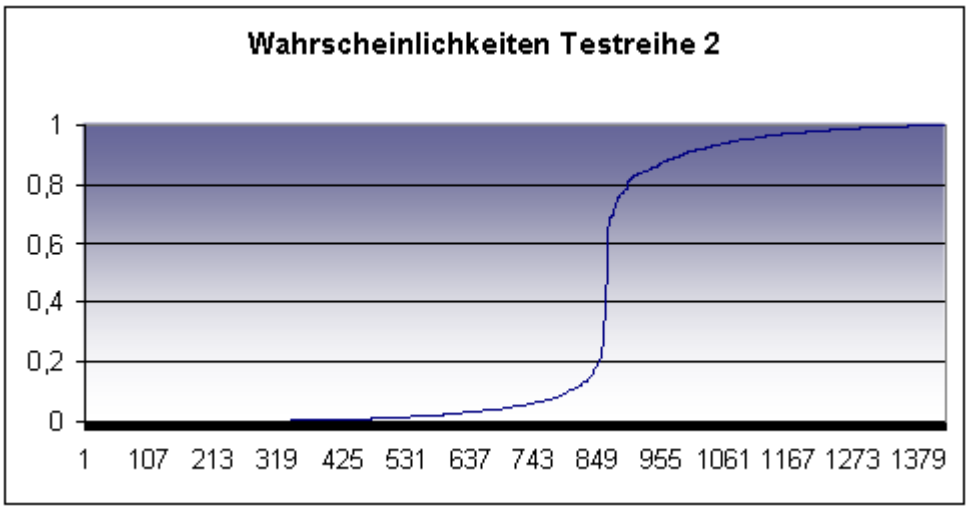


Abbildung 6.3: Wahrscheinlichkeitsdiagramm Testreihe 2

Das Wahrscheinlichkeitsdiagramm (Abbildung 6.3) zeigt den Übergangsbereich zwischen den Klassen Ham und Spam. Dieser liegt in der Liste der nach

Wahrscheinlichkeit sortierten E-Mails 6/10 vom linken Rand entfernt. Das entspricht genau der Verteilung der Klassen in der Validierungsmenge. Es fällt auf, dass der Klassenübergang sehr steil verläuft, was auf gute Separierbarkeit hinweist.

Zusammenfassend lässt sich feststellen, dass sich der *Plan for Spam* in beiden Testreihen sehr gut eignet, um die Ham-Mail des Autors von aktueller Spam-Mail zu trennen. Dabei werden Spam-Erkennungsraten von etwa 98% und False-Positive-Raten von 0-0.3% erreicht. Der Naive-Bayes-Algorithmus zeigt sich in Testreihe 1 wesentlich empfindlicher gegenüber verschiedenen Instanzen. Auf der Instanz des Corpus in Testreihe 2 arbeitet er vergleichbar gut.

6.3 Analyse anhand von bekannten Ham- und Spam-Corpora

In diesem Abschnitt soll anhand von öffentlich zugänglichen Corpora die Leistungsfähigkeit des Naive-Bayes-Klassifikators mit Grahams *Plan for Spam* verglichen werden. Dabei wird auf die Corpora Spamassassin, Xpert und Annexia zurückgegriffen. Testreihe 4 soll die Folge von einem veränderten Spam/Ham-Verhältnis zeigen, welches völlig anders ist als das Verhältnis in der Validierung. Das kann eine Rolle spielen, wenn Spam-Mails schwallartig auftreten und die A-Priori-Wahrscheinlichkeiten überhaupt nicht mehr stimmen.

Testreihe 3 (Spamassassin Easy-Ham-Corpus versus Spamassassin-Corpora):

Die Ursprungsdaten der ROC-Kurve zeigen, dass für *Plan for Spam* zwischen 0.2% und 1.3% FP ausreichend viele Datenpunkte vorliegen, um einen genauen Vergleich durchzuführen. *Plan for Spam* schneidet im Bereich zwischen 0.2% und 0.55% FP deutlich besser ab als Naive-Bayes, zwischen 0.55% und 1.3% nur geringfügig schlechter (Abbildung 6.4).

Die Präzision geht im hohen Recall-Bereich beim *Plan for Spam* etwas früher zurück (Abbildung 6.5).

Das Wahrscheinlichkeitsdiagramm (Abbildung 6.6) für den Naive-Bayes fördert Interessantes zu Tage: Die Trennlinie zwischen den Klassen befindet sich zwar an der erwarteten Stelle (57% nach der Klassenverteilung in der Evaluationsmenge), die Wahrscheinlichkeiten fallen aber viel weniger klar aus als in Testreihe 2. Gerade weil gegen den selben Ham-Corpus separiert wird, ist das ein deutlicher Hinweis darauf, dass es viel schwerer ist Ham-/Spam-Mails einer Sprache zu trennen als wenn jeweils jede Klasse eine andere Sprache besitzt, wie in Testreihe 2. Somit kann vermutlich davon ausgegangen werden, dass das Spam-Problem bei häufigen englischen Ham-Mails bzw. in englischsprachigen Ländern noch weitaus schwieriger zu lösen ist, wenn man davon ausgeht, dass die meisten Spam-Mails in englischer Sprache verfasst sind. Dafür kann vermutet werden, dass in Deutsch-

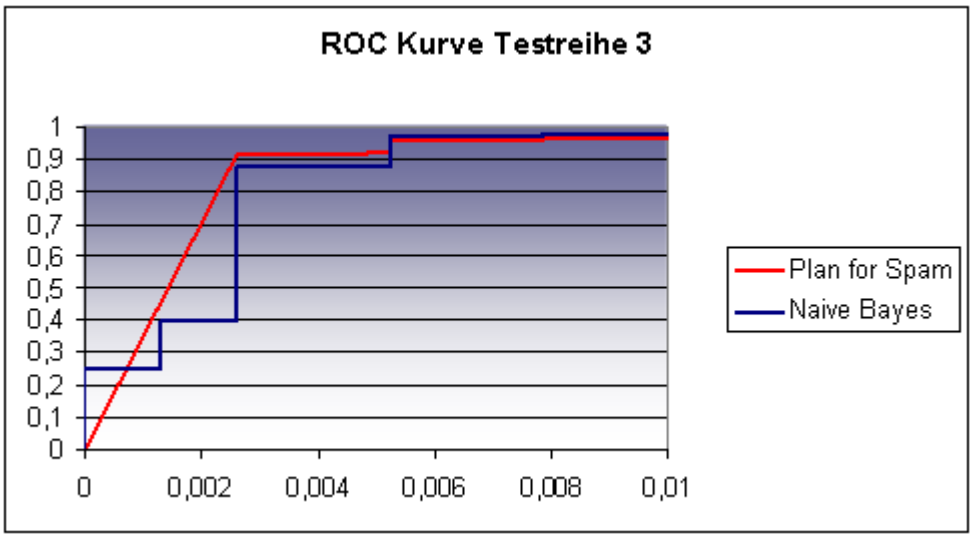


Abbildung 6.4: ROC-Kurve Testreihe 3

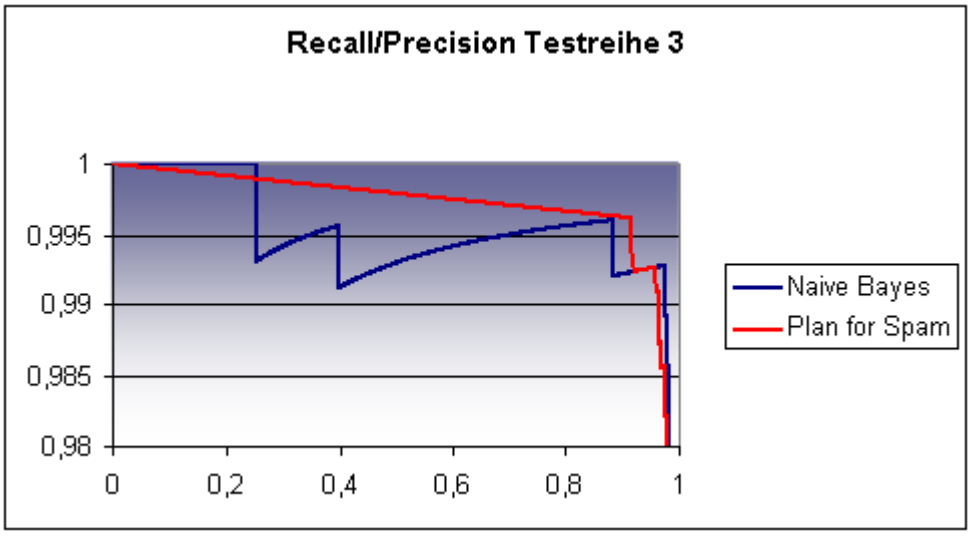


Abbildung 6.5: Recall-Precision-Diagramm Testreihe 3

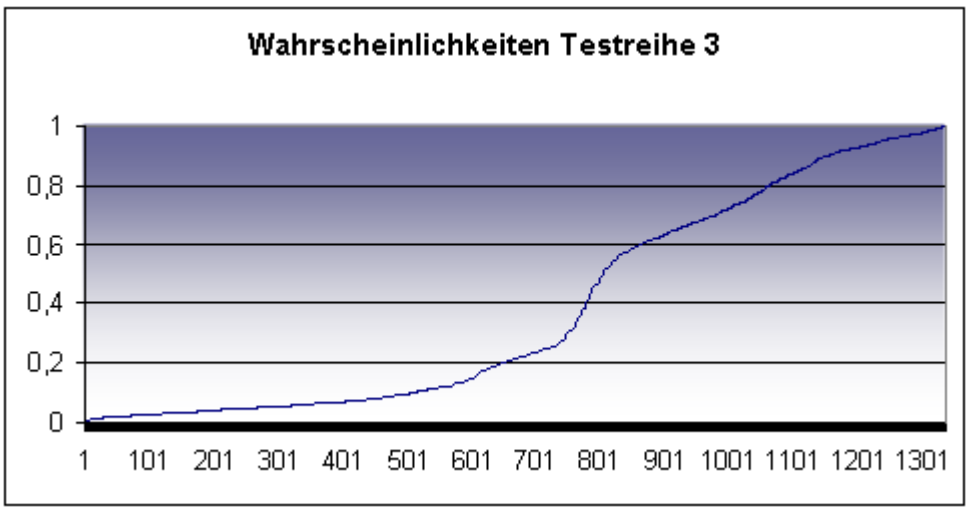


Abbildung 6.6: Wahrscheinlichkeitsdiagramm Testreihe 3

land englische E-Mails viel häufiger als *false positives* klassifiziert werden.

Testreihe 4 (Spamassassin Hard-Ham-Corpus versus Spamassassin-Corpora):

Die erwartete Klassenverteilung für das Wahrscheinlichkeitsdiagramm des Naive-Bayes für Testreihe 4 (Abbildung 6.7) ist 11% zu 89%.

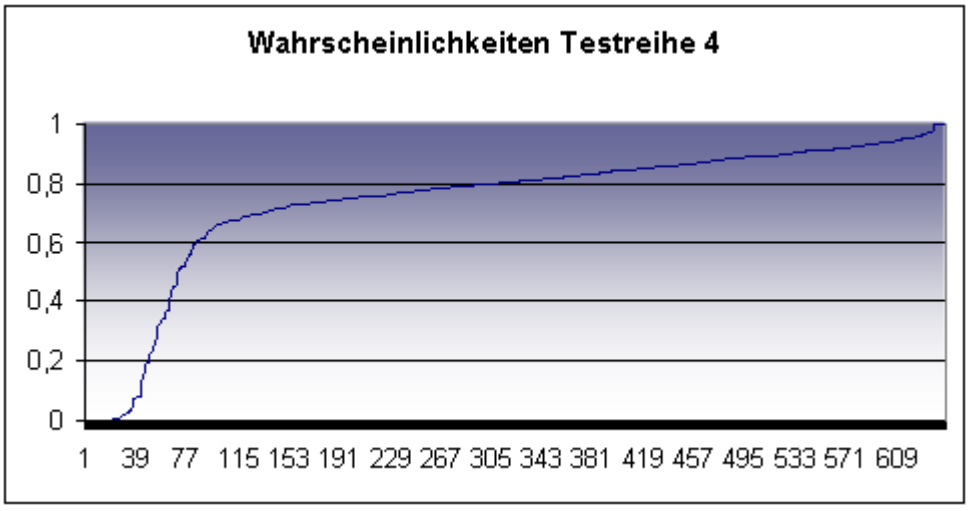


Abbildung 6.7: Wahrscheinlichkeitsdiagramm Testreihe 4

Somit sollten die ersten fnfundsiebzig E-Mails des Diagramms als Ham klassifiziert werden. Vergleicht man aber die Wahrscheinlichkeiten der ersten fnfundsiebzig E-Mails mit den Ham-Wahrscheinlichkeiten am linken Rand des Diagramm aus Testreihe 3 (Abbildung 6.6), sieht man sofort, dass die Ham-Mails in Testreihe

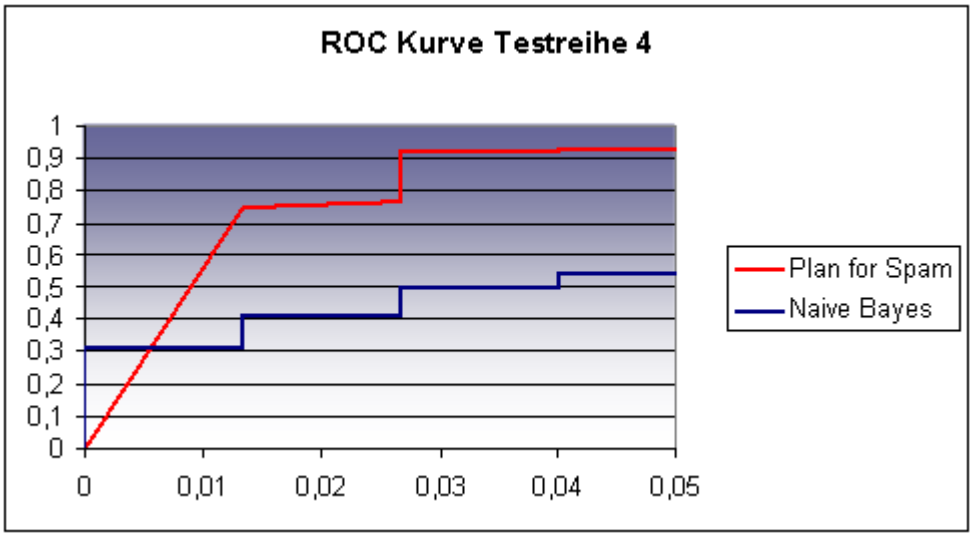


Abbildung 6.8: ROC-Kurve Testreihe 4

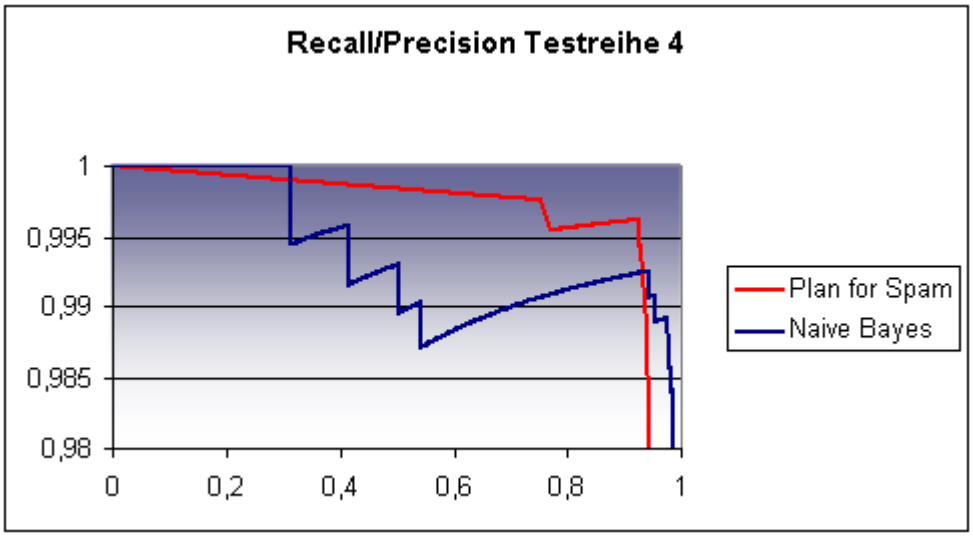


Abbildung 6.9: Recall-Precision-Diagramm Testreihe 4

4 viel höhere Spam-Wahrscheinlichkeiten besitzen (Hard-Ham-Corpus).

Sowohl Abbildung 6.8 als auch Abbildung 6.9 zeigen eine deutliche Überlegenheit des *Plan for Spam*. Das der Naive-Bayes den *Plan for Spam* trotzdem im höchsten Recall-Bereich des Recall-Precision-Diagramms übertrifft scheint daran zu liegen, dass der Schwellenwert des Naive Bayes, wie bereits mehrfach festgestellt, feiner justierbar ist. Die *False-Positive-Rate* liegt mit 3% bei 91% Spam-Erkennung deutlich höher als in Testreihe 3, was auf die schwierige Erkennung der E-Mails aus Hard-Ham hinweist.

Testreihe 5 (Spamassassin Hard+Easy Ham-Corpus versus Spamassassin-Corpora):

Das Wahrscheinlichkeitsdiagramm des Naive Bayes (Abbildung 6.10) sieht ähnlich aus wie das von Testreihe 4 (Abbildung 6.7), nur ist es flacher. Das liegt an der höheren Anzahl Ham-Mails im Training.

Auch hier zeigen ROC-Kurve und Recall-Precision-Diagramm (Abbildung 6.11 bzw. Abbildung 6.12) eine deutliche Überlegenheit des *Plan for Spam*. Die Spam-Erkennungsrate liegt mit 85% bei 4% *false positives* deutlich niedriger als in Testreihe 4. Dies entsteht nur durch eine grössere Trainingsmenge von Ham-Mails. Die Validierungsmenge blieb in Testreihe 5 dieselbe wie in Testreihe 4. Dies weist auf die Wichtigkeit der A-priori-Wahrscheinlichkeiten hin.

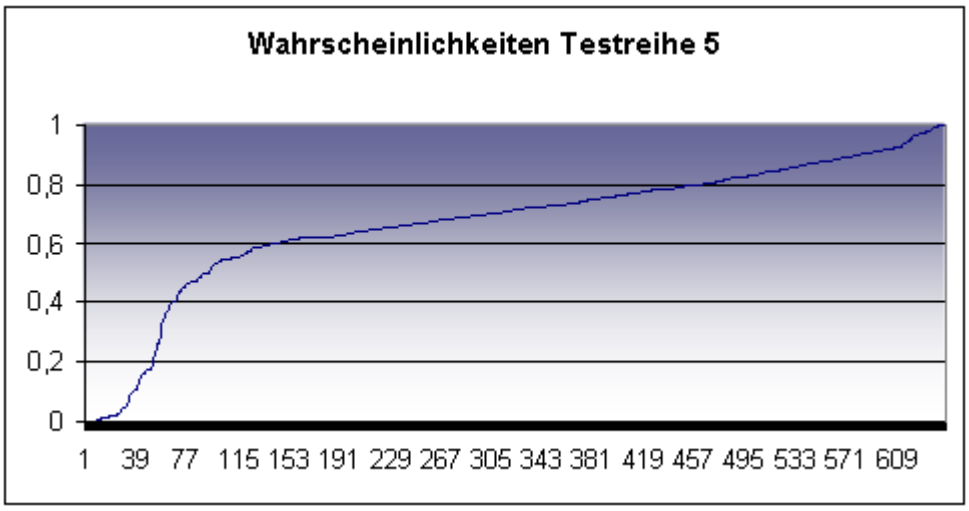


Abbildung 6.10: Wahrscheinlichkeitsdiagramm Testreihe 5

Testreihe 6 (Xpert Ham-Corpus versus Annexia Corpora):

ROC-Kurve und Recall-Precision-Diagramm (Abbildung 6.13 bzw. Abbildung 6.14) zeigen eine ähnliche Effizienz der beiden Algorithmen. In den unteren Recall-Bereichen von Abbildung 6.14 ist der *Plan for Spam* nicht wirklich überlegen. Dieser Anschein wird nur erweckt weil hier keine Datenpunkte vorhanden sind.

Das Wahrscheinlichkeitsdiagramm des Naive-Bayes (Abbildung 6.15) zeigt durch die steile Klassentrennung eine sehr gute Separierbarkeit.

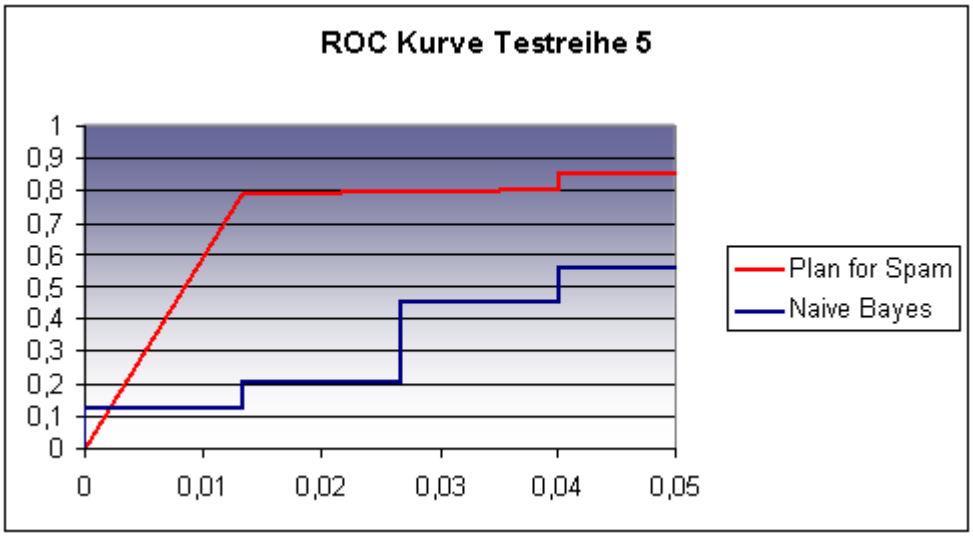


Abbildung 6.11: ROC-Kurve Testreihe 5

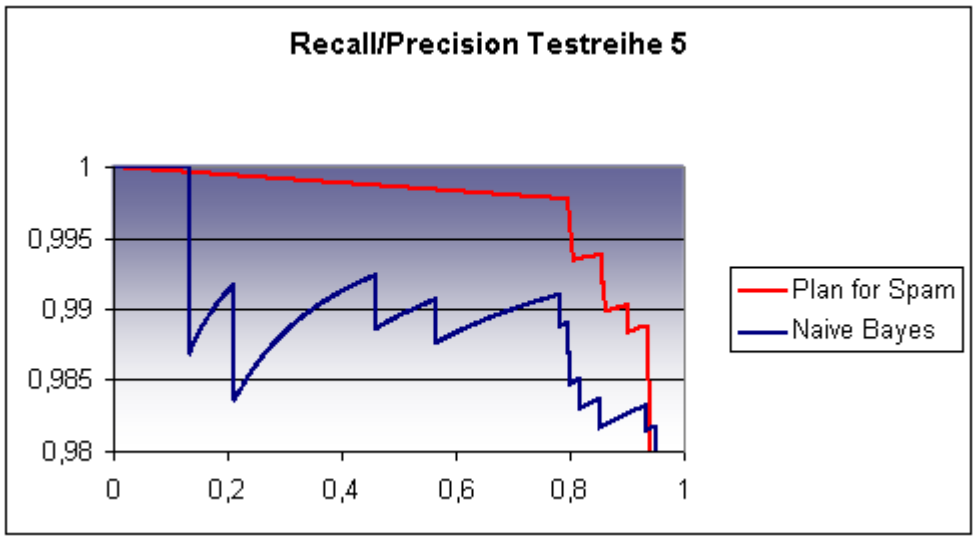


Abbildung 6.12: Recall-Precision-Diagramm Testreihe 5

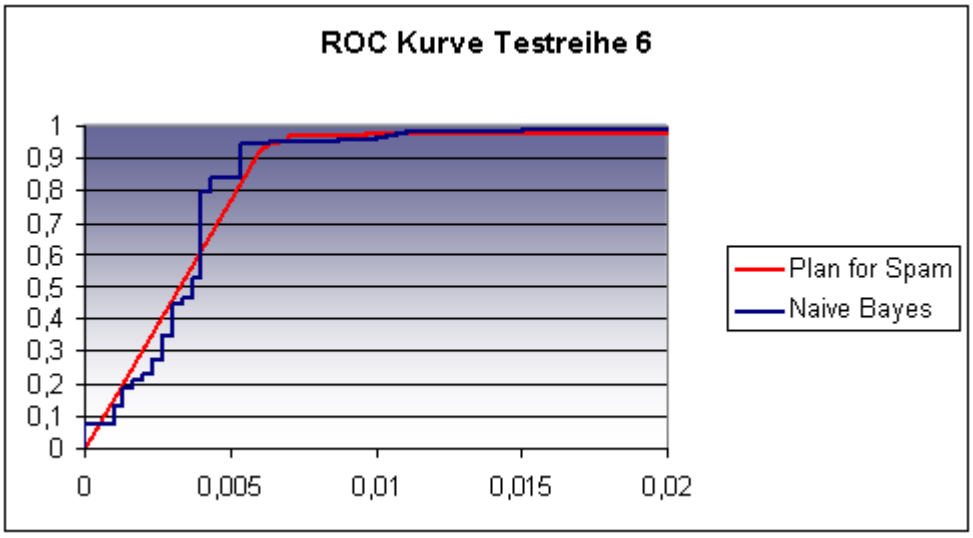


Abbildung 6.13: ROC-Kurve Testreihe 6

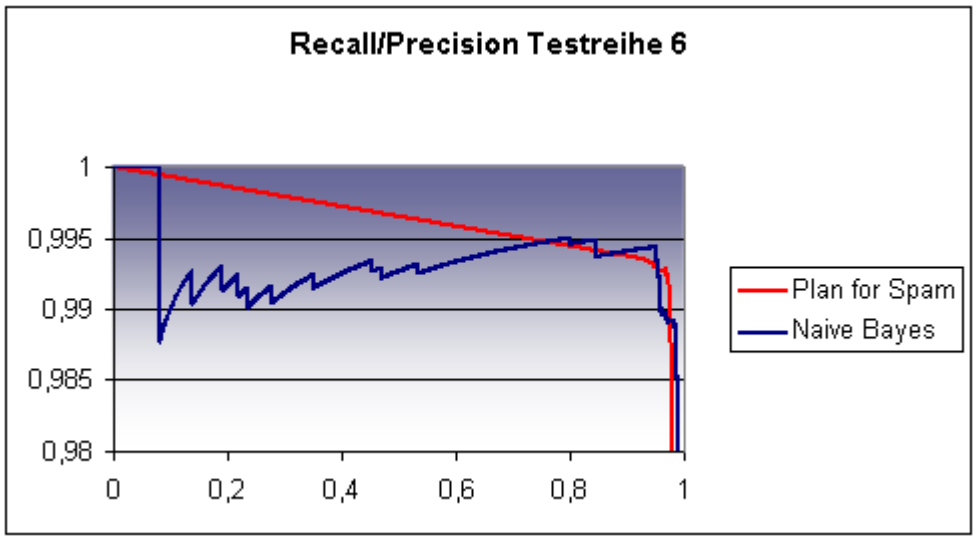


Abbildung 6.14: Recall-Precision-Diagramm Testreihe 6

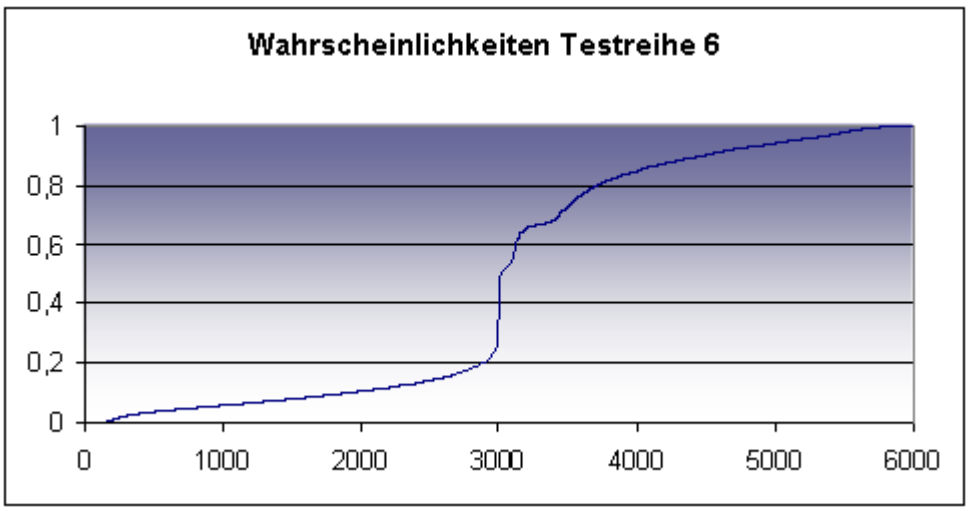


Abbildung 6.15: Wahrscheinlichkeitsdiagramm Testreihe 6

Testreihen 3 und 6 können mit den Testreihen 1 und 2 verglichen werden. Dabei zeigt sich, dass beide Algorithmen auf den Experimenten mit öffentlich zugänglichen Corpora deutlich schlechtere Ergebnisse liefern als auf den privaten Corpora des Autors. Testreihen 3 und 6 liefern Spam-Erkennungsraten von 97% bei einer *False-Positive-Rate* von 0.6% - 1% gegen 98% Spam-Erkennungsrate und 0%-0.3% False-Positive-Rate bei den privaten Corpora. Der Hard-Ham-Corpus aus Testreihe 4 schlägt sich in deutlich höheren *False-Positive-Raten* nieder. Testreihe 5 zeigt, dass nur durch eine Erhöhung der Anzahl der Trainings-Ham-Mails eine deutliche Veränderung in der Validierung erreicht werden kann.

6.4 Einsatz zur Mail-Sortierung

Die Mail-Sortierung könnte gerade im Bereich von Support- oder Beschwerde-E-Mail-Adressen von grossen Firmen eine erhebliche Bedeutung erlangen. So könnte man die Arbeit der Mitarbeiter, die sich um die Abwicklung dieser E-Mails kümmern, erheblich vereinfachen, wenn die E-Mails schon in ausreichendem Maße vorsortiert wären, da meist verschiedene Mitarbeiter für verschiedene Teilmengen der eingegangenen E-Mails zuständig sind. Die Firmen versuchen dies jedoch schon dadurch zu erreichen, dass man beim Abschicken der E-Mails die inhaltliche Kategorie in einem Webformular angibt. Eventuell könnte dann die E-Mail-Sortierung aber auch dazu dienen, diese Kategorisierung zu überprüfen. Im Falle von E-Mail-Sortierung sind die Fehlerkosten wieder anders zu betrachten als im Falle des Spam-Mail-Filterns. Ein Mitarbeiter, der für einen bestimmten Bereich des Supports verantwortlich ist und falsch einsortierte E-Mails erhält, kann diese einfach zu einem seiner Kollegen verschieben. Das ist nicht weiter schlimm und

setzt nicht den Vorteil herab, der gegenüber keiner Sortierung entsteht. Somit ist ein realistischer Anwendungszweck auf jeden Fall gegeben. Je nachdem, welche Richtigkeiten man mit der Mail-Sortierung erreicht, entscheidet sich, ob diese Methode auch für den privaten oder kleineren kommerziellen/universitären Rahmen geeignet sein könnte. Testreihe 7 soll die Möglichkeit des Sortierens nach verschiedenen Sprachen aufzeigen.

Testreihe 7 (Xpert Ham-Corpus versus Privat Ham-Corpus):

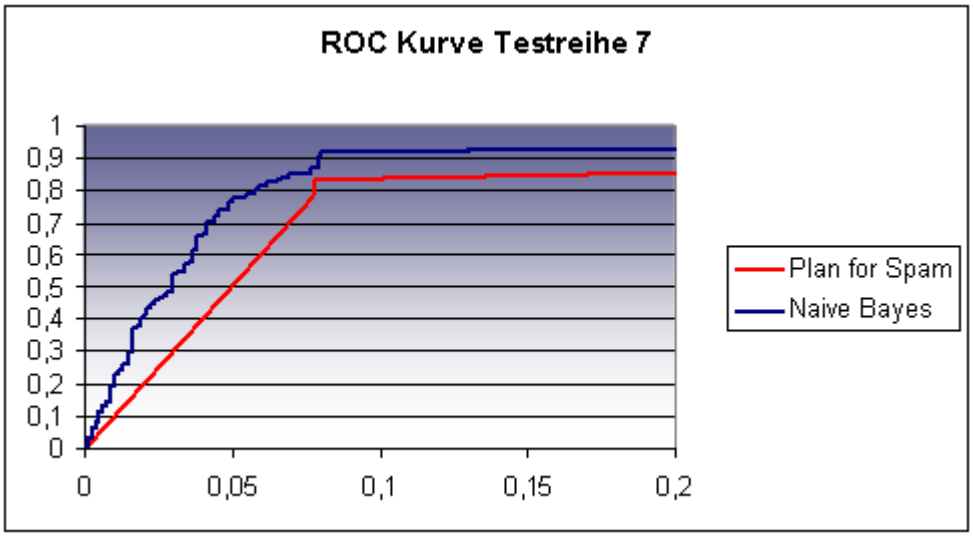


Abbildung 6.16: ROC-Kurve Testreihe 7

ROC-Kurve und Recall-Precision-Diagramm (Abbildung 6.16 bzw. Abbildung 6.17) zeigen eine Überlegenheit des Naive-Bayes-Klassifizierers. Das Wahrscheinlichkeitsdiagramm des Naive-Bayes (Abbildung 6.18) zeigt eine sehr gute Separierbarkeit und nur in Testreihe 2 wird eine ähnlich saubere Wahrscheinlichkeitsverteilung generiert. Nur in diesen beiden Testreihen sind jeweils die zwei verschiedenen Klassen in zwei verschiedenen Sprachen verfasst. Das weist nochmals ausdrücklich auf die Sprachenproblematik und deren Wichtigkeit hin. Es werden Erkennungsraten von 90% bei etwa 8% *false positives* erreicht. Das wäre für den Einsatz des Sortierens nach Sprachen zumindest ein brauchbarer Wert. Die starken Schwankungen im Recall-Precision-Diagramm, die auch in den anderen Testreihen zu beobachten sind, rühren daher, dass die Grafiken nicht interpoliert, sondern punktweise gezeichnet sind. Somit kann speziell am Anfang des Diagramms der Precision-Wert sehr stark zurückgehen, wenn eine Reihe von Ham-Mails hintereinander fälschlicherweise als Spam-Mails klassifiziert wird.

Testreihe 8 (Private Mails Freunde versus Uni Kollegen): Testreihe 8 zeigt in Abbildung 6.19 im Bereich von 0 bis 10% *false positives* eine Überlegenheit des *Plan for Spam*. Will man aber höhere Erkennungsraten erzielen und duldet dabei auch mehr *false positives*, eignet sich der Naive-Bayes-Algorithmus

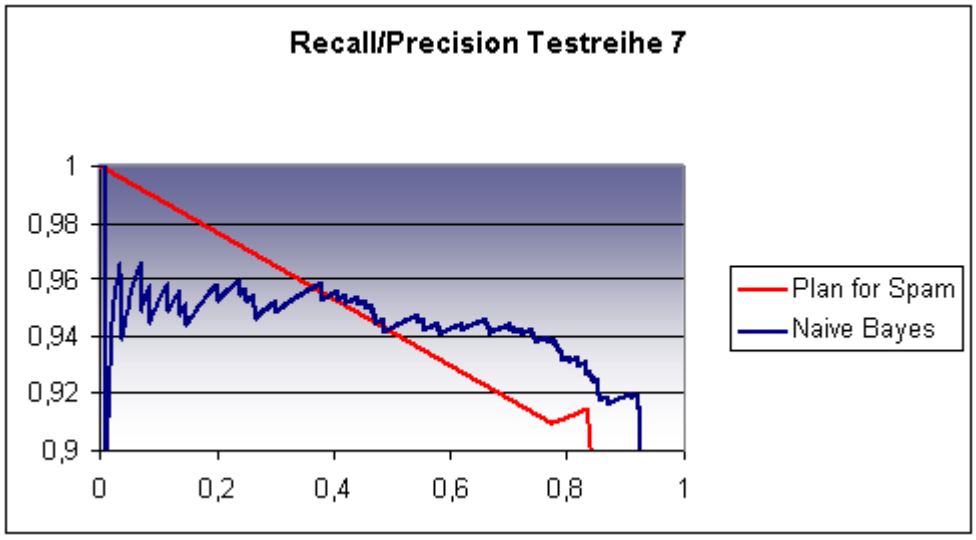


Abbildung 6.17: Recall-Precision-Diagramm Testreihe 7

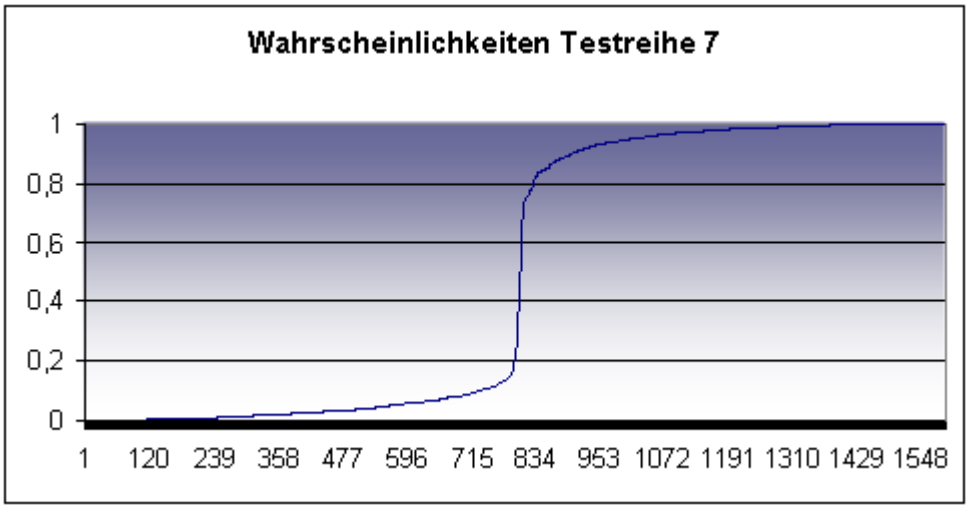


Abbildung 6.18: Wahrscheinlichkeitsdiagramm Testreihe 7

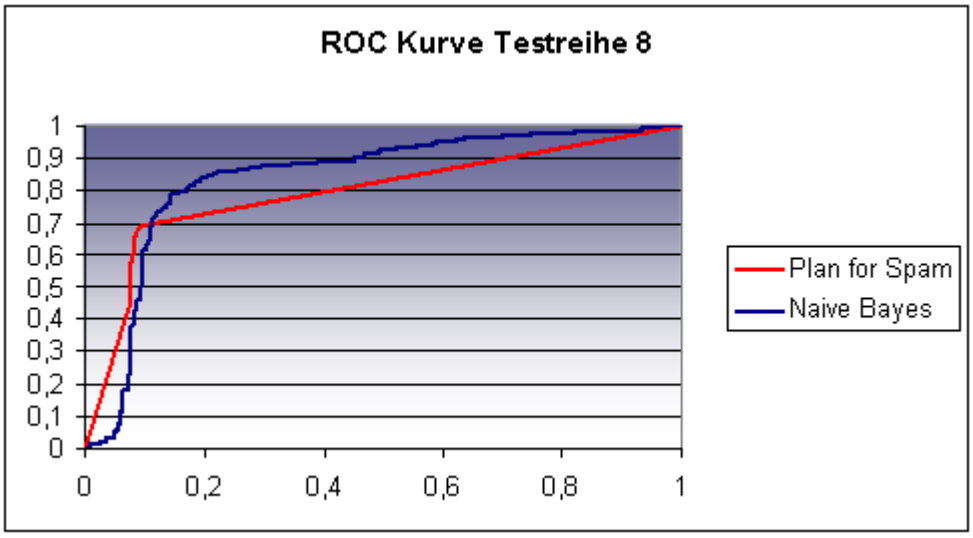


Abbildung 6.19: ROC-Kurve Testreihe 8

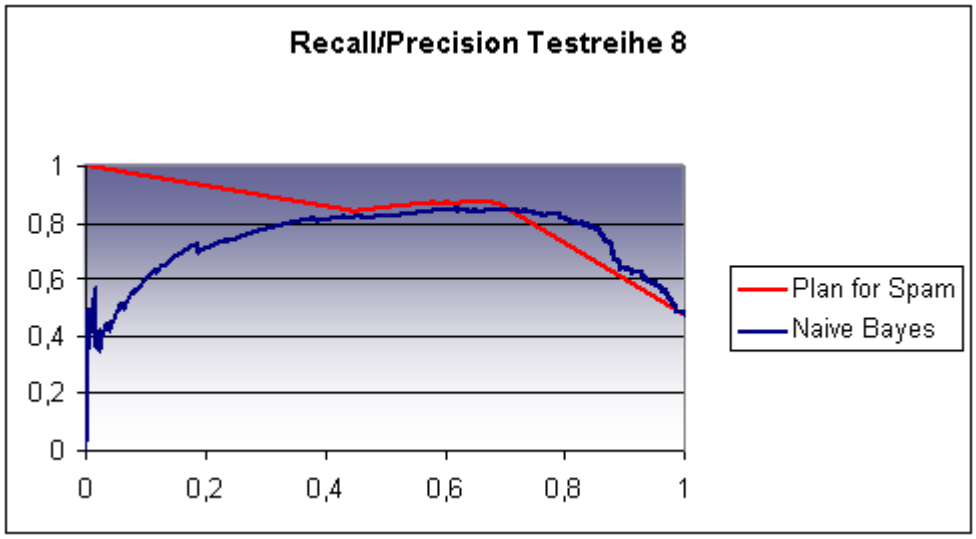


Abbildung 6.20: Recall-Precision-Diagramm Testreihe 8

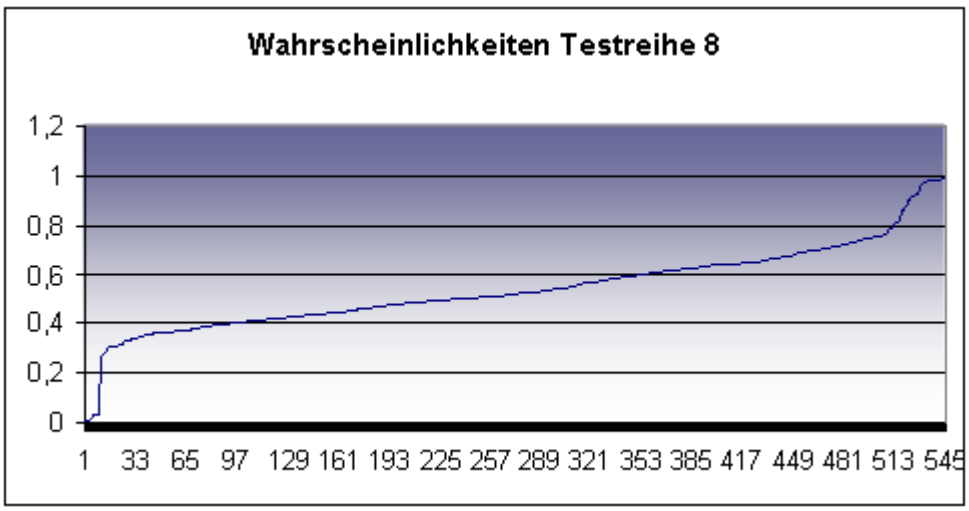


Abbildung 6.21: Wahrscheinlichkeitsdiagramm Testreihe 8

besser. Abbildung 6.20 zeigt eine Überlegenheit des *Plan for Spam*. Aussagekräftige Datenpunkte des *Plan for Spam* befinden sich dabei im Recall-Precision-Diagramm im mittleren Recall-Bereich. Das Wahrscheinlichkeitsdiagramm des Naive-Bayes (Abbildung 6.21) zeigt eine sehr flache Kurve. Das bedeutet, dass die Unterschiede der verschiedenen Klassen sich kaum in den Wahrscheinlichkeiten abbilden, die Klassen scheinen sehr schwer separierbar zu sein.

Die Erkennungsraten zur Mail-Sortierung scheinen in einem Bereich zu liegen, der für manche Anwendungszwecke durchaus effizient sein kann, für den privaten Bereich aber in dieser Form nur wenig Sinn macht. Die Sortierung mit den beiden Algorithmen *Plan for Spam* und Naive-Bayes ist etwa vergleichbar, einmal mehr mit der Eigenschaft, dass sich der Tradeoff *true positives* und *false positives* mit dem Naive-Bayes feiner regeln lässt. Weiter lässt sich feststellen, dass je nach Sorte des Textseparierungsproblems der Naive-Bayes-Algorithmus deutlich besser sein kann als Grahams *Plan for Spam*.

6.5 Zusammenfassung der Test-Ergebnisse

Insgesamt kann man feststellen, dass die Testreihen betreffend dem Vergleich der beiden Algorithmen kein völlig einheitliches Bild vermitteln. Es scheint (siehe Tabelle 6.3) aber so zu sein, dass der *Plan for Spam* bei schwierig zu separierenden Testreihen bessere Ergebnisse liefert als der Naive-Bayes-Klassifikator. Eine wichtige Erkenntnis ist, dass ROC-Kurven und Recall-Precision-Diagramme für den Naive-Bayes-Klassifikator ein sehr adäquates Mittel zur Evaluation sind, sich für den *Plan for Spam* aber nur bedingt eignen. Es ist auf jeden Fall nötig, sich die Ursprungsdaten anzusehen, da ansonsten die Diagramme nicht sehr aussa-

gekräftig sind. Es sollten generell bessere Maße zur Spam-Evaluation entwickelt werden, viele Studien gehen diesem Problem nur unzureichend nach. Die Spam-Erkennungsraten und *False-Positive-Raten* auf den Corpora des Autors sind sehr gut, auf den öffentlich zugänglichen Corpora bewegen sie sich aber nur noch auf mittlerem Niveau. Das könnte daran liegen, dass sich die neueren Spam-Mails auch an Grahams *Plan for Spam* anzupassen beginnen. Aus diesem Grunde finden sich im Internet auch zahlreiche Weiterentwicklungen von Grahams Modell, die meist unter dem Begriff Bayessche Verfahren zu finden sind.

Weiterhin könnte die Sprachproblematik eine wesentlich grössere Rolle spielen als bisher angenommen. Zumindest gibt es darüber keine ausreichenden Studien. Es ist wahrscheinlich, dass sich allein wegen der Daten, vor allem, wenn verschiedene Sprachen in den Daten enthalten sind, die Vergleichbarkeit der Studien stark reduziert. Selbst wenn die wichtigen Studien alle mit englischen E-Mails arbeiten, so bedeutet dies noch lange nicht, dass sie die realen globalen Spam-Probleme treffend abbilden, weil alle Länder, in denen anderssprachige E-Mails verschickt werden, nicht berücksichtigt werden.

Die Evaluierung der Unterscheidung von *Plan for Spam* und dem Naive-Bayes-Klassifikator gestaltet sich mit den bisher üblichen Evaluierungsmaßen nicht einfach. Bezüglich der in Abschnitt 3.5 diskutierten Unterschiede sollte angemerkt werden, daß es durchaus möglich sein könnte, einen Naive-Bayes-Algorithmus ähnlich effizient wie den *Plan for Spam* zu gestalten. Dabei müssten die in die Klassifizierung eingehenden Einzelwahrscheinlichkeiten auf N Worte eingestellt werden, wobei dieses N über eine Heuristik auf vielen Instanzen optimiert werden sollte. Weiter müsste über die Umrechnung der beiden Klassenwahrscheinlichkeiten (siehe Formel 3.15) auf eine Einzelwahrscheinlichkeit hinaus eine weitere Funktion verwendet werden, die die Wahrscheinlichkeiten zu Null und Eins hin konzentriert, um die Auswahl des Schwellenwertes invarianter bezüglich verschiedener Instanzen zu machen. Der Eindruck des Autors, der sich auch in den Testreihen bestätigt, ist, dass Grahams *Plan for Spam* eine Art Instanz eines Naive-Bayes-Algorithmus oder zumindest ein naher Verwandter ist. Man sollte die weitere Forschung im Feature-Engineering eines Naive-Bayes-Algorithmus für verschiedene Sorten von Textklassifizierungsproblemen betreiben. Dabei sollten auch Ansätze mit mehreren Worten als Features untersucht werden. Die verschiedenen freien Parameter des Feature-Engineerings könnten mit Heuristiken auf verschiedenen Instanzen optimiert werden. Damit könnte man auch der Natur von verschiedenen Spam-Separierungsproblemen begegnen.

In Grafik 6.22 finden sich noch einmal alle Testreihen des Naive Bayes im Überblick. Dabei fallen drei Typen von Kurven ins Auge. Testreihen 2, 3 und 6 weisen sehr eckige Graphen auf, die bei geringer *False-Positive-Rate* schon sehr hohe Spam-Erkennungsraten liefern. Dabei ist der Graph der privaten Corpora des Autors aus Testreihe 2 den Testreihen 3 und 6 noch einmal merklich überlegen. Testreihen 4 und 5 weisen eine ähnliche Gestalt auf. In beiden Testreihen wurde Hard-Ham validiert. Sie sind nur entlang der X-Achse verschoben, was

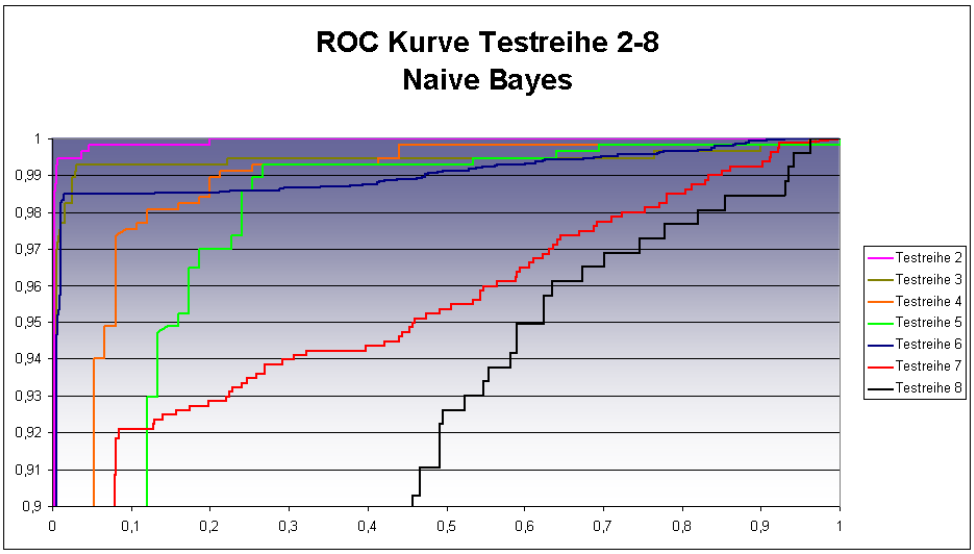


Abbildung 6.22: ROC-Kurven aller Testreihen 1

daran liegt, dass in Testreihe 5 mit einer grösseren Menge Ham-Mails trainiert wurde. Testreihen 7 und 8, die der Auswertung zur Mail-Sortierung dienen, weisen eine völlig andere, aber ähnliche Struktur auf. Weitere Übersichtsgrafiken aller Testreihen finden sich im Anhang B.

Die Übersichtsmaße AUC und 11-Point-Average-Precision in Tabelle 6.3 sind zum Vergleich der beiden Algorithmen nur sehr bedingt geeignet. Die vorher erwähnten Probleme der nicht ausreichenden Datenpunkte beim *Plan for Spam* übertragen sich auf die Übersichtsmaße. Darüberhinaus fassen AUC und 11-Point-Average-Precision die gesamte Kurve des Klassifikators zusammen. Für bestimmte Problemstellungen sind aber nur kleine Teilbereiche der Graphen von Bedeutung. So kann der AUC einer Kurve höher sein als der einer anderen, obwohl die erstere als Spam-Filter ungeeigneter ist, da sie im Bereich von kleinen *False-Positive-Raten* schlechter abschneidet. Zur Evaluation der Testreihen gegeneinander eignen sich die Übersichtsmaße aber gut. So bekommt Testreihe 2 des *Plan for Spam*, die die privaten Ham-Corpora des Autors gegen öffentliche Spam-Corpora separiert, 0.99494 als AUC Wert. Testreihen 3 und 6 belegen die Plätze zwei und drei mit AUC-Werten von 0.98979 und 0.98686. Testreihen 7 und 8, die die schwierigen Separierungsprobleme zur Mail-Sortierung enthalten, liegen weit abgeschlagen mit AUC-Werten von 0.91086 und 0.87509. Die Spalte „Separierbarkeit der Klassen“ in Tabelle 6.3 leitet sich aus den Wahrscheinlichkeitsdiagrammen der Experimente her und gibt somit an, wie klar sich die verschiedenen Klassen bei der Validierung in Wahrscheinlichkeiten abbilden. Die Spalte „Gewinner“ zeigt den Algorithmus, der sich bei genauer Betrachtung besser zur Separierung der entsprechenden Testreihen eignete.

NR	AUC (PFS)	AUC (NB)	11-P. Av. Prec. (PFS)	11-P. Av. Prec. (NB)	Separierbarkeit der Klassen	Gewinner
2	0.99494	0.99824	0.94596	0.97644	sehr gut	?
3	0.98979	0.99265	0.94786	0.94452	mittel	PFS
4	0.98238	0.96371	0.98878	0.98529	schwer	PFS
5	0.97789	0.94355	0.98808	0.98090	schwer	PFS
6	0.98686	0.98796	0.95453	0.95027	sehr gut	?
7	0.91086	0.93008	0.93926	0.90902	sehr gut	NB
8	0.87509	0.84015	0.88773	0.66714	sehr schwer	PFS

Tabelle 6.3: Zusammenfassungsmaße und Übersichtsdarstellung aller Testreihen

Kapitel 7

Zusammenfassung und Ausblick

Nach einer Zusammenfassung der gesamten Arbeit werden weitere Forschungsanregungen aufgeworfen und diskutiert.

7.1 Zusammenfassung

Das Phänomen „Spam“ entwickelte sich ab der Jahrtausendwende rasanter denn je zum Problem für alle Benutzer des E-Mail-Dienstes und richtet auch zunehmend ernsthaften volkswirtschaftlichen Schaden an. Die Definition von Spam-Mail existiert in sehr verschiedenen Ausführungen, eine große Menge der Spam-Mails wird aber von allen involvierten Parteien einheitlich als solche betrachtet. Das Konzept der Spam-Mails funktioniert, weil winzige Antwortraten den Spam-Mail-Versendern genügen, um Profit zu erwirtschaften. Die gesamte Problematik des Versendens von Spam-Mail und die fließenden Übergänge zu halblegalen Marketingkampagnen finden in rechtlichen Grauzonen statt, die Übergangsbereiche werden von Land zu Land juristisch verschieden gehandhabt. Inhalte von Spam-Mails sind zumeist Angebote aus den Bereichen Pornographie, Versicherungen, pharmazeutische Produkte, Angebote aus dem Finanzwesen, Online Casinos, Reisen und fragliche Produkte wie gefälschte Zeugnisse und gestohlene Software. Es gibt verschiedene Ansätze, dem Problem entgegenzutreten.

Rechtliche Bemühungen und wirtschaftliche Konzepte versuchen von vornherein, das Versenden der unerwünschten E-Mails zu verhindern. Falls die E-Mails aber schon unterwegs sind, helfen nur noch Filter beim Endbenutzer, Filter auf Unternehmensebene, Vertrauensnetzwerke, Authentifikationsmechanismen, Spurenrückverfolgung und Black/Whitelists. Spezielle Untersuchungen versuchen auch, die Funktionsweise der Beschaffung der potentiellen Spam-Mail-Adressen zu durchleuchten, um diese eventuell einzuschränken.

In meiner Arbeit wurde die Effizienz des Filters aus Paul Grahams *Plan for Spam* mit einem klassischen Naive-Bayes-Textklassifizierer verglichen. Der Naive-Bayes-Klassifizierer wurde neben dem bereits existierenden *Plan for Spam* Filter

zusätzlich in das E-Mail-Programm des bekannten Open-Source-Browsers Mozilla integriert. Das Mozilla Paket eignet sich relativ gut als Testumgebung für bayessche Filteransätze. Die Funktionen sind gut strukturiert, somit ist es möglich, den eingebauten *Plan for Spam* Algorithmus zu verändern und Performance-Vergleiche durchzuführen.

Da ich meine Untersuchungen mit Mozilla durchführte, kam ich auch mit den sehr eng verwandten Begriffen „Open-Source-Software“ und „freie Software“ in Berührung. Zusammenfassend lässt sich sagen, dass es sehr viele verschiedene Auslegungen von freier Software gibt, dementsprechend gibt es auch viele verschiedene Lizenzmodelle. Maßgeblich an der Entwicklung beteiligt ist das GNU-Projekt, dass ursprünglich geschaffen wurde, um ein völlig freies Betriebssystem zu entwickeln. Weitere beispielhaft angeführte wichtige Vertreter der Open Source Welt sind die verschiedenen Linux Distributionen, die grafischen Oberflächen GNOME und KDE für Linux, die Mozilla Organisation und MySQL. Die Funktionsweise von Open-Source-Projekten im Netz funktioniert ähnlich einer Evolution. Jeder versucht, Programme zu verbessern und erzeugt damit Mutationen, die schlechten Versionen sterben automatisch aus, da sie nicht mehr benutzt und angeboten werden. Trotz alledem bedarf es in großen Open Source Projekten leitenden Instanzen die letztendlich entscheiden, was in die Release Version kommt.

Bevor ich die Ergebnisse des Vergleiches zwischen Naive Bayes und Grahams *Plan for Spam* zusammenfasse, möchte ich noch kurz auf die Fehlerquellen bei der Evaluierung von Spam-Filtern eingehen. Die Evaluierung von Spam-Filtern ist äußerst fehleranfällig und nicht zu unterschätzen. Zwischen den Ergebnissen aus den wissenschaftlichen Berichten und der Realität tut sich aus verschiedenen Gründen eine bedeutsame Kluft auf. Die Ergebnisse sind durchwegs zu optimistisch, hauptsächlich weil die Benutzbarkeit der Spam-Filter außer Acht gelassen wird, oftmals auch wegen einer Überidealisierung der Spam-Corpora.

Das Hauptproblem ist, dass reale Ham-Mail-Corpora kaum existieren weil Ham-Mails privat sind und keiner sie in irgendeiner Ham-Mail-Sammlung wiederfinden möchte, deshalb sind Ham-Mail-Sammlungen meist nur eine Approximierung der Realität. Meine eigene Ham-Mail-Sammlung ist z.B. fast durchwegs nur auf deutsch, die verwendeten Spam-Corpora sind aber durchwegs auf englisch. Daraus ergibt sich eine Verzerrung in Form der Sprachproblematik. Oft stammen die Corpora aus verschiedenen Zeiten und deshalb kann der Spam-Filter Spam-Mails z.B. am Datum erkennen. Wichtig wäre meiner Meinung nach die bessere Organisation und Pflege irgendwelcher internationalen Spam und Ham-Corpora.

Insgesamt kann man als Ergebnis meiner Untersuchung festhalten, dass Grahams *Plan for Spam* sehr effizient zur Spam-Erkennung eingesetzt werden kann. Der Naive-Bayes-Algorithmus ist am deutlichsten bei schwierigen Separierungsproblemen unterlegen. Eine Kernaussage von Grahams „Plan for Spam“ scheinen die 15 „aussagekräftigsten“ Wörter zu sein, deren Wahrscheinlichkeit sich am weitesten weg von neutral, sprich 0.5, befindet. Das Einsatzgebiet des Naive-Bayes-Algorithmus scheint eher in anders strukturierten Texterkennungsproblemen zu

liegen.

Es gibt ein Defizit an vernünftigen Evaluierungsmaßen für Spam-Algorithmen, so eignen sich die verwendeten Maße ROC-Kurve und Recall-Precision-Diagramm für den Naive-Bayes-Algorithmus sehr gut, für den *Plan for Spam* aber nur bedingt. Die Sprachproblematik sollte mehr mit in die Forschung einbezogen werden. Letztendlich bin ich der Meinung, dass vor allem in der Benutzerfreundlichkeit oder der praktischen Anwendung von Spam-Filtern grosse Defizite bestehen.

7.2 Eigene Forschungsanregungen

Mir ist während meiner Diplom-Arbeit aufgefallen, dass ein Engpass im Verbreiten von Spam-Filtern besteht, dass die Anwender die Spam-Filter nicht wirklich verwenden oder kein Gefühl dafür haben, wie sie funktionieren. Es werden im Allgemeinen eher 2-5 Spam-Mails am Tag gelöscht, als sich wirklich mit den Spam-Filtern zu beschäftigen, vor allem mit denen, die man selbst trainieren muss. Ähnlich wie bei einer großangelegten Marketing-Kampagne ein Link auf einer Internetseite der etwas schwer zu sehen ist, alles zerstören kann, so kann auch eine winzige Kleinigkeit in der Bedienung eines Spam-Filters dazu führen, dass dieser fast nicht verwendet wird, außer von denen, die wirklich beruflich sehr viele E-Mails bekommen und die auch keine psychische Blockade gegen die Verwendung von Spam-Filtern haben. Was ich damit sagen will, ist, um das Phänomen Spam zu stoppen, bedarf es eines großanlegten Herausfilterns der Spam-Mails. Aber gerade die unbedarften Nutzer benutzen Spam-Filter nicht, weil es ihnen zu kompliziert erscheint.

Hier ist meiner Meinung nach Nachholbedarf im Bereich Software Ergonomie vorhanden. Es ist für die Gesamtreduktion von empfangenen Spam-Mails wichtiger, die Bedienbarkeit der Spam-Filter zu erhöhen, als die Treffer Quoten der Spam-Filter. Diese Aspekte bleiben bei wissenschaftlichen Untersuchungen meist unberücksichtigt, sind aber für wirtschaftliche Zusammenhänge äußerst bedeutsam.

So wissen immerhin 84.4% einer Testgruppe einer Studie zur Software Ergonomie von Spam-Filtern, dass ihr E-Mail Provider einen Spam-Filter anbietet. 8.2% können diesen nicht selbstständig an- oder ausschalten. Ein Drittel der Benutzer weiß nicht, wie der Spamschutz genutzt wird. Defizite gibt es auch in den erzielten Trefferquoten. So gaben 6.9% der Personen der Testgruppe an, dass sehr oft auch Ham E-Mails aussortiert werden. Bei 33.4% ist dies manchmal der Fall!

Damit zeigt sich, wie von mir vermutet eine riesige Divergenz zwischen den Angaben in den Spam-Filter-Studien und der Realität. Ich versuche mich bei solchen Gedanken in die Situation eines „normalbegabten“ Computernutzers zu versetzen, in dem ich z.B. bei meinem E-Mail Provider die Zeit betrachte, die ich brauche, um den Spam-Filter abzuschalten. Sind das bei mir als Informatik-Student schon 60 Sekunden, weiß ich, dass der normale Benutzer diesen Filter

niemals wird abschalten können, oder zumindest wird er niemals die Zeit aufbringen, es herauszufinden. Zurück zur Studie. 16.8% der Benutzer achten nicht darauf, was aussortiert wird und was nicht, 15.3% nutzen den Spam Schutz gar nicht. Somit beeurteilen nur 35.2% ihren Spam Schutz als zweckmäßig und ausreichend. Bei dieser Studie ist nur von providerbasiertem Spamschutz die Rede. Der Mozilla Spam-Filter dürfte nur für 1-3% der Benutzer überhaupt von Bedeutung sein, wenn man von einer Verbreitung des Mozilla Browsers von zur Zeit 6% ausgeht. Weitere Informationen zur Softwareergonomie-Problematik von Spam-Filtern finden sich unter [44].

Wie wirken sich eigentlich unterschiedliche Sprachen auf die Spam Problematik aus? In Deutschland sind doch meistens die Ham-Mails in deutsch und die Spam-Mails in englisch verfasst, wie ist das in anderen Ländern? Welche Schwierigkeiten oder Chancen könnten sich daraus ergeben?

Meiner Meinung nach wäre es sinnvoll, sehr viel Wert auf die Pflege von Spam- und Ham-Corpora zu legen, vielleicht an einer zentralen Stelle, von einer zentralen Organisation. Mit diesen immer aktuell gehaltenen Spam-Mails sollten die freien Parameter für diverse Verfahren optimal bestimmt werden und dann nur die Parameter automatisch an die Spam-Filter übertragen werden, weil wie schon in meinen Andeutungen zur Softwareergonomie ausgeführt, ein eigenständiges Justieren der Benutzer unrealistisch erscheint.

Anhang A

Liste einsetzbarer Spam-Filter, Spam-Konferenzen und weitere Informationen

Name	Server	Client	Verfahren
Alchemy Spamfilter		×	Bayes
AOL Spamfilter	×		?
Atqui Spam-Filter	×		Authentifizierung per Turing Test
Barracuda Spamfilter		×	Kombination
Bitdefener Spamfilter	×	×	Kombination
Bsfilter		×	Bayes
ChoicE-Mail One	×	×	Kombination
Cleanfeed Usenet Spamfilter	×	×	Bayes
Cleanmail	×	×	?
CRM 114	×	×	CRM 114
E-MailProtect		×	?
G-Lock SpamCombat	×		Kombination
GoodbyeSpam		×	?
Hexamail Spamfilter	×	×	?
iHateSpam		×	?
ITIC Spamfilter		×	?
jwSpamSpy		×	Hashing, Blacklists
K9		×	Blacklists, Whitelists, stochastisch
MailMate		×	?, Zurückweisungsfunktion*
MailSanctity Spamfilter		×	Whitelists, stochastisch
Mailshield	×		Kombination
MailWasher Pro		×	Blacklists, Whitelists, stochastisch

Fortsetzung auf der nächsten Seite

Name	Server	Client	Verfahren
Matador		×	Bayes, Blacklists, Whitelists
McAfee Spam Killer	×	×	Kombination
Mozilla Spam-Filter		×	Grahams Plan for Spam
Outlook Spam-Filter		×	Bayes, Blacklists, Whitelists
Polesoft Spamfilter		×	?
Quick Spamfilter	×	×	Bayes
Qurb		×	Whitelists, stochastisch
Rmail Spamfilter		×	?
Spam Agent	×	×	Kombination
Spam Bully		×	Bayes
Spam Buster		×	?
Spam Monitor		×	stochastisch, Blacklists, Whitelists
SpamAssassin	×	×	Punktesystem, Kombination
SpamBayes	×	×	Grahams Plan for Spam, variiert
SpamButcher		×	Fuzzy Logic
SpamCatcher		×	Fuzzy Logic, Bayes, Heuristiken
SpamEater Pro		×	Kombination
Spamihilator		×	Bayes
SpamNet		×	P2P
Spampal		×	?
SpamSubtract Pro		×	?
Symantec Brightmail	×	×	Kombination
T-Online SpamSchutz		×	stochastisch, Blacklists, Whitelists

*** Zurückweisungsfunktion:** Der Spam-Filter sendet bei Erkennung von Spam eine E-Mail zurück, die besagt, die Mail-Adresse existiere nicht.
Kombination: Damit gemeint sind mehrstufige Systeme, die alle bekannten Methoden kombinieren.

Wichtigste Spam Konferenzen

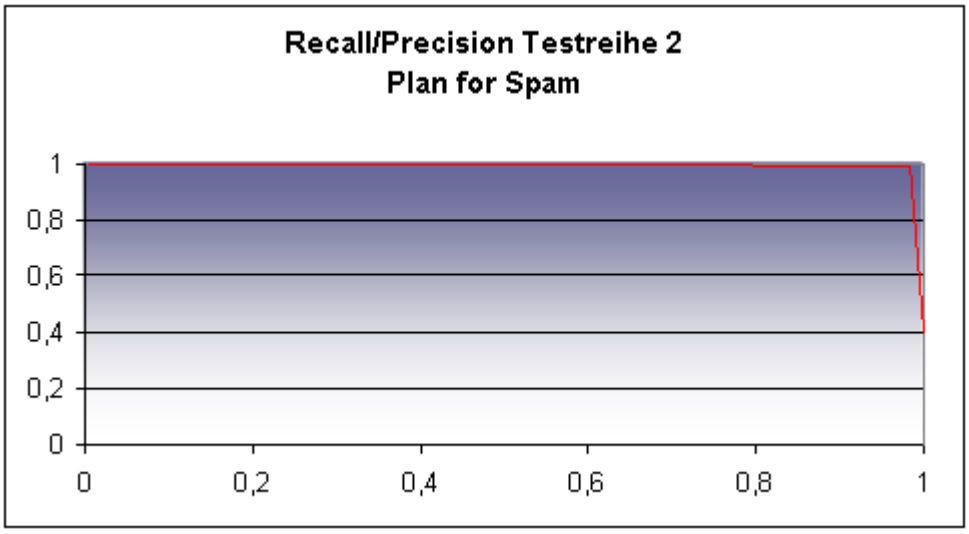
- MIT Spamconference [62]
- Conference on E-Mail and Anti-Spam [63]
- Eine Übersicht anstehender Spam Veranstaltungen findet sich unter [65][66]

Weitere Informationen zur Spam Problematik im Internet

- Spam Statistik[54], eine ältere Spam Statistik 1996-2002. Besonderheit: Eine Liste von Aktien deren Kurse durch Spam-Mails versucht wurden zu manipulieren
- SpamCop[55], pflegt eine Liste von IP Adressen von Servern die Spam-Mail verschickt haben, arbeitet mit Internet Service Providern zusammen
- Halverson Spam Statistik[56], visualisierte Statistiken von 2002-2004 über die Spam Entwicklung
- Spam Statistik[58], Spam Statistik 1997-2004
- Gute Spam-Linksammlung[59]
- Tool zur Bearbeitung von E-Mail-Dateien im Spam Kontext[60]

Anhang B

Alle Auswertungsgrafiken der Testreihen



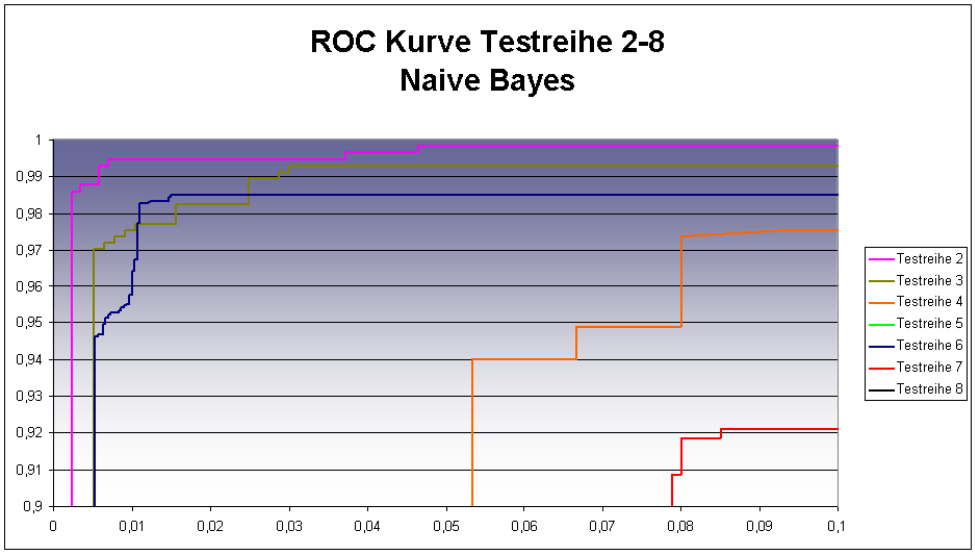


Abbildung B.1: ROC-Kurven aller Testreihen 2

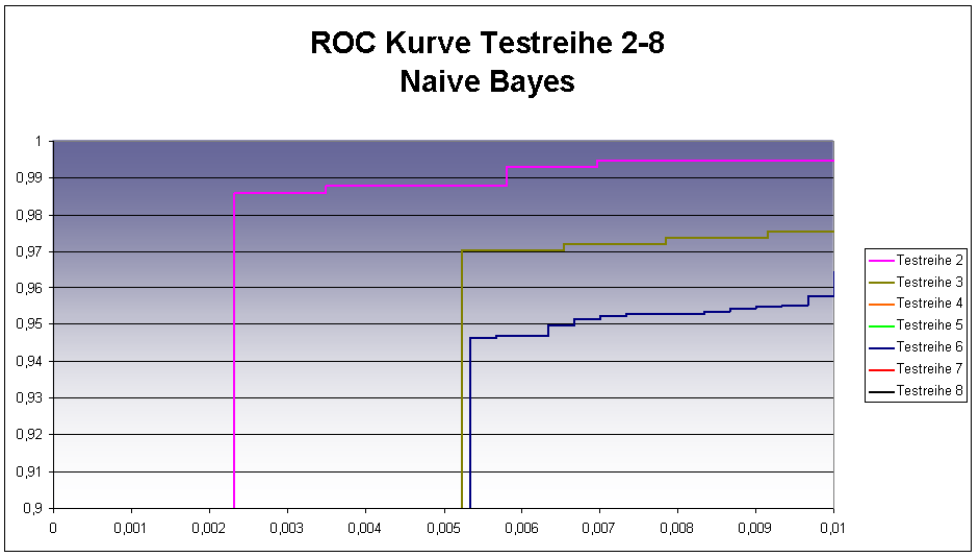


Abbildung B.2: ROC-Kurven aller Testreihen 3

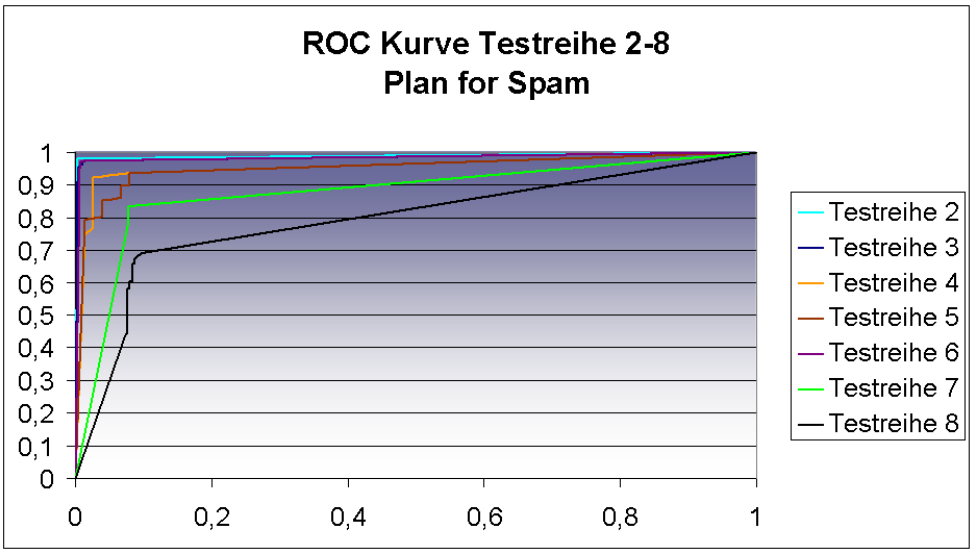


Abbildung B.3: Recall-Precision-Diagramm aller Testreihen 1

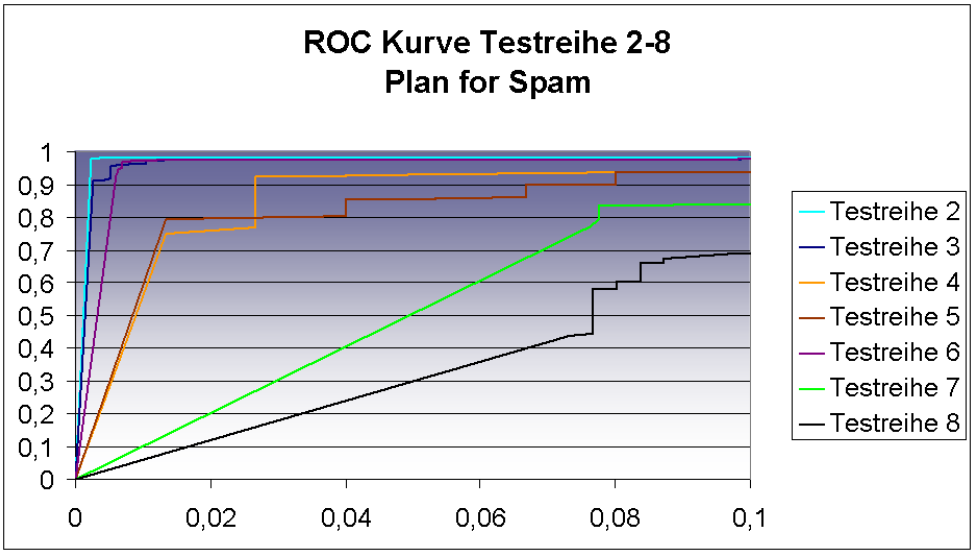


Abbildung B.4: Recall-Precision-Diagramm aller Testreihen 2

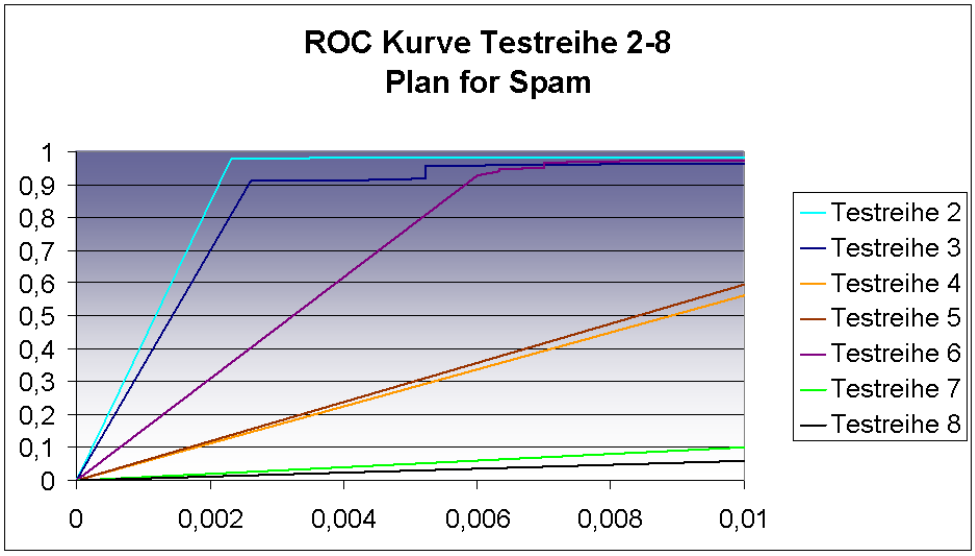
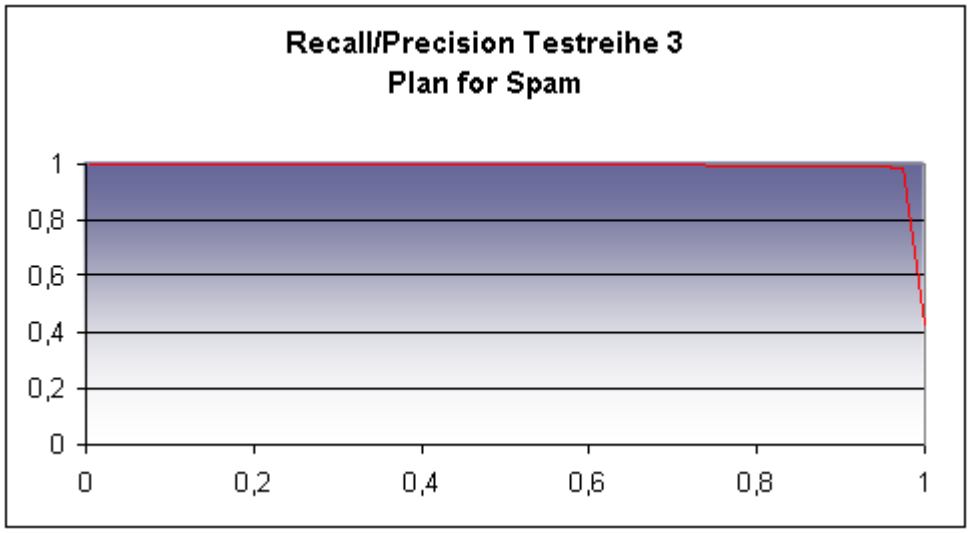
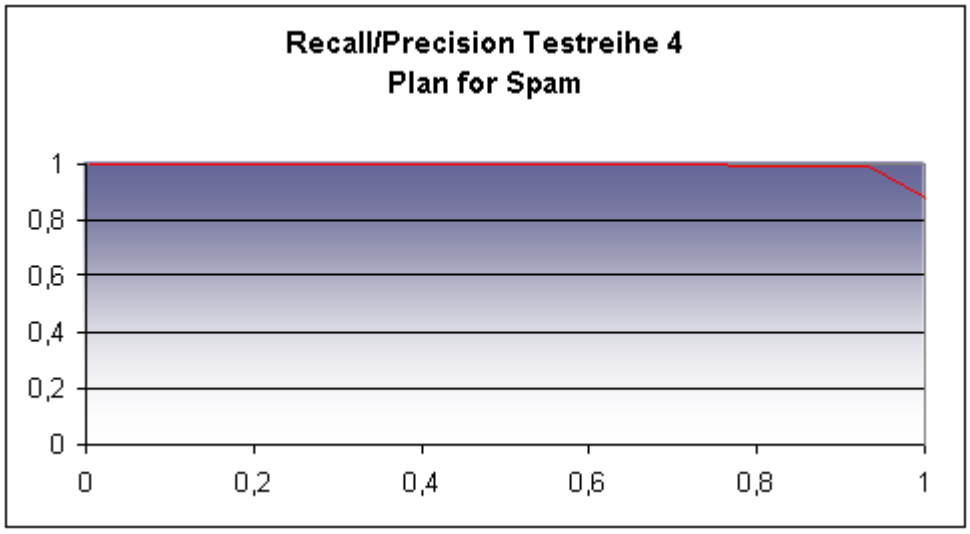
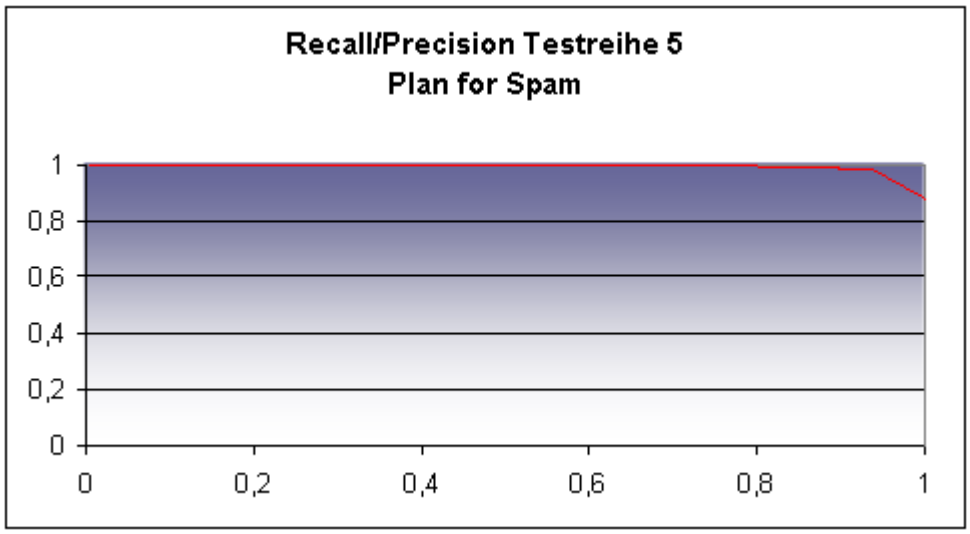
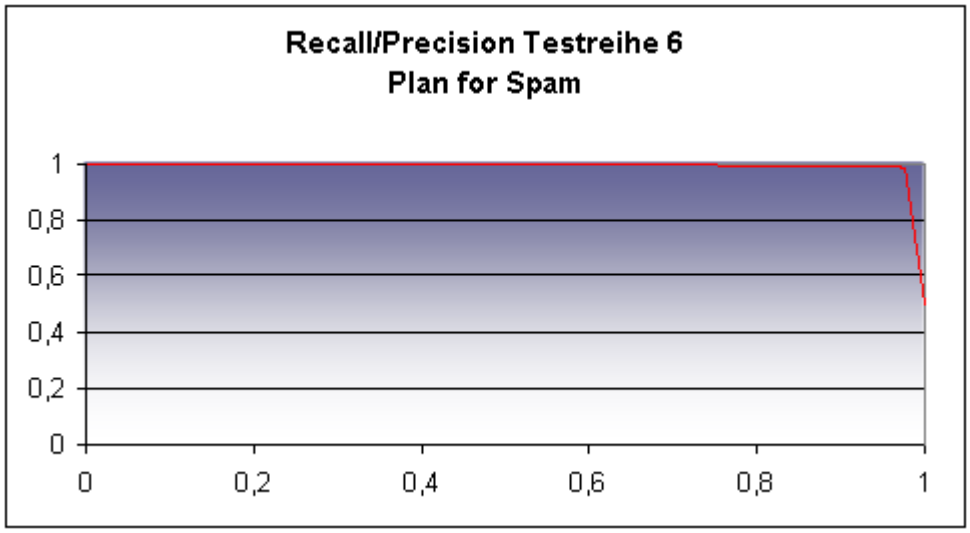


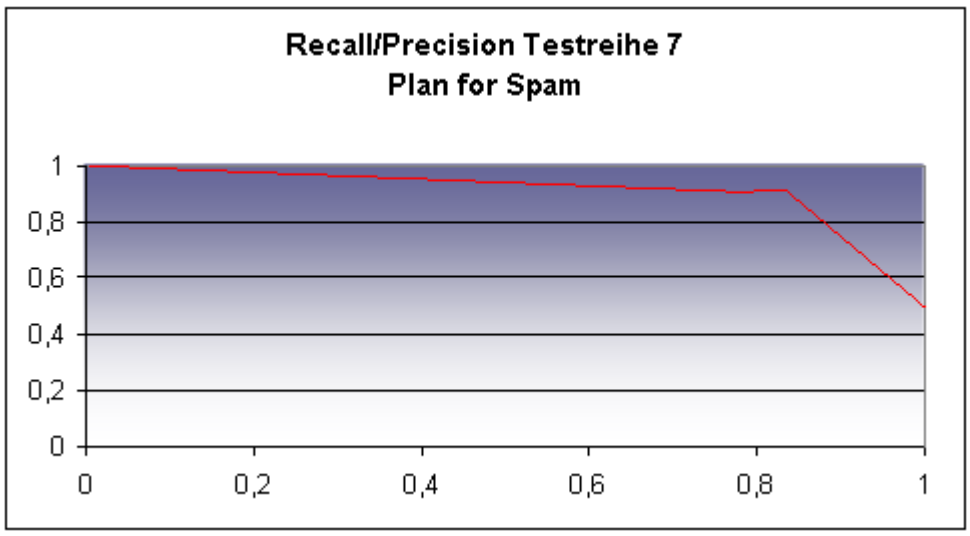
Abbildung B.5: Recall-Precision-Diagramm aller Testreihen 3

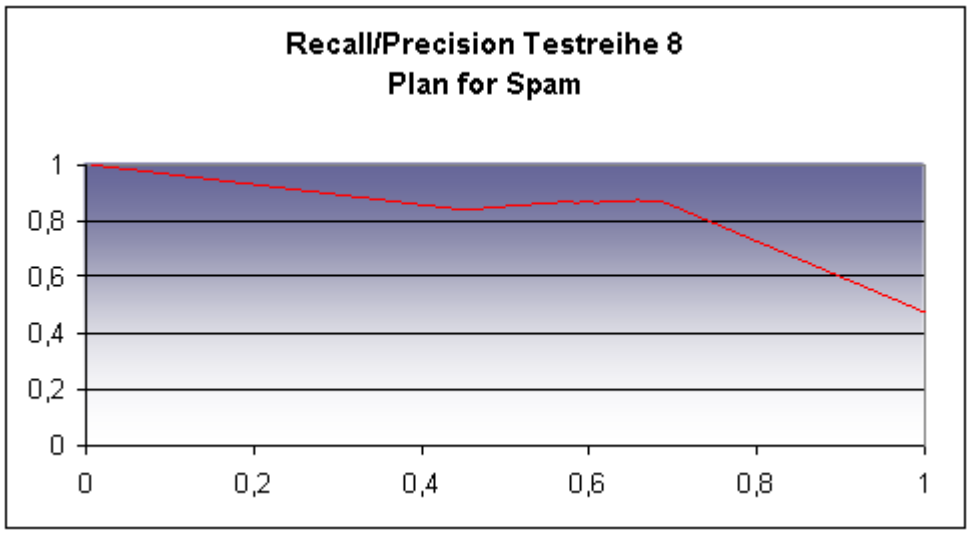


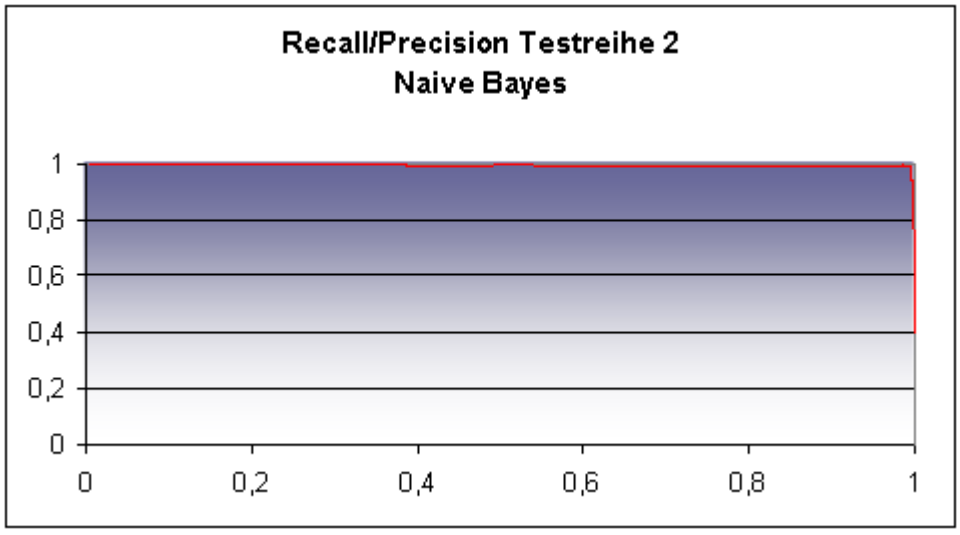

Recall/Precision Testreihe 4
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

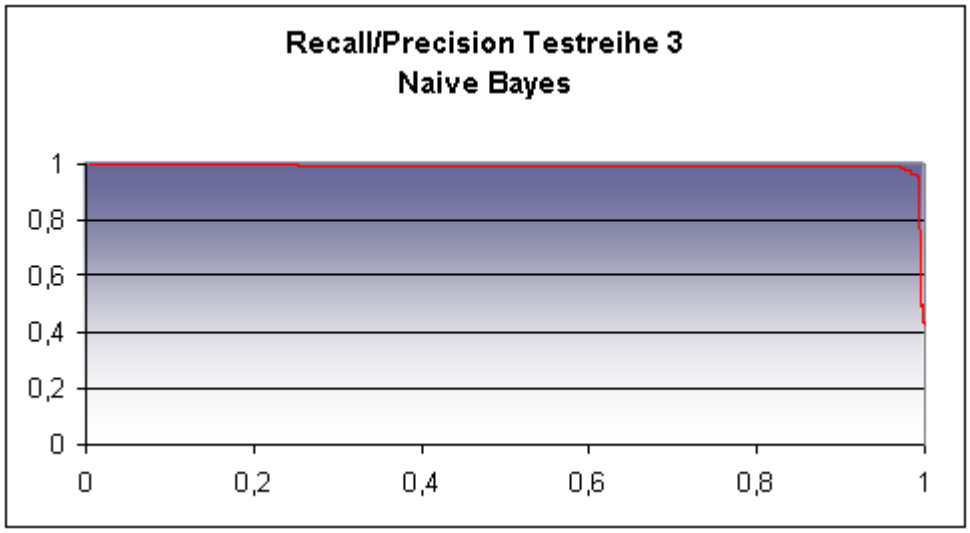

Recall/Precision Testreihe 5
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

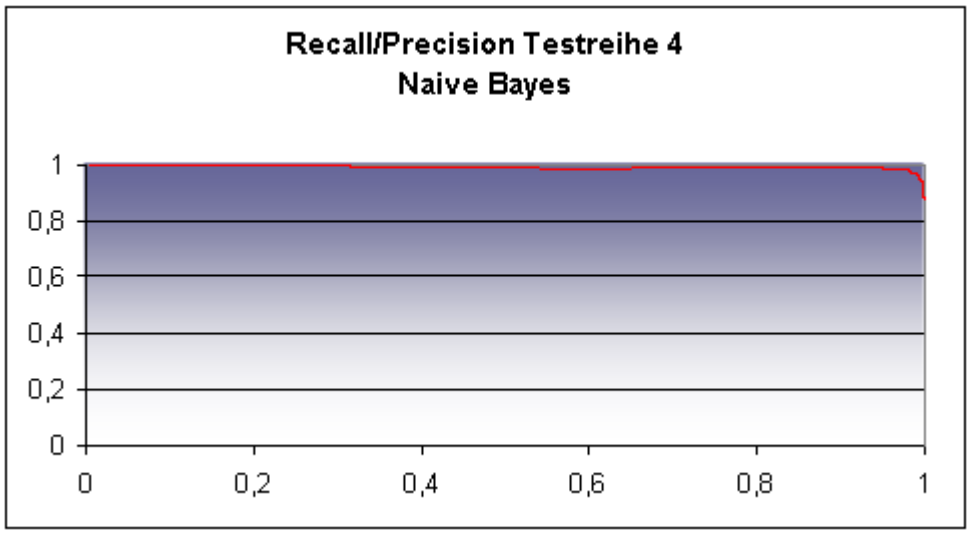

Recall/Precision Testreihe 6
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

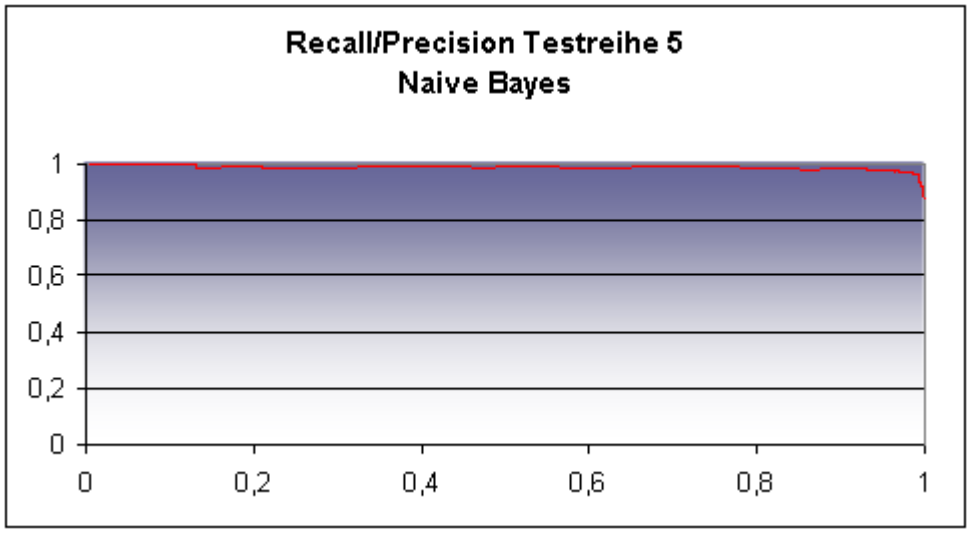

Recall/Precision Testreihe 7
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

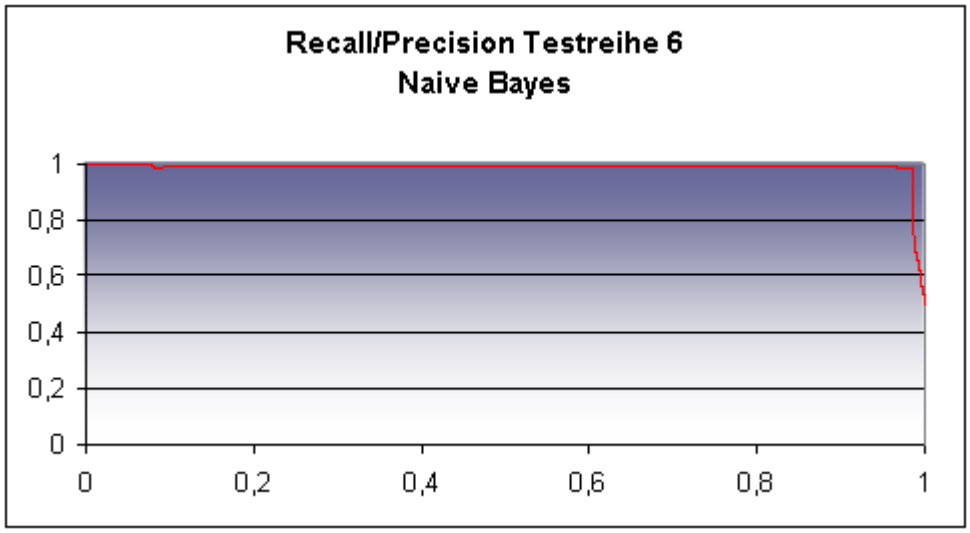

Recall/Precision Testreihe 8
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

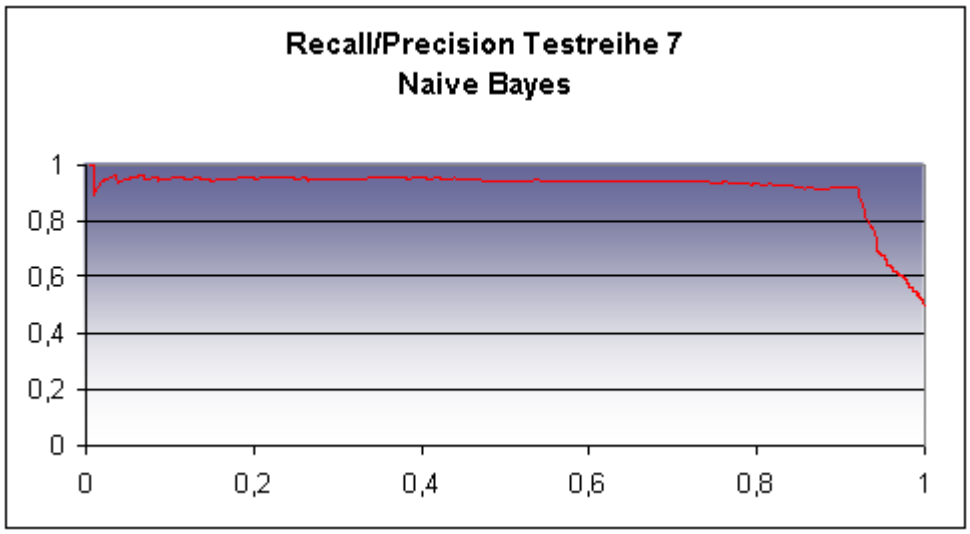

Recall/Precision Testreihe 2
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

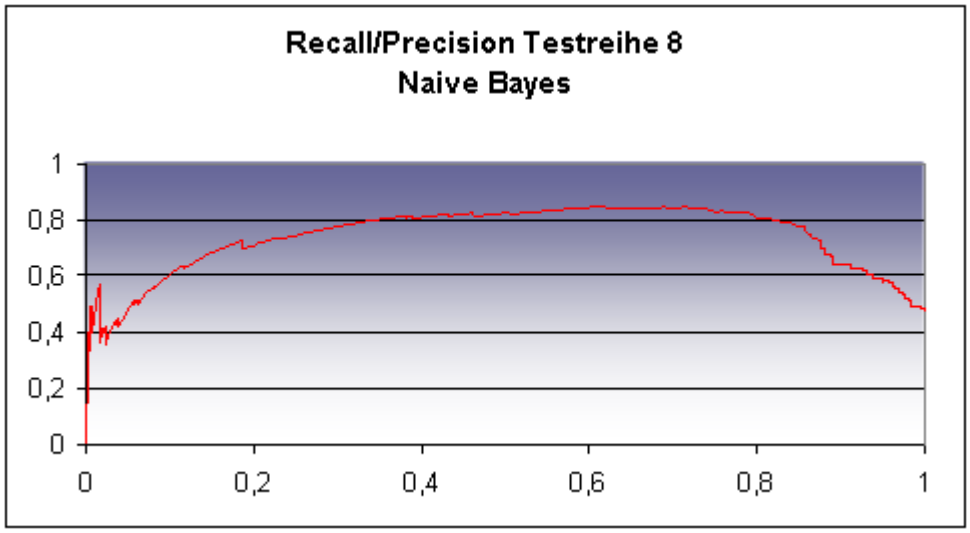

Recall/Precision Testreihe 3
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

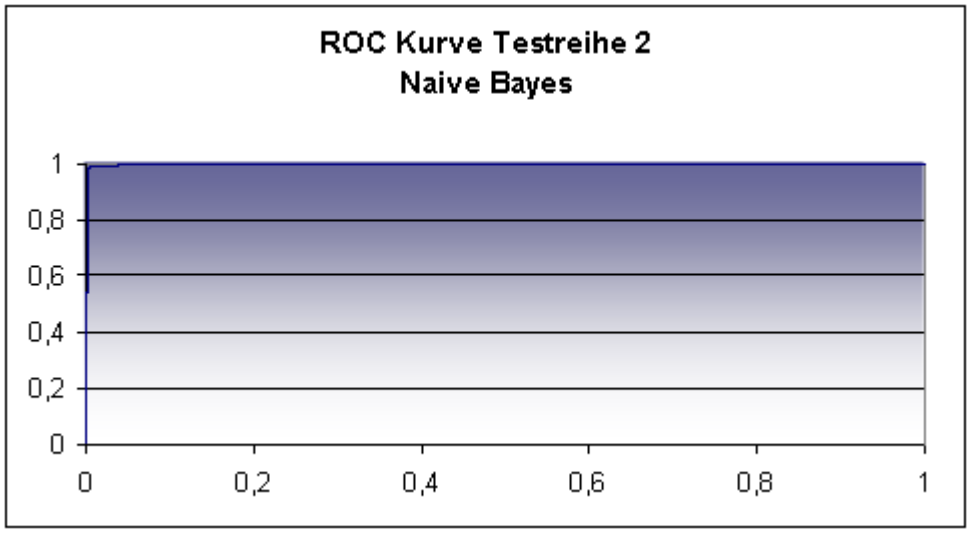

Recall/Precision Testreihe 4
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

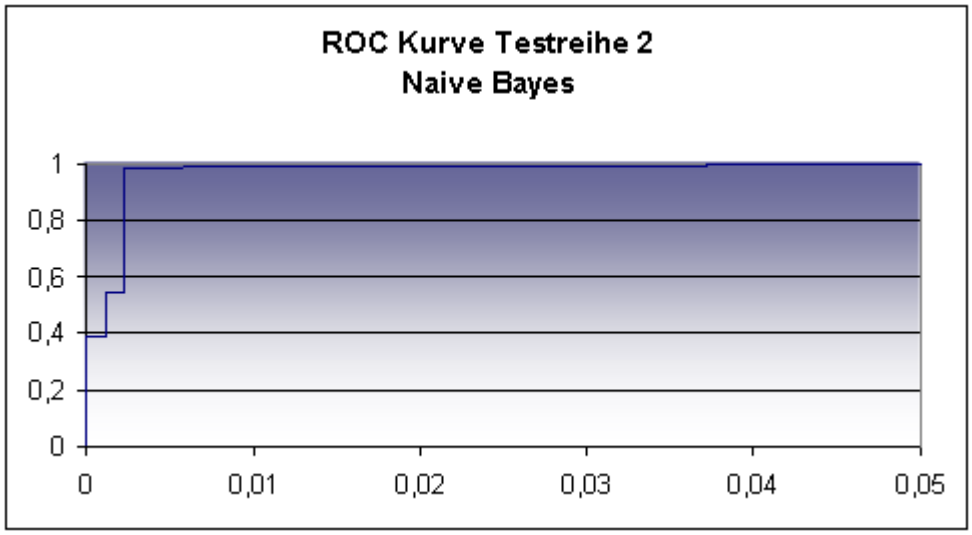

Recall/Precision Testreihe 5
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

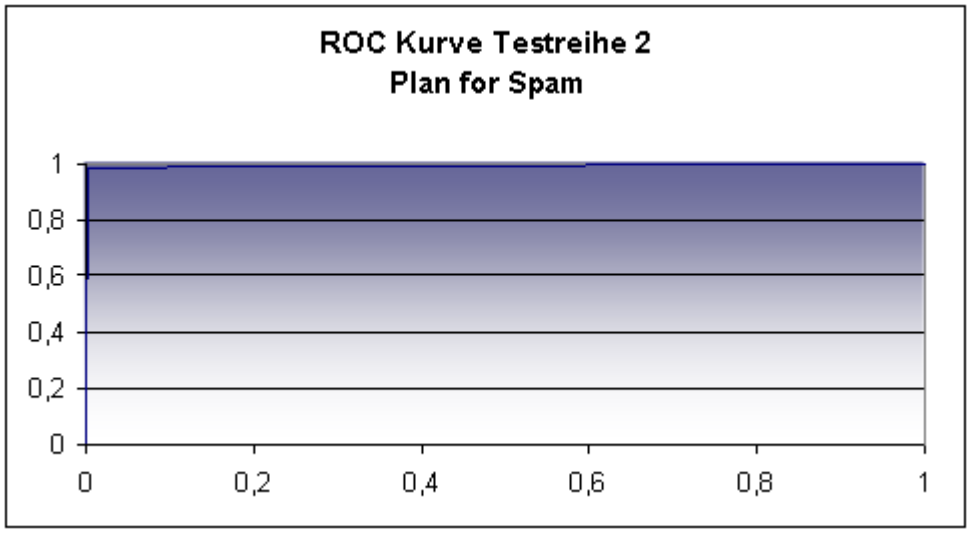

Recall/Precision Testreihe 6
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

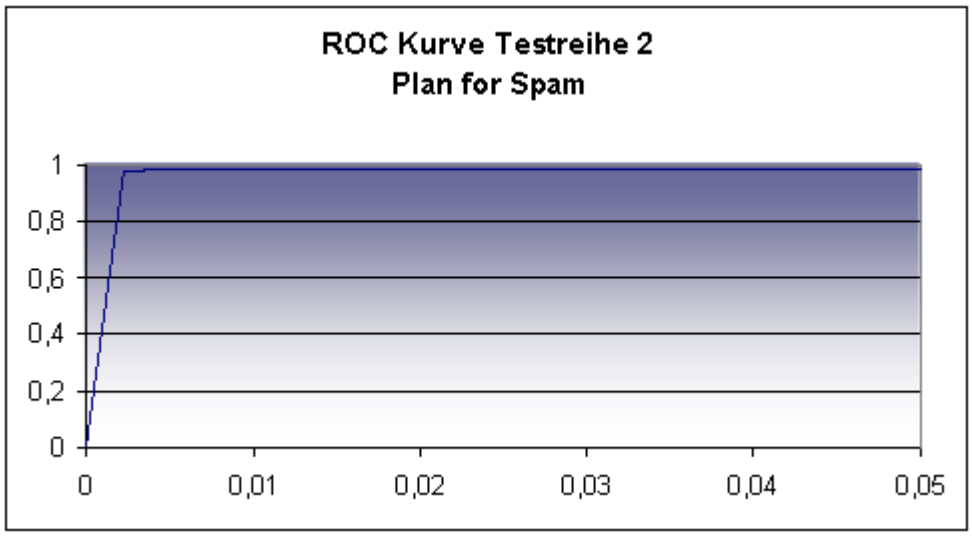

Recall/Precision Testreihe 7
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

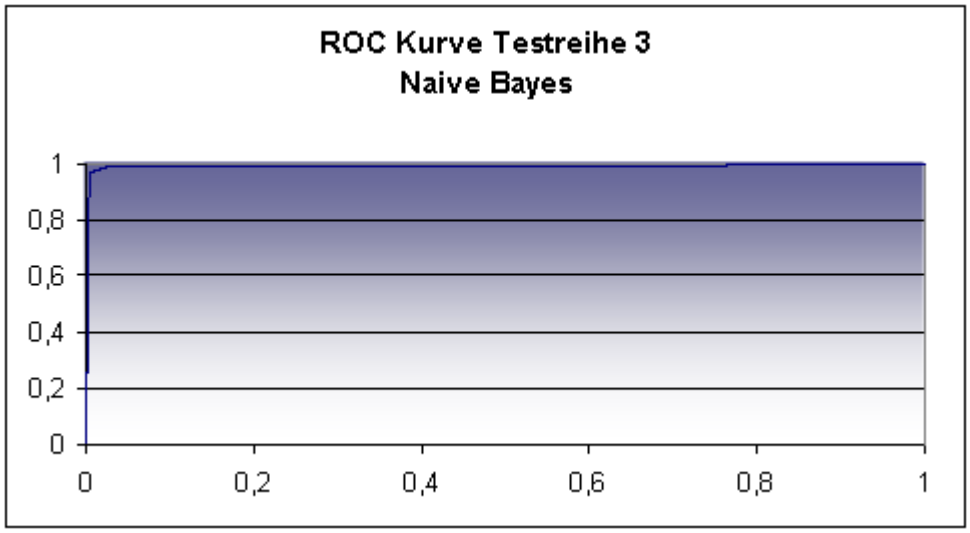

Recall/Precision Testreihe 8
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

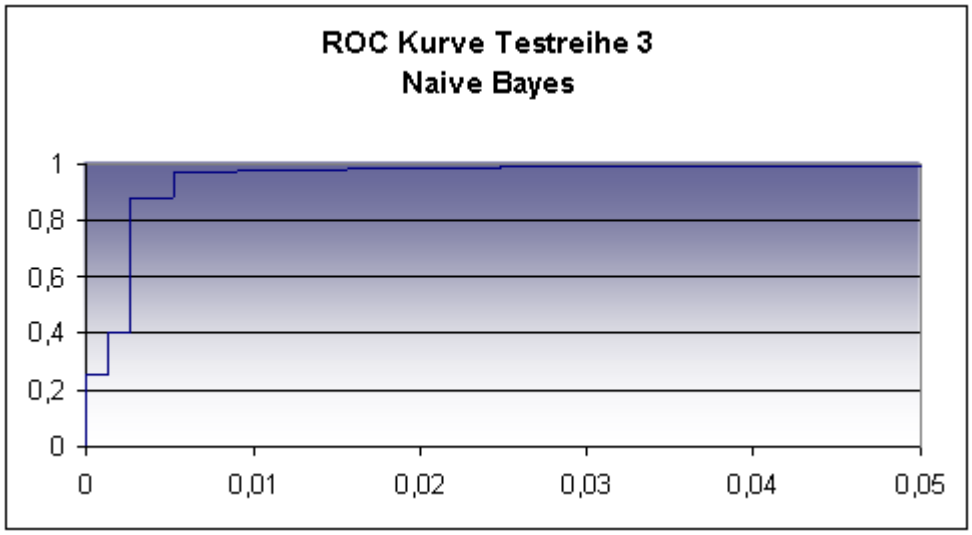

ROC Kurve Testreihe 2
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

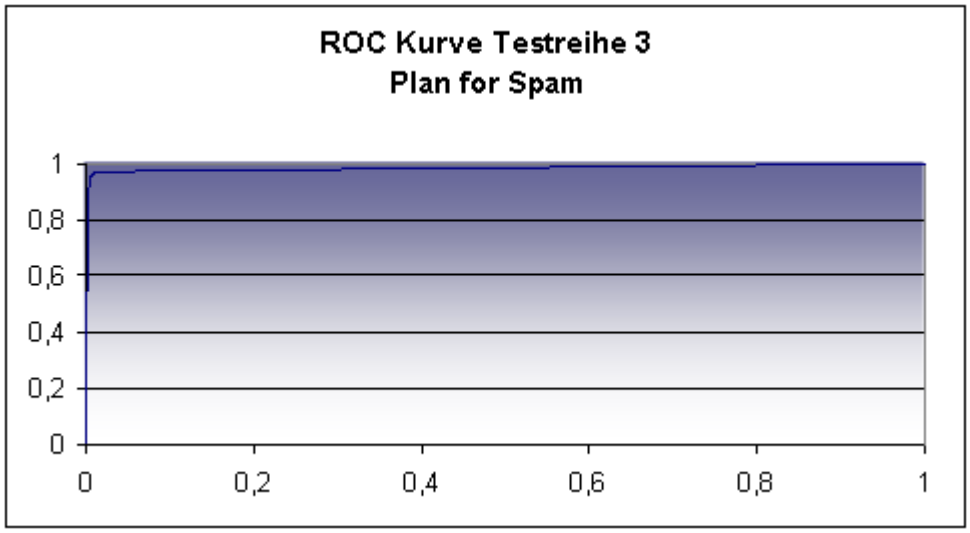

ROC Kurve Testreihe 2
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

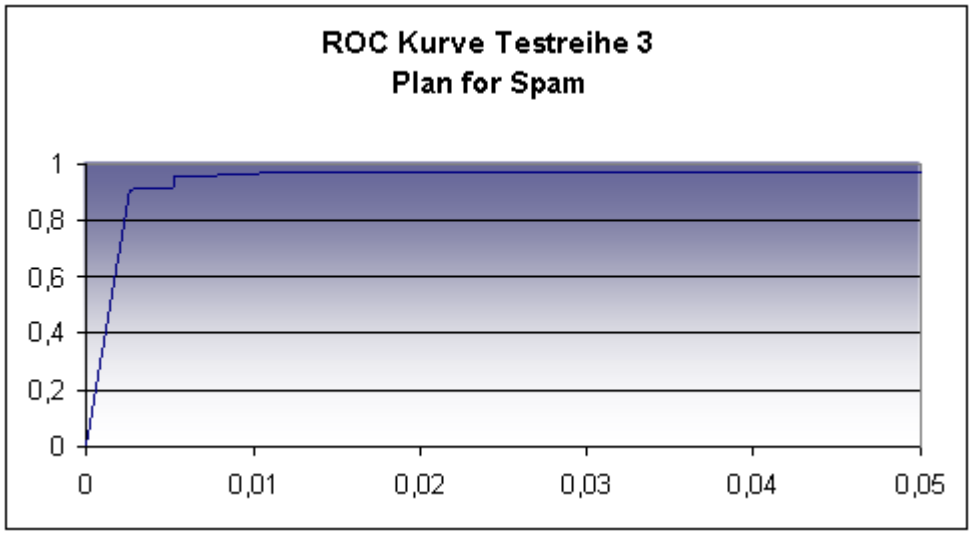

ROC Kurve Testreihe 2
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

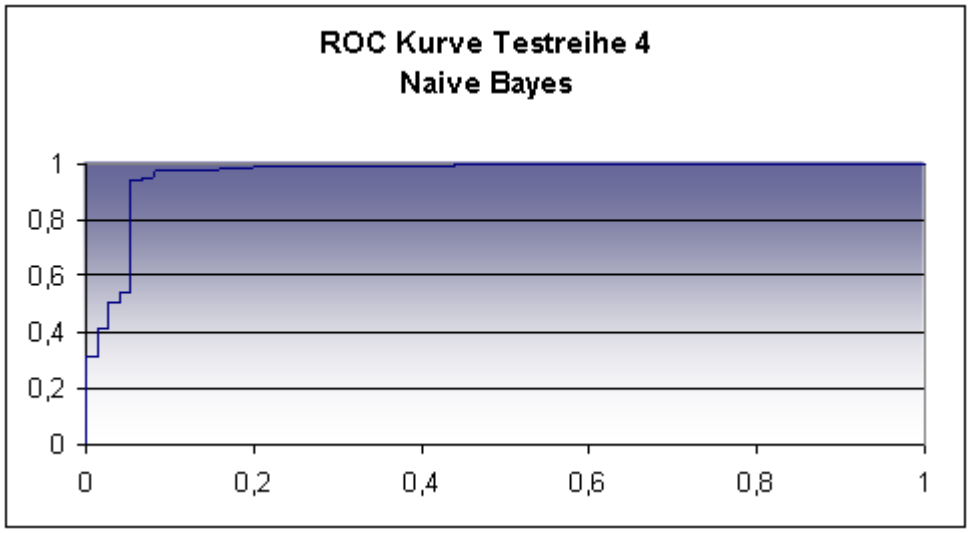

ROC Kurve Testreihe 2
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

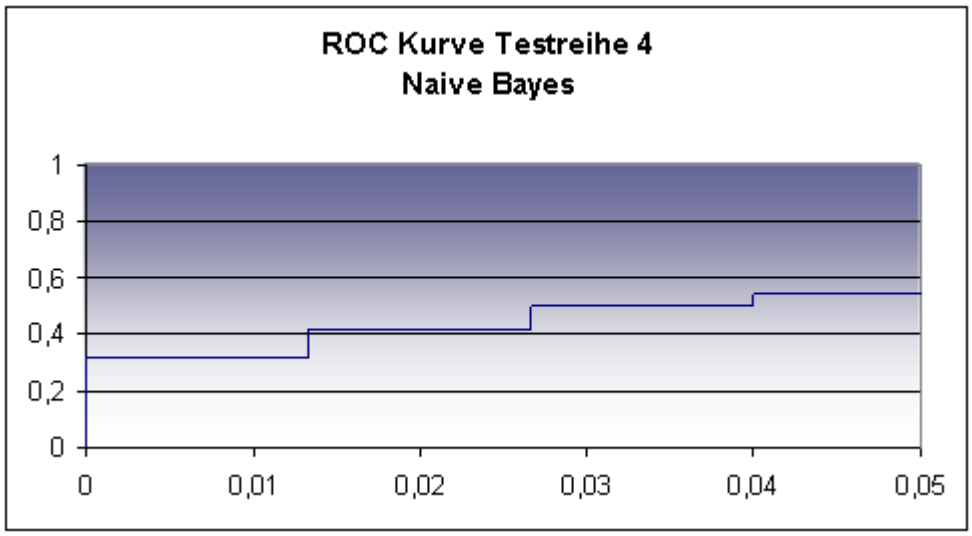

ROC Kurve Testreihe 3
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

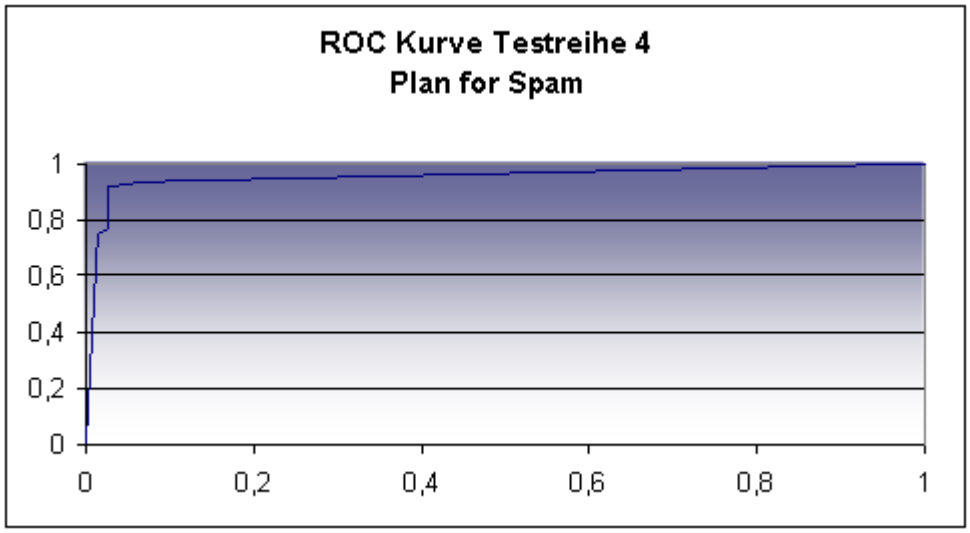

ROC Kurve Testreihe 3
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

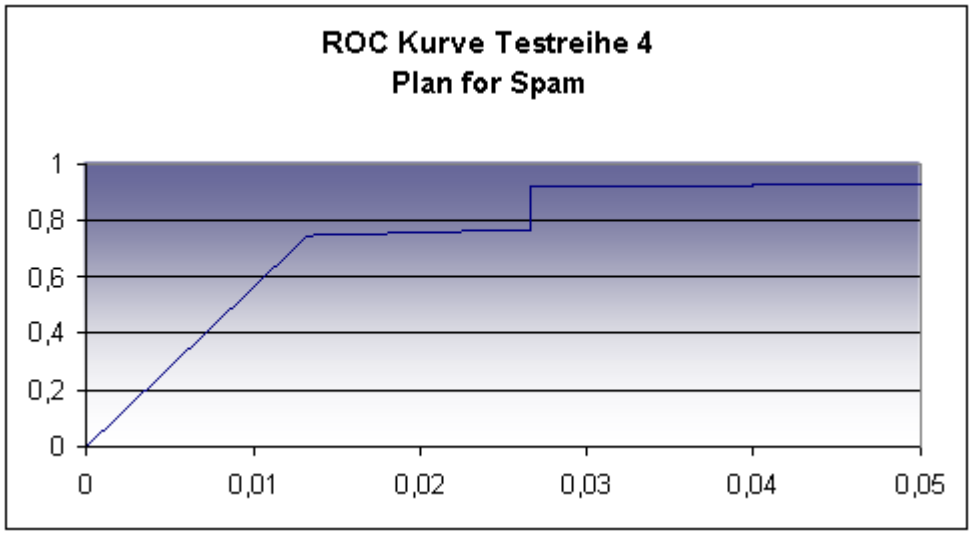

ROC Kurve Testreihe 3
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

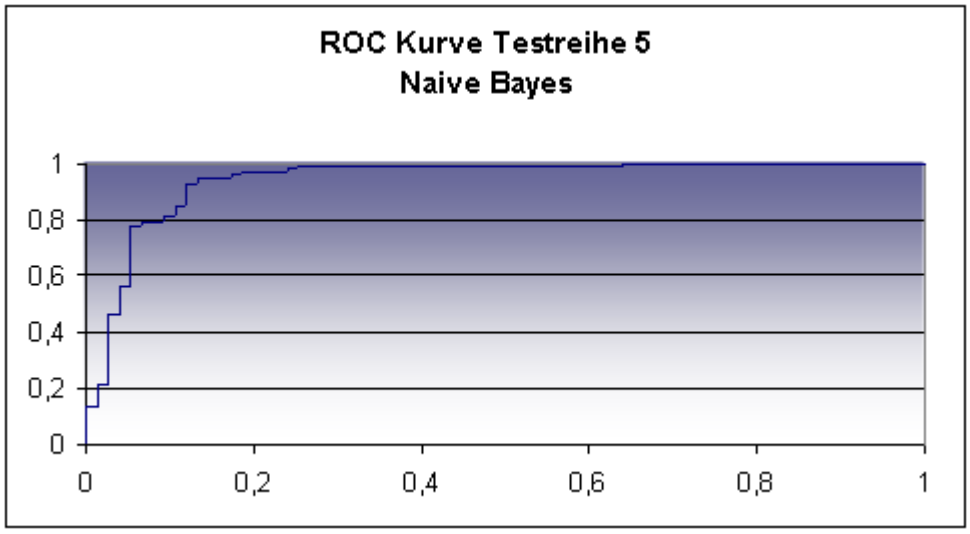

ROC Kurve Testreihe 3
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

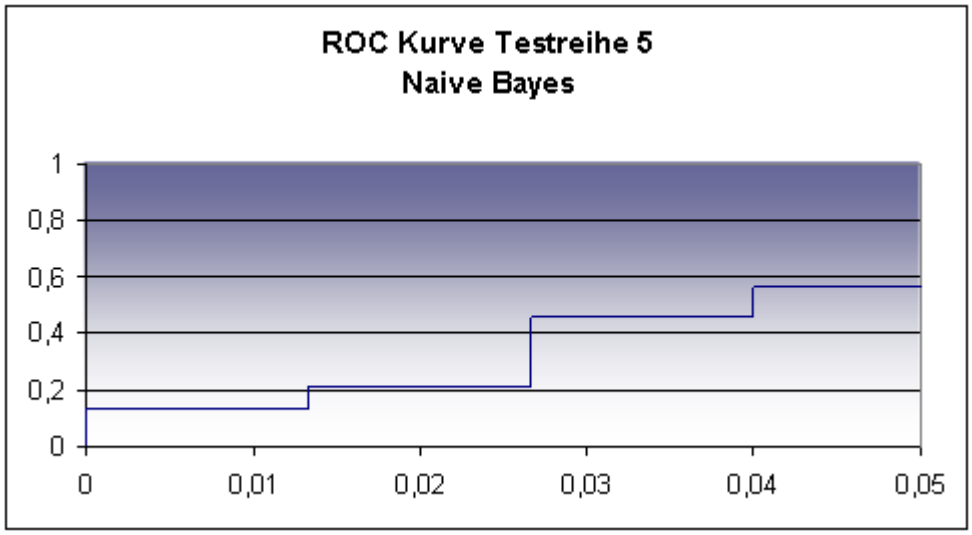

ROC Kurve Testreihe 4
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

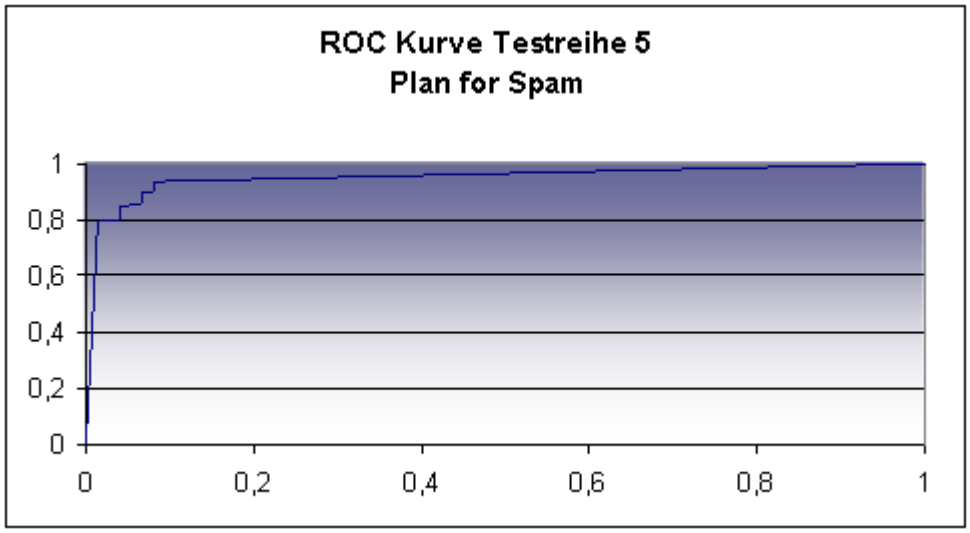

ROC Kurve Testreihe 4
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

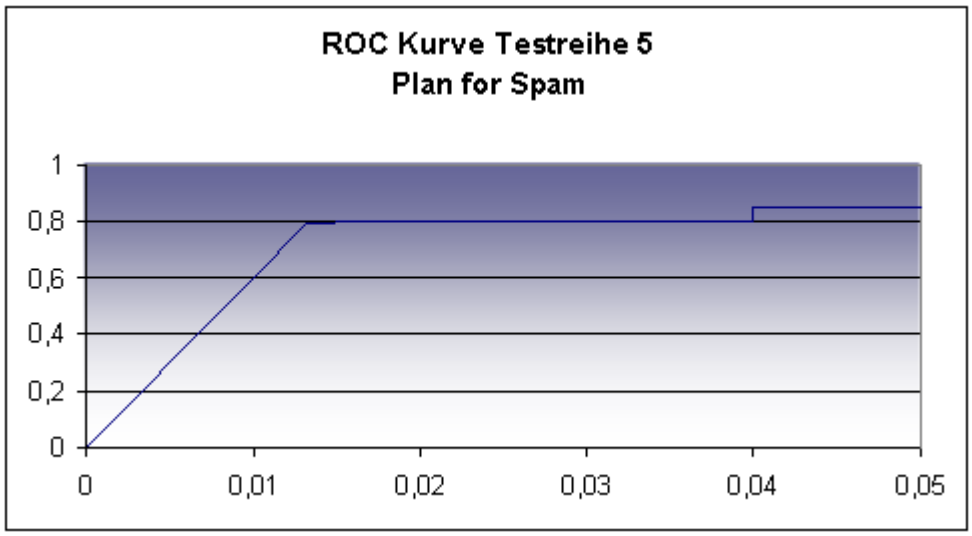

ROC Kurve Testreihe 4
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

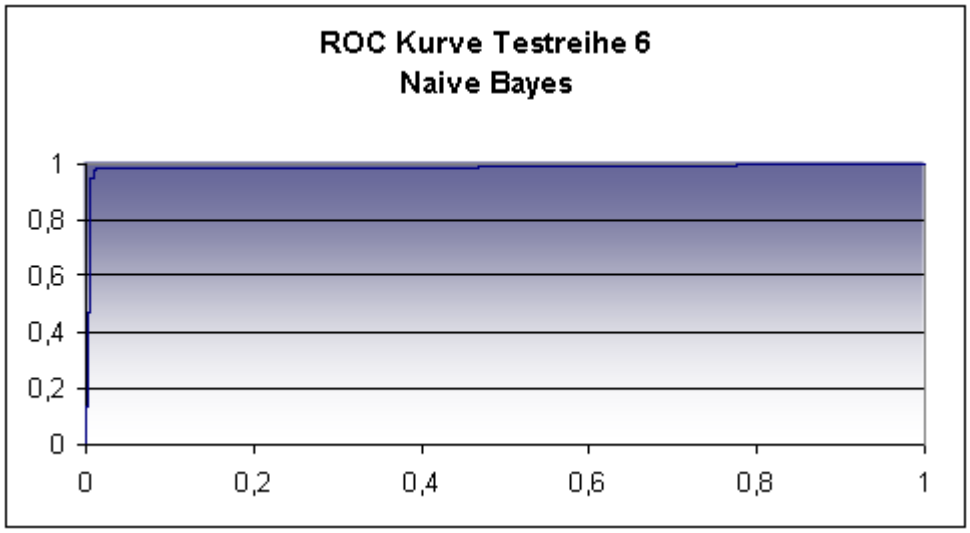

ROC Kurve Testreihe 4
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

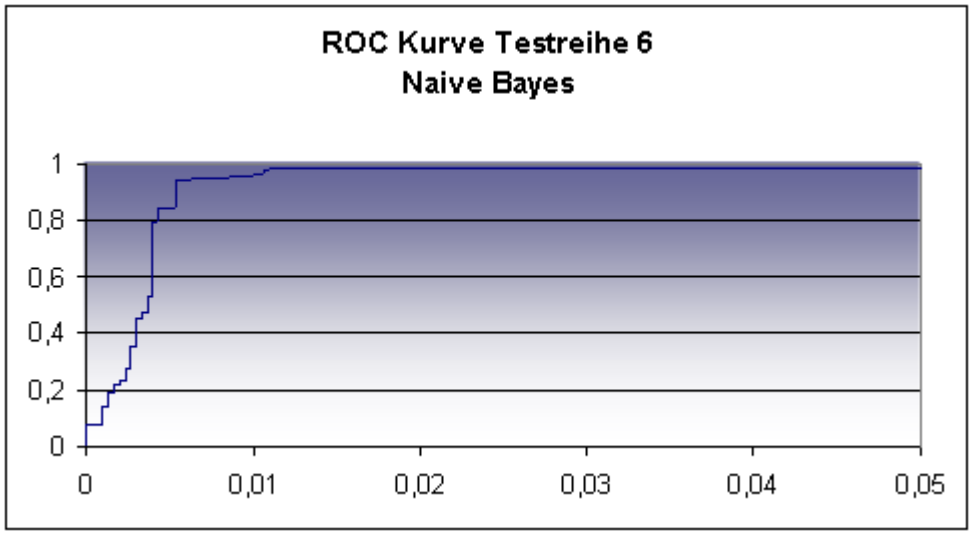

ROC Kurve Testreihe 5
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

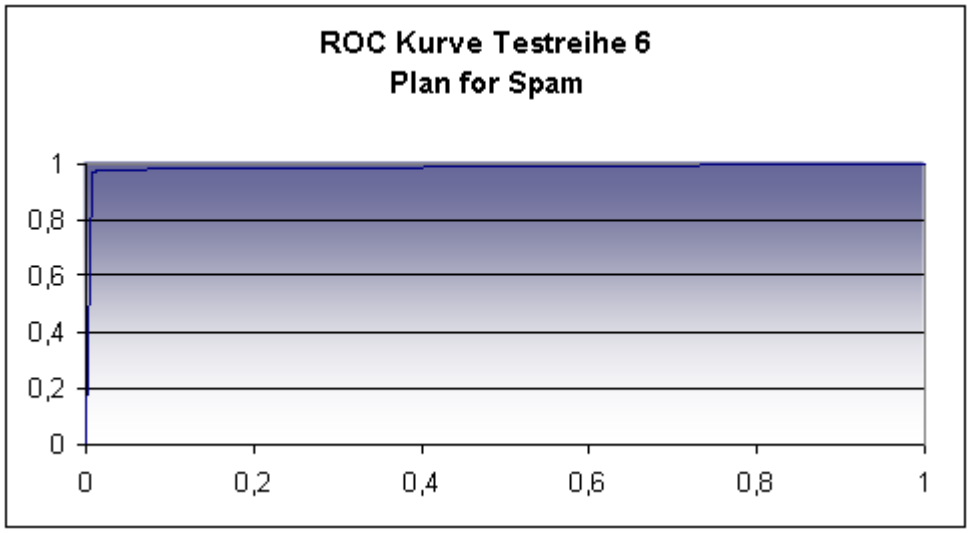

ROC Kurve Testreihe 5
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

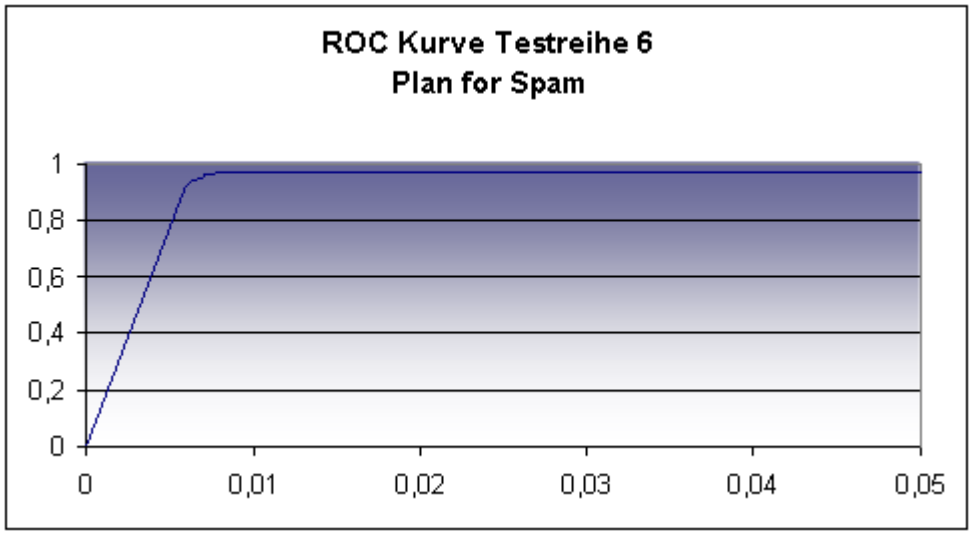

ROC Kurve Testreihe 5
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

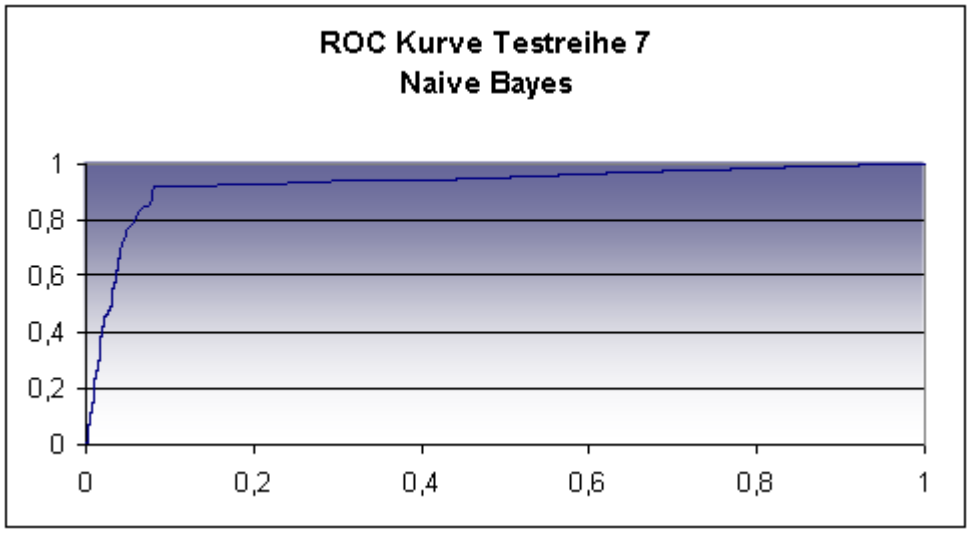

ROC Kurve Testreihe 5
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

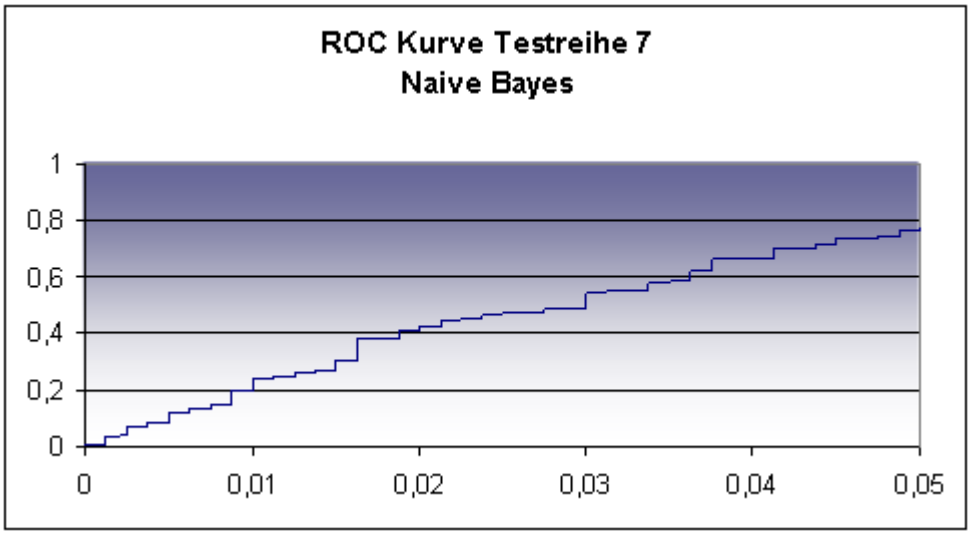

ROC Kurve Testreihe 6
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

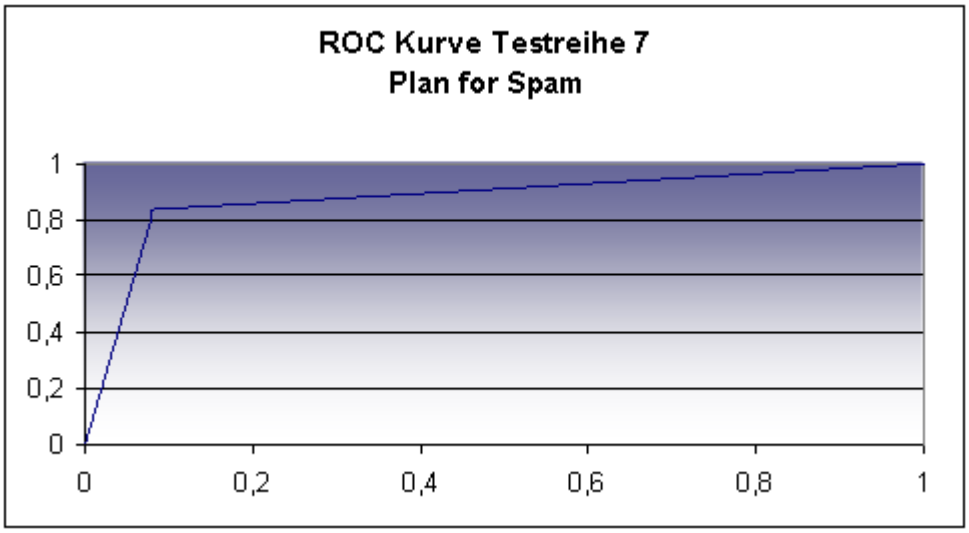

ROC Kurve Testreihe 6
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

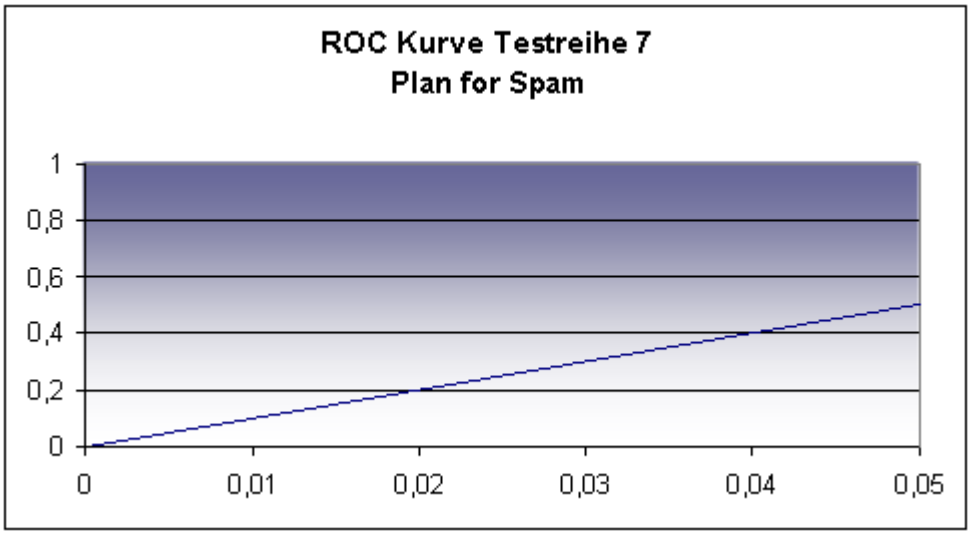

ROC Kurve Testreihe 6
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

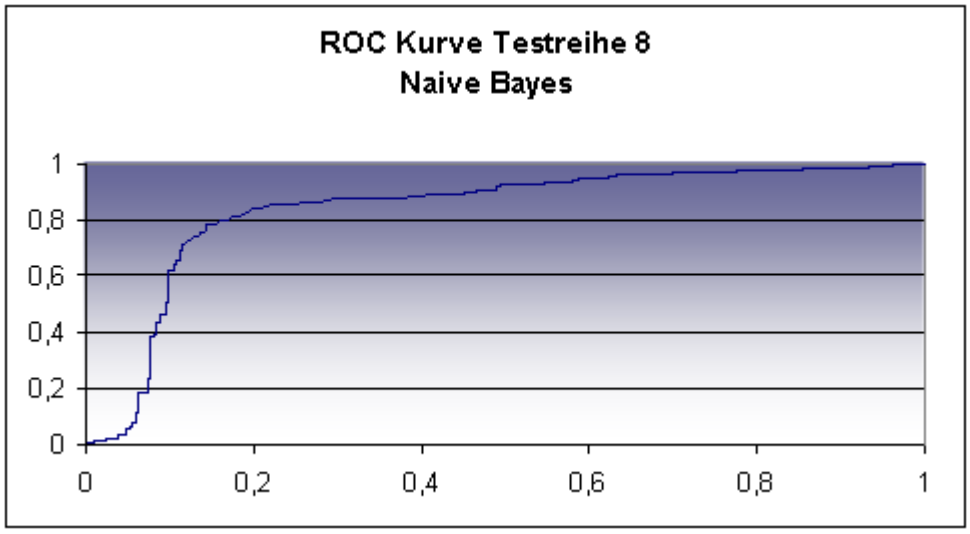

ROC Kurve Testreihe 6
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

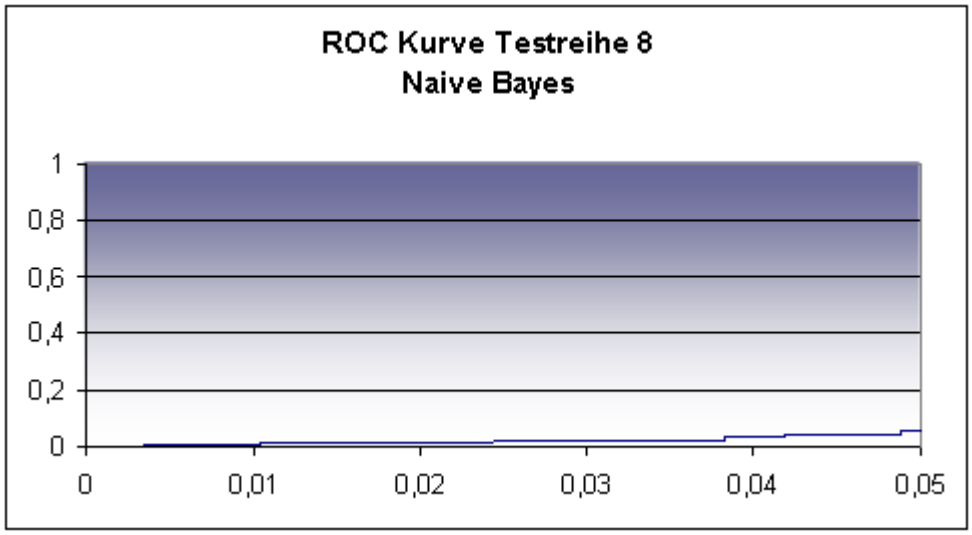

ROC Kurve Testreihe 7
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1

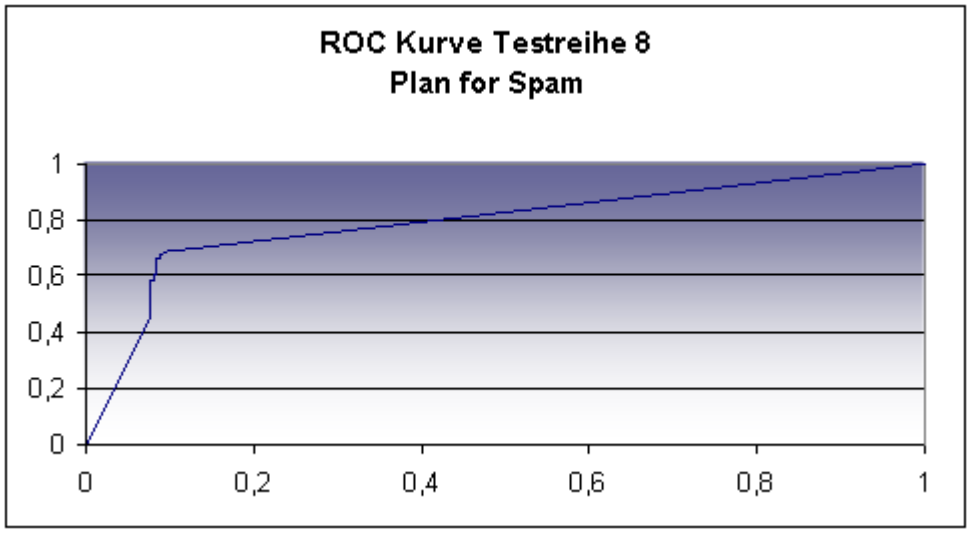

ROC Kurve Testreihe 7
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

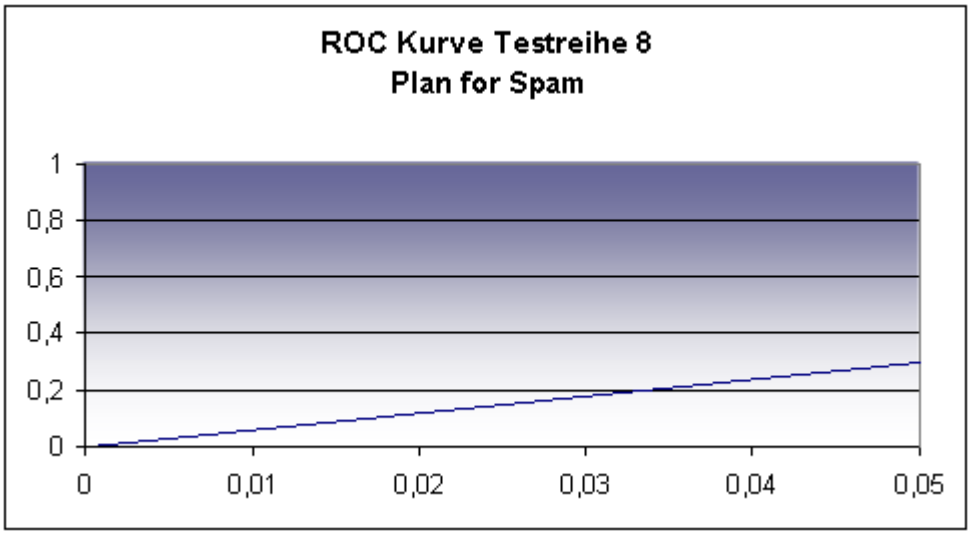

ROC Kurve Testreihe 7
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1


ROC Kurve Testreihe 7
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05


ROC Kurve Testreihe 8
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1


ROC Kurve Testreihe 8
Naive Bayes
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05


ROC Kurve Testreihe 8
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,2
0,4
0,6
0,8
1


ROC Kurve Testreihe 8
Plan for Spam
1
0,8
0,6
0,4
0,2
0
0
0,01
0,02
0,03
0,04
0,05

Anhang C

Quellcode

Die Methode EmptyTrash enthält die gesamte Funktionalität für die Testumgebung, also den Umgang mit den E-Mail-Verzeichnissen, das Training und den Aufruf der Validierung.

```
NS_IMETHODIMP
nsMessenger::EmptyTrash(nsIRDFCompositeDataSource* db,
                        nsIRDFResource* folderResource)
{
#define ANZAHL_SPAMMAILS 1329
#define ANZAHL_HAMMAILS 2009

#define ANZAHL_INTERVALLE 5
#define INTERVALL_JETZT 4

int spamstopper[100]; /* es ist moeglich nach einer bestimmten Anzahl von */
int hamstopper[100];  /* Trainingsmails zu stoppen                        */

hamstopper[0] = 50000;
spamstopper[0] = 50000;

int stopperanzahl = 0;
int stopperanzahl = ANZAHL_INTERVALLE - 1;
int i, intervall_spam = ANZAHL_SPAMMAILS / ANZAHL_INTERVALLE + 1;
int intervall_ham = ANZAHL_HAMMAILS / ANZAHL_INTERVALLE + 1;

/************************** DIPLOMARBEIT ***********************************/
printf("Testumgebung Kai Brodmann Diplomarbeit\n");
/************************** DIPLOMARBEIT ***********************************/

system("rm \"/root/.mozilla/default/akoy3ztx.slt/Mail/Local Folders/spam\"\n");
system("rm \"/root/.mozilla/default/akoy3ztx.slt/Mail/Local Folders/ham\"\n");

system("cp /usr/src/corpus/ham \"/root/.mozilla/default/akoy3ztx.slt/Mail/Local Folders
     /ham\"\n");
system("cp /usr/src/corpus/spam \"/root/.mozilla/default/akoy3ztx.slt/Mail/Local Folders
     /spam\"\n");

system("rm /usr/src/mozilla/suite/dist/bin/ergebnis.csv");
system("rm /usr/src/mozilla/suite/dist/bin/ergebnis2.csv");


for (i = 1; i <= ANZAHL_INTERVALLE; i++)
{
  hamstopper[i-1] = intervall_ham * i;
```

```
  spamstopper[i-1] = intervall_spam * i;
  printf("hamstopper[%d] = %d\n", i-1, hamstopper[i-1]);
  printf("spamstopper[%d] = %d\n", i-1, spamstopper[i-1]);
}

hamstopper[0] = hamstopper[INTERVALL_JETZT];
spamstopper[0] = spamstopper[INTERVALL_JETZT];

nsresult rv = NS_OK;

nsCOMPtr<nsIEnumerator> my_nsIEnumerator;

nsCOMPtr<nsISimpleEnumerator> my_nsISimpleEnumerator;

nsCOMPtr<nsISupports> my_aSupport;

nsCOMPtr<nsIMsgDatabase> my_nsIMsgDatabase;

nsCOMPtr<nsIMsgFolder> my_Folder;
nsCOMPtr<nsIMsgFolder> my_Folder2;

nsCOMPtr<nsIMsgIncomingServer> my_IncomingServer;

nsCOMPtr<nsIMsgAccountManager> my_AccountManager =
do_GetService(NS_MSGACCOUNTMANAGER_CONTRACTID, &rv);

nsCOMPtr<nsIJunkMailPlugin> my_junkPlugin;

nsCOMPtr<nsIMsgFilterPlugin> my_nsIMsgFilterPlugin;

rv = my_AccountManager->GetLocalFoldersServer(getter_AddRefs(my_IncomingServer));

rv = my_IncomingServer->GetSpamFilterPlugin(getter_AddRefs(my_nsIMsgFilterPlugin));

my_junkPlugin = do_QueryInterface(my_nsIMsgFilterPlugin);

my_junkPlugin->ResetTrainingData();

rv = my_IncomingServer->GetRootFolder(getter_AddRefs(my_Folder));

FILE *ausgabedatei;
char filnam[256];
char puffer[4096];

int schleifenzaehler;


for (schleifenzaehler = 0; schleifenzaehler <= stopperanzahl; schleifenzaehler ++)
{

  rv = my_Folder->GetSubFolders(getter_AddRefs(my_nsIEnumerator));


  rv = my_nsIEnumerator->First();

  int nachrichtenzaehler = 0;

  my_junkPlugin->ResetTrainingData();

  while (NS_SUCCEEDED(rv))
  {
    rv = my_nsIEnumerator->CurrentItem(getter_AddRefs(my_aSupport));

    my_Folder2 = do_QueryInterface(my_aSupport);
```

```
rv = my_Folder2->GetMsgDatabase(NULL, getter_AddRefs(my_nsIMsgDatabase));

nsXPIDLCString my_folderuri;
my_Folder2->GetURI(getter_Copies(my_folderuri));
printf ("Ordner = <%s>\n", my_folderuri.get());


if ((!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/spam"))
   || (!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/ham")))
{

  rv = my_nsIMsgDatabase->EnumerateMessages(getter_AddRefs(my_nsISimpleEnumerator));


  if (NS_SUCCEEDED(rv) && my_nsISimpleEnumerator)
  {
    PRBool hasMore;

    nachrichtenzaehler = 0;

    while (NS_SUCCEEDED(rv = my_nsISimpleEnumerator->HasMoreElements(&hasMore))
          && (hasMore == PR_TRUE))
    {
      if (!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/spam"))
      {
        if (spamstopper[schleifenzaehler] == nachrichtenzaehler)
        break;
      }
      else
      {
        if (hamstopper[schleifenzaehler] == nachrichtenzaehler)
        break;
      }

      nsCOMPtr <nsIMsgDBHdr> my_nsIMsgDBHdr;
      rv = my_nsISimpleEnumerator->GetNext(getter_AddRefs(my_nsIMsgDBHdr));

      if (my_nsIMsgDBHdr && NS_SUCCEEDED(rv))
      {
        nsXPIDLCString my_uri;
        my_Folder2->GetUriForMsg(my_nsIMsgDBHdr, getter_Copies(my_uri));
        if (!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/spam"))
        {
          my_junkPlugin->SetMessageClassification(my_uri.get(),
             nsIJunkMailPlugin::UNCLASSIFIED, nsIJunkMailPlugin::JUNK, nsnull, nsnull);
        }
        else
        {
          my_junkPlugin->SetMessageClassification(my_uri.get(),
             nsIJunkMailPlugin::UNCLASSIFIED, nsIJunkMailPlugin::GOOD, nsnull, nsnull);
        }

        nsXPIDLCString my_messageId;
        my_nsIMsgDBHdr->GetMessageId(getter_Copies(my_messageId));

        printf ("** Trainiere Nachricht Nr %d messageId=<%s> uri=<%s>\n",
           nachrichtenzaehler, my_messageId.get(), my_uri.get());
      } /* if */
      nachrichtenzaehler ++;
    } /* while */

  } /* if */
} /* if (!strcmp(my_folderuri.get */
```

```
    if ((!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/validierungspam"))
       || (!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/validierungham")))
    {
      rv = my_nsIMsgDatabase->EnumerateMessages(getter_AddRefs(my_nsISimpleEnumerator));
      nachrichtenzaehler = 1;

      if (NS_SUCCEEDED(rv) && my_nsISimpleEnumerator)
      {
        PRBool hasMore;

        while (NS_SUCCEEDED(rv = my_nsISimpleEnumerator->HasMoreElements(&hasMore))
      && (hasMore == PR_TRUE))
        {
          nsCOMPtr <nsIMsgDBHdr> my_nsIMsgDBHdr;
          rv = my_nsISimpleEnumerator->GetNext(getter_AddRefs(my_nsIMsgDBHdr));

          if (my_nsIMsgDBHdr && NS_SUCCEEDED(rv))
          {
            nsXPIDLCString my_uri;
            my_Folder2->GetUriForMsg(my_nsIMsgDBHdr, getter_Copies(my_uri));
            my_junkPlugin->ClassifyMessage(my_uri.get(), nsnull, nsnull);

            nsXPIDLCString my_junkScoreStr;
            my_nsIMsgDBHdr->GetStringProperty("junkscore", getter_Copies(my_junkScoreStr));

            nsXPIDLCString my_messageId;
            my_nsIMsgDBHdr->GetMessageId(getter_Copies(my_messageId));
            PRUint32 my_flags;
            my_nsIMsgDBHdr->GetFlags(&my_flags);

            sprintf(filnam, "/usr/src/mozilla/suite/dist/bin/ergebnis.csv");
            ausgabedatei = fopen(filnam, "a");

            printf ("\n\n** Nachricht messageId=<%s> uri=<%s> junkscore=<%s> Flags %d\n"
                   , my_messageId.get(), my_uri.get(), my_junkScoreStr.get() ,       my_flags);

            if (!strcmp(my_folderuri.get(), "mailbox://nobody@Local%20Folders/validierungspam"))
            {
              sprintf(puffer, "1\n");
              fputs(puffer, ausgabedatei);
            }
            else
            {
              sprintf(puffer, "0\n");
              fputs(puffer, ausgabedatei);
            }
            fclose(ausgabedatei);


          } /* if */
          nachrichtenzaehler ++;
        } /* while */
      } /* if */
    } /* if (!strcmp(my_folderuri.get */

    rv = my_nsIEnumerator->Next();
  } /* while */
}

/************************** DIPLOMARBEIT ***********************************/
```

```
rv = NS_ERROR_NULL_POINTER;

if (!db || !folderResource) return rv;
nsCOMPtr<nsISupportsArray> folderArray;

rv = NS_NewISupportsArray(getter_AddRefs(folderArray));
if (NS_FAILED(rv)) return rv;
folderArray->AppendElement(folderResource);
rv = DoCommand(db, NS_LITERAL_CSTRING(NC_RDF_EMPTYTRASH), folderArray, nsnull);
if (NS_SUCCEEDED(rv) && mTxnMgr)
mTxnMgr->Clear();
return rv;
}
```

Die Methode classifyMessage des Bayesian Filters wurde in einen Naive-Bayes-Filter umgewandelt. Die Original-Datei wurde zum Zwecke der Validierung des *Plan for Spams* beibehalten.

```
void nsBayesianFilter::classifyMessage(Tokenizer& tokenizer, const char* messageURI,
                                       nsIJunkMailClassificationListener* listener)
{
Token* tokens = tokenizer.copyTokens();
if (!tokens) return;

// the algorithm in "A Plan For Spam" assumes that you have a large good
// corpus and a large junk corpus.
// that won't be the case with users who first use the junk mail feature
// so, we do certain things to encourage them to train.
//
// if there are no good tokens, assume the message is junk
// this will "encourage" the user to train
// and if there are no bad tokens, assume the message is not junk
// this will also "encourage" the user to train
// see bug #194238

double anzahl_good_tokens = mGoodTokens.countTokens();
double anzahl_bad_tokens = mBadTokens.countTokens();

Token* t = mGoodTokens.get("daswortwasnirgendsvorkommt");
double wordpositionsg = ((t != NULL) ? t->mCount : 0);

Token* t2 = mBadTokens.get("daswortwasnirgendsvorkommt");
double wordpositionsb = ((t2 != NULL) ? t2->mCount : 0);

FILE *ausgabedatei;
char filnam[256];
char puffer[4096];

/* Diplom Arbeit */


/* Naive Bayes nach Table 6.2 */

PRUint32 i, count = tokenizer.countTokens();
double ngood = mGoodCount, nbad = mBadCount;
double wahrscheinlichkeit_spam = 1, wahrscheinlichkeit_ham = 1;

for (i = 0; i < count; ++i)
{
  Token& token = tokens[i];
  const char* word = token.mWord;

  Token* t = mGoodTokens.get(word);
  double g = ((t != NULL) ? t->mCount : 0);

  wahrscheinlichkeit_ham = wahrscheinlichkeit_ham +
     log(((g + 1)/(wordpositionsg+(anzahl_good_tokens + anzahl_bad_tokens))));
} /* for */


for (i = 0; i < count; ++i)
{
  Token& token = tokens[i];
  const char* word = token.mWord;

  Token* t = mBadTokens.get(word);
  double g = ((t != NULL) ? t->mCount : 0);
```

```
  wahrscheinlichkeit_spam = wahrscheinlichkeit_spam +
     log(((g + 1)/(wordpositionsb+(anzahl_good_tokens + anzahl_bad_tokens))));
} /* for */

double faktor = wahrscheinlichkeit_spam / wahrscheinlichkeit_ham + wahrscheinlichkeit_spam;

printf("HAM %f\n", wahrscheinlichkeit_ham);*/
printf("%f\n", faktor);


sprintf(filnam, "/usr/src/mozilla/suite/dist/bin/ergebnis2.csv");
ausgabedatei = fopen(filnam, "a");

sprintf(puffer, "%f\n", faktor);
fputs(puffer, ausgabedatei);

fclose(ausgabedatei);

PRBool isJunk;

if (faktor < 0.5)
{
  isJunk = 1;
}
else
{
  isJunk = 0;
}

if (listener)
listener->OnMessageClassified(messageURI, isJunk ?
   nsMsgJunkStatus(nsIJunkMailPlugin::JUNK) : nsMsgJunkStatus(nsIJunkMailPlugin::GOOD));

}
```

Mit den zwei folgenden Programm wurden dann die Dateien ausgewertet, die zuvor von dem von mir implementierten neuen Teil von Mozilla geschrieben worden waren.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>


/* wandelt eine Zeichenkette(Zahl) der Laenge len in ein long um */
/*****************************************************************/

long erw_strtol(char *zeichen, int len)
{
  char temp[128];
  temp[len] = 0;
  strncpy (temp, zeichen, len);
  return strtol(temp, NULL, 10);
} /* erw_strtol */

/* wandelt eine Zeichenkette(Zahl) der Laenge len in ein float um */
/*****************************************************************/

float erw_atof (char *zeichen, int len)
{
  char temp[128];
  temp[len] = 0;
  strncpy (temp, zeichen, len);
  return (float) atof(temp);
} /* erw_atof */

main()
{
FILE *fil1, *fil2;
char filnam[256];
char puffer[4096];
char puffer2[4096];
float liste[20000];
float liste2[20000];
float temp;
int zeilenzaehler = 0;
int schleifenzaehler = 0;
int schleifenzaehler2 = 0;
double richtige = 0;
int offset = 0;
double biswo;
int real_spam = 0;
float summe_precision = 0;
float schwelle_recall = 1;
float liste_alt = -1;

sprintf(filnam, "ergebnis.csv");
fil1 = fopen(filnam, "r");

sprintf(filnam, "ergebnis2.csv");
fil2 = fopen(filnam, "r");

zeilenzaehler = 0;

while (fgets(puffer, sizeof(puffer), fil1))
{
  liste[zeilenzaehler] = (float) atof(puffer);
```

```
  zeilenzaehler ++;
} /* while */

zeilenzaehler = 0;

while (fgets(puffer, sizeof(puffer), fil2))
{
  liste2[zeilenzaehler] = (float) atof(puffer);
  zeilenzaehler ++;
} /* while */

for (schleifenzaehler = 0; schleifenzaehler < zeilenzaehler; schleifenzaehler ++)
{
  for (schleifenzaehler2 = schleifenzaehler; schleifenzaehler2 < zeilenzaehler;
     schleifenzaehler2 ++)
  {
    if (liste2[schleifenzaehler] < liste2[schleifenzaehler2])
    {
      temp = liste[schleifenzaehler];
      liste[schleifenzaehler] = liste[schleifenzaehler2];
      liste[schleifenzaehler2] = temp;
      temp = liste2[schleifenzaehler];
      liste2[schleifenzaehler] = liste2[schleifenzaehler2];
      liste2[schleifenzaehler2] = temp;
    }
  }
}


for (biswo = schleifenzaehler; biswo >= 0; biswo = biswo - 1)
{

  int als_spam_klassifiziert = 0, spam_richtig_klassifiziert = 0;

  for (schleifenzaehler2 = 0; schleifenzaehler2 <= biswo; schleifenzaehler2++)
  {
    if ((liste[schleifenzaehler2] == 1) && (biswo == schleifenzaehler))
    {
      real_spam ++;
    }
    als_spam_klassifiziert ++;

    if (liste[schleifenzaehler2] == 1)
    spam_richtig_klassifiziert ++;
  }

  if  (
  ((float)spam_richtig_klassifiziert/(float)real_spam < schwelle_recall)
  ||
  (biswo == 0)
  )
  {
    schwelle_recall = schwelle_recall - 0.1;
    summe_precision = summe_precision +
       (float)spam_richtig_klassifiziert/(float)als_spam_klassifiziert;
  }

  printf("%f %f ", liste[schleifenzaehler2], liste2[schleifenzaehler2]);
  if ((liste2[schleifenzaehler2] != liste_alt) || (biswo == 0))
  printf("%f %f\n",  (float)spam_richtig_klassifiziert/(float)real_spam,
     (float)spam_richtig_klassifiziert/(float)als_spam_klassifiziert  );
  printf("\n");
  liste_alt = liste2[schleifenzaehler2];
}
```

```
printf("11-Point Average Precision %f\n", summe_precision/11);
printf("Anzahl Spam-Mails %d Ham-Mails %d\n", real_spam, schleifenzaehler-real_spam);


fclose(fil1);
fclose(fil2);
}




#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

/* wandelt eine Zeichenkette(Zahl) der Laenge len in ein long um */
/*****************************************************************/

long erw_strtol(char *zeichen, int len)
{
  char temp[128];
  temp[len] = 0;
  strncpy (temp, zeichen, len);
  return strtol(temp, NULL, 10);
} /* erw_strtol */

/* wandelt eine Zeichenkette(Zahl) der Laenge len in ein float um */
/*****************************************************************/

float erw_atof (char *zeichen, int len)
{
  char temp[128];
  temp[len] = 0;
  strncpy (temp, zeichen, len);
  return (float) atof(temp);
} /* erw_atof */

main()
{
FILE *fil1, *fil2;
char filnam[256];
char puffer[4096];
char puffer2[4096];
float liste[20000];
float liste2[20000];
float temp;
int zeilenzaehler = 0;
int schleifenzaehler = 0;
int schleifenzaehler2 = 0;
double TP = 0, FP = 0;
int offset = 0;
int biswo;
double AUC = 0;
double letzte_x = 1, letzte_y = 1;
int ANZAHL_SPAMMAILS = 0;
int ANZAHL_HAMMAILS = 0;
float liste_alt = -1;

sprintf(filnam, "ergebnis.csv");
fil1 = fopen(filnam, "r");

sprintf(filnam, "ergebnis2.csv");
```

```
fil2 = fopen(filnam, "r");

zeilenzaehler = 0;

while (fgets(puffer, sizeof(puffer), fil1))
{
  liste[zeilenzaehler] = (float) atof(puffer);
  zeilenzaehler ++;
} /* while */

zeilenzaehler = 0;

while (fgets(puffer, sizeof(puffer), fil2))
{
  liste2[zeilenzaehler] = (float) atof(puffer);
  zeilenzaehler ++;
} /* while */

for (schleifenzaehler = 0; schleifenzaehler < zeilenzaehler; schleifenzaehler ++)
{
  for (schleifenzaehler2 = schleifenzaehler; schleifenzaehler2 < zeilenzaehler;
     schleifenzaehler2 ++)
  {
    if (liste2[schleifenzaehler] < liste2[schleifenzaehler2])
    {
      temp = liste[schleifenzaehler];
      liste[schleifenzaehler] = liste[schleifenzaehler2];
      liste[schleifenzaehler2] = temp;
      temp = liste2[schleifenzaehler];
      liste2[schleifenzaehler] = liste2[schleifenzaehler2];
      liste2[schleifenzaehler2] = temp;
    }
  }
}


for (biswo = schleifenzaehler; biswo >= 0; biswo = biswo - 1)
{
  TP = 0;
  FP = 0;

  for (schleifenzaehler2 = 1; schleifenzaehler2 <= biswo; schleifenzaehler2 ++)
  {
    if
    (liste[schleifenzaehler2+offset-1] == 1)
    TP ++;
    if
    (liste[schleifenzaehler2+offset-1] == 0)
    FP ++;
  }

  if (biswo == schleifenzaehler)
  {
    ANZAHL_HAMMAILS = FP;
    ANZAHL_SPAMMAILS = TP;
  }

  printf("%f %f ", liste[biswo], liste2[biswo]);

  if ((liste_alt != liste2[biswo]) || (biswo == 0))
    printf("%f ", FP/ANZAHL_HAMMAILS);

  if (letzte_x != FP/ANZAHL_HAMMAILS)
    AUC = AUC + (letzte_x - FP/ANZAHL_HAMMAILS) * letzte_y;
```

```
  letzte_x = FP/ANZAHL_HAMMAILS;

  if ((liste_alt != liste2[biswo]) || (biswo == 0))
    printf("%f\n", TP/ANZAHL_SPAMMAILS);

  printf("\n");
  letzte_y = TP/ANZAHL_SPAMMAILS;

  liste_alt = liste2[biswo];
}

printf("AUC %f\n", AUC);
printf("Anzahl Ham-Mails %d Spam-Mails %d\n", ANZAHL_HAMMAILS, ANZAHL_SPAMMAILS);

fclose(fil1);
fclose(fil2);

}
```

Literaturverzeichnis

[1] Paul Graham, *A Plan for Spam*, 2002, http://www.paulgraham.com/spam.html

[2] Paul Graham, *Better Bayesian Filtering*, MIT Spam Conference 2003, http://www.paulgraham.com/better.html

[3] Tom M. Mitchell, Machine Learning, McGraw Hill, 1997, Seite 177-184

[4] William S. Yerazunis, Shalendra Chhabra, Christian Siefkes, Fidelis Assis, Dimitrios Gunopulos, *A Unified Model Of Spam Filtration*, MIT Spam Conference 2005, http://crm114.sourceforge.net/UnifiedFilters.pdf

[5] Aleksander Kolcz, Abdur Chowdhury, Joshua Alspector, *The Impact of Feature Selection on Signature-Driven Spam Detection*, CEAS 2004, http://www.ceas.cc/papers-2004/147.pdf

[6] Alan Gray, Mads Haahr, *Personalised, Collaborative Spam Filtering*, CEAS 2004, https://www.cs.tcd.ie/publications/tech-reports/reports.04/TCD-CS-2004-36.pdf

[7] Jennifer Golbeck, James Hendler, *Reputation Network Analysis for E-Mail Filtering*, CEAS 2004, http://www.ceas.cc/papers-2004/177.pdf

[8] Margaret Fleck, David Forsyth, Chris Bregler, *Finding Naked People*, 4th European Conference on Computer Vision, 1996, http://www.cs.hmc.edu/~fleck/naked-people.ps.gz

[9] Cynthia Dwork, Moni Naor, *Pricing via Processing or Combatting Junk Mail*, 12th Annual International Cryptology Conference, 1992, http://research.microsoft.com/research/sv/PennyBlack/junk1.pdf

[10] Cynthia Dwork, Moni Naor, Andrew Goldberg *On Memory-Bound Functions for Fighting Spam*, Proc. Crypto 2003, http://www.wisdom.weizmann.ac.il/\%7Enaor/PAPERS/mem.ps

[11] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, Constantine D. Spyropoulos,*An Evaluation of Naive Bayesian Anti-Spam Filtering*, 11th European Conference on Machine Learning (ECML), 2000, http://www.aueb.gr/users/ion/docs/mlnet_paper.pdf

[12] Ravi Kiran S S, Indriyati Atmosukarto, *Spam or Not Spam, That is the question*, 2004, www.cs.washington.edu/homes/indria/research/spamfilter_ravi_indri.pdf

[13] David Lewis, *The Independence Assumption in Information Retrieval*, ECML 1998, http://citeseer.ist.psu.edu/lewis98naive.html

[14] Tom Fawcett, *„In vivo“ spam filtering: A challenge problem for KDD*, SIGKDD Explorations 5(2): 140-148, 2003, http://home.comcast.net/~tom.fawcett/public_html/papers/spam-KDDexp.pdf

[15] Fabrizio Sebastiani, *Machine Learning in Automated Text Categorization*, ACM Computing Surveys, 1999, http://www.math.unipd.it/~fabseb60/Publications/ACMCS02.pdf

[16] Ralph Sontag, *Hat Bayes eine Chance?*, Tagungsbericht Netz- und Service-Infrastrukturen, TU Chemnitz, 2004, http://archiv.tu-chemnitz.de/pub/2004/0055/index.html

[17] William S. Yerazunis, *The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It*, 2004 MIT Spam Conference, http://crm114.sourceforge.net/Plateau99.pdf

[18] Moni Naor, *Verification of a human in the loop or Identification via the Turing Test*, 1996, http://www.wisdom.weizmann.ac.il/\~naor/PAPERS/human_abs.html

[19] Richard Clayton, *Stopping Spam by Extrusion Detection*, CEAS 2004, http://www.ceas.cc/papers-2004/172.pdf

[20] Bryan Klimt, Yiming Yang, *Introducing the Enron Corpus*, CEAS 2004, http://www.ceas.cc/papers-2004/168.pdf

[21] Jonathan A. Zdziarski, *Bayesian Noise Reduction: Progressive Noise Logic for Statistical Language Analysis*, MIT Spam Conference 2005, http://spamconference.org/abstracts.html

[22] Jonathan Oliver, *Using Lexigraphical Distancing to Block Spam*, MIT Spam Conference 2005, http://spamconference.org/abstracts.html

[23] Isidore Rigoutsos, *Chung-Kwei: a Pattern-discovery-based System for the Automatic Identification of Unsolicited E-Mail Messages*, CEAS 2004, http://www.research.ibm.com/spam/papers/chung-kwei.pdf

[24] Abdur Chowdhury, *Duplicate Data Detection*, Search&Navigation Group America Online, 2004, http://ir.iit.edu/~abdur/Research/Duplicate.html

[25] Barry Leiba, Nathaniel Borenstein, *A Multifaceted Approach to Spam Reduction*, CEAS 2004, www.ceas.cc/papers-2004/127.pdf

[26] Richard Segal, Jason Crawford, Jeffrey Kephart, Barry Leiba, *SpamGuru: An Enterprise Anti-Spam Filtering System*, CEAS 2004, www.ceas.cc/papers-2004/126.pdf

[27] Shabbir Ahmed, Farzana Mithun, *Word Stemming to Enhance Spam Filtering*, CEAS 2004, http://www.ceas.cc/papers-2004/167.pdf

[28] Flavio D. Garcia, Jaap-Henk Hoepman, Jeroen van Nieuwenhuizen, *Spam Filter Analysis*, Proceedings of 19th IFIP International Information Security Conference, WCC2004-SEC, 2004, http://arxiv.org/abs/cs.CR/0402046

[29] Gordon Rios, Hongyuan Zha, *Exploring Support Vector Machines and Random Forests for Spam Detection*, CEAS 2004, http://www.ceas.cc/papers-2004/174.pdf

[30] Eirinaios Michelakis, Ion Androutsopoulos, Georgios Paliouras, George Sakkis, Panagiotis Stamatopoulos, *Filtron: A Learning-Based Anti-Spam Filter*, CEAS 2004, http://www.ceas.cc/papers-2004/142.pdf

[31] Gregory L. Wittel, S. Felix Wu, *On Attacking Statistical Spam Filters*, CEAS 2004, http://www.ceas.cc/papers-2004/170.pdf

[32] T.A Meyer, B.Whateley, *SpamBayes: Effective open-source, Bayesian based, E-Mail classification system*, CEAS 2004, http://www.ceas.cc/papers-2004/136.pdf

[33] Geoff Hulten, Anthony Penta, Gopalakrishnan Seshadrinathan, Manav Mishra, *Trends in Spam Products and Methods*, CEAS 2004, http://www.ceas.cc/papers-2004/165.pdf

[34] Gerhard Widmer, Miroslav Kubat, *Learning in the Presence of Concept Drift and Hidden Contexts*, Machine Learning Journal, 1996, Seite 69-101, http://citeseer.ist.psu.edu/widmer96learning.html

[35] *Projekt Honeypot*, http://www.projecthoneypot.org

[36] Loebner Prize Homepage, http://www.loebner.net/Prizef/loebner-prize.html

[37] GNU Projekt, http://www.gnu.org/

[38] Definitionen und Beschreibungen des bayrischen Rundfunks, http://www.br-online.de/wissen-bildung/thema/virenschutz/spam.xml

[39] The Apache Software Foundation, http://www.apache.org/

[40] Open Source SQL Datenbank MySQL, http://www.mysql.de/

[41] Open Source Leitfaden für kleine und mittlere Unternehmen, http://oss-broschuere.berlios.de/broschuere/broschuere-de.html/

[42] BerliOS, der Open Source Mediator, http://www.berlios.de/

[43] Wikipedia, die freie Enzyklopädie, http://www.wikipedia.de/

[44] Benutzbarkeitsstudie Spamfilter, http://www.go4emarketing.de/seiten/E-Mail/e37.html/

[45] Spamassassin E-Mail Corpus,http://spamassassin.apache.org/publiccorpus/

[46] Mirror Annexia Spam-Corpus,http://darleyconsulting.com/www.annexia.org/spam/

[47] Annexia Spam-Corpus, http://www.annexia.org/spam/

[48] Spam-Corpora, http://nexp.cs.pdx.edu/psam-archives/corpora/

[49] Ling Spam-Corpus, http://www.aueb.gr/users/ion/publications.html

[50] Spam Archiv, http://www.spamarchive.org/

[51] Paul Wouters's personal spam statistics 1997-2004, http://www.xtdnet.nl/paul/spam/

[52] Spam Archiv, http://www.em.ca/~bruceg/spam/

[53] Spam Archiv, http://bloodgate.com/spams/

[54] Spam Statistik, http://www.raingod.com/angus/Computing/Internet/Spam/Statistics/index.html

[55] Antispam Organisation, http://www.spamcop.net/

[56] Spam Statistik, http://www.halverson.org/spam/

[57] Japanisches Spam Archiv, http://www.nurs.or.jp/~ogochan/exdoc.cgi?view\&oid=spam/index.doc

[58] Spam Statistik, http://blogs.msdn.com/oldnewthing/archive/2004/09/16/230388.aspx

[59] Spam Linksammlung, http://spamlinks.net/archives.htm

[60] Hilfsprogramm zum Erstellen von Testmailcorpora, http://iit.demokritos.gr/skel/i-config/downloads/MailboxEncoder.html

[61] Hilfe zur Konvertierung ins MBOX-Format, http://dbacl.sourceforge.net/testsuite-2.html

[62] MIT Spam Konferenz 2005 , http://spamconference.org/

[63] *First/Second Conference on E-Mail and Anti-Spam* 2004 /2005 (CEAS), http://www.ceas.cc/

[64] Text REtrieval Conference (TREC), http://trec.nist.gov/

[65] Anstehende Spam Veranstaltungen, http://www.isipp.com/events.php

[66] Anstehende Spam Veranstaltungen, http://spamlinks.net/conf.htm

[67] *The Spammers Compendium*, http://www.jgc.org/tsc/

[68] Firma die sich unter anderem mit Softwareergonomie beschäftigt, http://www.go4emarketing.de/

[69] Jörg Auf dem Hövel, *Die digitale Plage und der Frust mit der E-Mail*, FAZ, 12. Juli 2005

[70] Hristomir Stankov, *Verlinkt und zugemüllt*, FAZ, 2. Dezember 2004

[71] Gordon Cormack, Tom Lynam, *Spam Track Guidelines*, http://plg.uwaterloo.ca/~gvcormac/spam/

[72] Homepage Gordon Cormack, University of Waterloo, http://plg.uwaterloo.ca/~gvcormac/