Implementierung einer Browser-Erweiterung zur Visualisierung und Erkennung von privatsphärenkritischen Nachrichten in sozialen Netzwerken

Implementation of a Browser-Extension to Detect and Visualize Privacy Critical Messages in Social Networks

Bachelor-Thesis von Bennet Jeutter

April 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Sicherheit in der Informationstechnik

CASED

EC SPRIDE

Implementierung einer Browser-Erweiterung zur Visualisierung und Erkennung von privatsphären-kritischen Nachrichten in sozialen Netzwerken
Implementation of a Browser-Extension to Detect and Visualize Privacy Critical Messages in Social Networks

Vorgelegte Bachelor-Thesis von Bennet Jeutter

Prüfer: Prof. Dr. Michael Waidner
Betreuer: Marco Ghiglieri, M.Sc., Martin Stopczynski, M.Sc.

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 08. April 2013

(Bennet Jeutter)

Zusammenfassung

Die sozialen Netzwerke wie etwa Facebook und Google+ gehören mittlerweile für viele zum Alltag und werden von den verschiedensten Altersgruppen genutzt. Die Beliebtheit dieser sozialen Netzwerke nimmt immer weiter zu und die Benutzerzahlen steigen kontinuierlich. Eine wesentliche Funktion für die Benutzer der sozialen Netzwerke ist die Kommunikation und der Informationsaustausch mit anderen Benutzern, dabei kann es sich um Freunde, Arbeitskollegen oder sogar Unbekannte handeln. Das große Spektrum an Personen, die in sozialen Netzwerken erreicht werden können, birgt Gefahren hinsichtlich der *Privacy* für die Benutzer. Diese veröffentlichen oftmals persönliche Informationen, ohne sich über Reichweite und Konsequenzen bewusst zu sein. Konsequenzen für die Benutzer können zum Beispiel geschädigte Beziehungen oder rechtliche Folgen sein, aber auch Identitätsdiebstahl oder Stalking. Dabei verlieren sie die Kontrolle über diese Informationen und es entstehen oftmals unerwünschte Effekte. Zum Beispiel ist es durch falsche Einstellungen möglich, dass Personen diese Informationen sehen, für die sie nicht bestimmt sind. Diese Personen könnten diese Information falsch interpretieren, aber auch missbrauchen. Zudem wird von den meisten Benutzern nicht darüber nachgedacht, inwieweit die Inhalte kritisch bezüglich der Privatsphäre sein könnten und werden oftmals im Nachhinein bereut.

Diese Arbeit beschäftigt sich mit dieser Problematik und stellt ein Konzept sowie eine Implementierung für eine Browser-Erweiterung am Beispiel Facebook vor, die das Bewusstsein für solch kritische Inhalte erhöhen soll. Die entwickelte Browser-Erweiterung überprüft Statusmeldungen des Benutzers während der Eingabe in Facebook und visualisiert kritische Textstellen. Diese Erweiterung soll dem Benutzer helfen, kritische Inhalte zu erkennen und erst gar nicht zu veröffentlichen. Dadurch sollen negative Auswirkungen auf den Benutzer verhindert werden.

Die Umsetzung dieses Konzeptes wird im späteren Verlauf der Arbeit beschrieben und diskutiert. Zudem werden existierende Ansätze vorgestellt und die Notwendigkeit, der in dieser Arbeit entwickelten Browser-Erweiterung, erklärt. Abschließend wird ein Fazit gezogen und mögliche zukünftige Erweiterungen für diese Arbeit aufgelistet.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel erläutert zunächst ein Problem heutiger sozialer Netzwerke, indem vom Benutzer unbewusst für ihn persönlich schädliche Inhalte veröffentlicht werden und er keine Kenntnis darüber hat, welche Konsequenzen das nach sich ziehen kann. Anschließend wird die Motivation für diese Arbeit beschrieben, also das angestrebte Ziel, um das Problem zu bewältigen und welche Verbesserungen sich dadurch für den Benutzer ergeben können. Im letzten Abschnitt wird dann der Aufbau der Arbeit präsentiert.

1.1 Problemstellung

Soziale Netzwerke wie Facebook und Google+ gehören heute für viele zum täglichen Leben, so nutzen etwa zwei Drittel der deutschen Internetbenutzer soziale Netzwerke [11]. Mehr als 584 Millionen Menschen sind täglich auf Facebook aktiv [15] und veröffentlichen private Inhalte, wie zum Beispiel Statusmeldungen, Fotos und Videos. Diese können öffentlich, also für jeden Nutzer, zugänglich gemacht werden. Es ist zwar möglich die sogenannten Privatsphäre-Einstellungen anzupassen, wonach zum Beispiel nur bestimmte Freunde die Information sehen können, jedoch fehlt oftmals die Kenntnis, um diese Einstellungen richtig zu verwenden. Hierbei werden die Einstellungsmöglichkeiten von vielen Benutzern als undurchsichtig und zu komplex beschrieben, insbesondere bei älteren Generationen [2]. Dadurch kam es zum Beispiel schon des Öfteren zu sogenannten Facebook-Partys, die sogar mediale Aufmerksamkeit bekamen. Dabei waren vermeintlich private Veranstaltungen plötzlich auf Facebook öffentlich zu finden und mehrere Hundert Gäste kamen, anstatt nur die eingeladenen Personen [22].

Junge Benutzer kennen diese Einstellungen zwar häufiger, aber scheinen jedoch weniger über die Inhalte nachzudenken, die sie in den sozialen Netzwerken veröffentlichen. Die Studie von Giovani und Pashley [12] zeigt, dass viele Benutzer zwar wissen, dass die Veröffentlichung ihrer Daten auf Facebook Konsequenzen haben kann, aber empfinden das nicht als kritisch. Konsequenzen können unter anderem geschädigte Beziehungen, Jobverlust und rechtliche Folgen sein, aber auch Identitätsdiebstahl oder Stalking [35]. Oftmals fehlt das Bewusstsein dafür, wie einfach es für Stalker, Arbeitgeber oder Eltern ist, Informationen über die betreffenden Personen zu beschaffen [12].

Es passiert immer wieder, dass Personen solche Konsequenzen erleben und ihre Veröffentlichungen in sozialen Netzwerken bereuen. Die Inhalte dieser Veröffentlichungen sind unterschiedlich. Oft werden Mitteilungen verfasst, die sensible Informationen enthalten. Sensible Themen dieser Veröffentlichungen sind häufig Alkohol und Drogenmissbrauch, Sex, Religion und Politik, persönliche und familiäre Probleme und Arbeit / Arbeitsstelle. Oftmals werden aber ebenso Veröffentlichungen mit emotionalem Inhalt getätigt, in denen andere Leute beleidigt oder angegriffen werden, aber auch ganze Streitigkeiten im Internet ausgefochten. Die Gründe für die Veröffentlichungen sind vielfältig, häufig, weil es vermeintlich ”cool” oder lustig ist, aber auch um Frustrationen in stark emotionalen Zuständen abzulassen. Die Reue entsteht in der Regel, weil die Informationen an die falsche Zielgruppe gelangen oder, weil das Konzept des Weiterteilens in sozialen Netzwerken nicht verstanden wurde [35].

Ein weiteres Problem ist die Kommerzialisierung der eigenen Daten auf sozialen Netzwerken. Dem Benutzer wird zwar suggeriert, die Nutzung sei kostenlos, allerdings ”bezahlt” dieser mit seinen Daten. Ein Kerngeschäft von sozialen Netzwerken ist die Nutzung der benutzergenerierten Daten, um zum Beispiel Werbung zu personalisieren. Darüber hinaus ist in den Datenschutzbestimmungen nicht ausgeschlossen, dass Informationen an Dritte verkauft werden. Wie die Daten also genau verarbeitet werden, bleibt für den Benutzer intransparent [1].

1.2 Motivation

Die oben genannten Probleme zeigen, dass sehr viele Benutzer in sozialen Netzwerken kaum ein Bewusstsein dafür haben, welche Informationen sie teilen und welche Auswirkungen dies haben kann. Um diese Auswirkungen zu verringern und den Missbrauch der Informationen zu verhindern, besteht die Notwendigkeit, ein unterstützendes Werkzeug für den Benutzer zu entwickeln, das ihn auf kritische Inhalte aufmerksam macht. Es existieren zwar Ansätze, die zum Beispiel die Facebook-Grundeinstellungen untersuchen und optimieren, allerdings beschäftigt sich bisher keine Arbeit mit den konkreten Inhalten von veröffentlichten Texten in sozialen Netzwerken.

Am Beispiel einer Browser-Erweiterung für den Browser Google Chrome und das soziale Netzwerk Facebook wird ein solches Werkzeug entwickelt - der Facebook Post Checker (FPC). Die Entscheidung für Google Chrome geht aus aktuellen Nutzungsstatistiken [38] hervor, da dieser am meisten Verwendung findet. Durch die zeitliche Einschränkung der Bachelorarbeit wurde die Proof-of-Concept Implementierung für das derzeit am stärksten verbreitete sozialen Netzwerk Facebook [11, 15] umgesetzt. Die Browser-Erweiterung ermöglicht es dem Benutzer, seine Statusmeldungen beim Eingeben automatisch auf kritische Inhalte überprüfen und die Ergebnisse auswerten zu lassen. Eine Statusmeldung ist ein

(in der Regel kurzer) Text, um anderen Benutzern eigenen Gedanken, Meinungen oder wichtige Informationen bereitzustellen. Durch verschiedene visuelle Rückmeldungen in Echtzeit, wie zum Beispiel Textfärbungen, soll das Bewusstsein des Benutzers bezüglich kritischer Inhalte gesteigert werden und somit Veröffentlichungen verhindern, die später bereut werden. Zudem soll der Inhalt dem Betreiber weniger Aufschluss über den Benutzer geben, indem private Informationen, die kommerzialisiert werden können, gekennzeichnet und vom Benutzer gegebenenfalls entfernt werden. Am Lehrstuhl wurde zu dieser Arbeit eine Studie [7] von Sebastian Funke durchgeführt, die gezeigt hat, dass privatsphärenkritische Beiträge durch solch ein Werkzeug reduziert werden können.

1.3 Aufbau der Arbeit

Zunächst werden in Kapitel 2 bisherige Ansätze vorgestellt, die dem Benutzer eine höhere Kontrolle über seine Daten in sozialen Netzwerken ermöglichen sollen. Am Ende dieses Kapitels wird mit einer Abgrenzung zu diesen untersuchten Arbeiten abgeschlossen. In Kapitel 3 werden die theoretischen Grundlagen der *Privacy* für diese Arbeit vermittelt und die angewandten Techniken, Levenshtein und Naive Bayes, erklärt. Anschließend folgt das Kapitel 4 mit dem Konzept des ”Facebook Post Checker”, ein in dieser Arbeit entstandenes Werkzeug zur Überprüfung von Statusmeldungen in Facebook. Die Implementierung dieses Konzepts, anhand einer Browser-Erweiterung für Google Chrome, wird anschließend in Kapitel 5 detailliert beschrieben. Die Ergebnisse der Evaluierung dieser Implementierung werden in Kapitel 6 präsentiert. Abschließend wird in Kapitel 7 ein Fazit zur Arbeit gegeben und noch ungelöste Probleme und zukünftige Aufgabenstellungen vorgestellt, die sich aus dieser Arbeit ergeben haben.

2 Verwandte Arbeiten

Eine Übersicht interessanter verwandter Arbeiten, die sich auch mit der Lösung des in Kapitel 1 vorgestellten Problems beschäftigen, wird in diesem Kapitel gezeigt. Zunächst wird eine wissenschaftliche Arbeit (Lockr [28]) präsentiert und anschließend werden drei existierende Ansätze vorgestellt. Am Ende dieses Kapitels wird ein Fazit zu den existierenden Ansätzen gezogen und eine Abgrenzung zu dieser Arbeit beschrieben.

Dieses Kapitel und die Folgenden enthalten oft den Begriff ”teilen”. Hiermit ist das Bereitstellen von Informationen in sozialen Netzwerken gemeint. Es bedeutet nicht unbedingt, dass die Information direkt öffentlich (für alle) sichtbar ist, sondern kann sich auch auf eine bestimmte Benutzergruppe (zum Beispiel Freunde) beschränken.

2.1 Lockr: Better Privacy for Social Networks

In der wissenschaftlichen Arbeit von A. Tootoonchian und S. Saroiu [28] wird ein System beschrieben, dass die Privatsphäre unter anderem mittels Kryptografie in Online-Plattformen verbessern soll, auf denen Informationen geteilt werden können. Kryptografische Maßnahmen sind digital signierte Freundschaften und eine Verschlüsselung für den Datenaustausch zwischen zwei Freunden. Die digitalen Signaturen sollen dazu dienen, dass sich Freunde, für die Inhalte bereitgestellt werden, auf dem entsprechenden Netzwerk nicht anmelden müssen, sondern mit der Signatur beweisen können, dass sie Freunde sind. Die Verschlüsselung hat den Vorteil, dass wirklich nur die Personen / Parteien die Informationen sehen können, die einen entsprechenden Schlüssel besitzen. Zudem trennt es den durch den Benutzer generierten Inhalt von den restlichen Funktionalitäten in einem sozialen Netzwerk (SN). Dadurch soll sich die Kontrolle der Benutzer über die Daten erhöhen, indem sie spezifizieren können, in welchen SNs diese Informationen gespeichert werden und wer dort Zugriff haben soll.

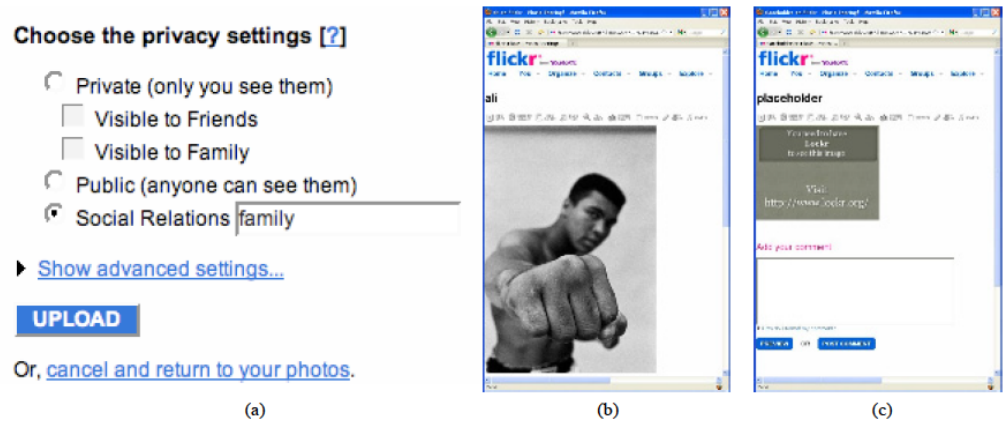

Abbildung 2.1: Screenshot einer Testimplementierung von Lockr in Flickr. (a) Ein Benutzer erstellt eine Lockr-Access-Control-List (b) Ein Benutzer schaut sich ein Bild mit korrekter Bescheinigung für digitale Freundschaft an (c) Ein Benutzer ohne korrekte Bescheinigung sieht nur einen Platzhalter [28]

Das entwickelte Konzept wurde in verschiedenen OSNs testweise implementiert (alte Flickr Implementierung auf Abbildung 2.1 zu sehen), ist aber nicht mehr verfügbar.

2.2 Reclaim Privacy

Reclam Privacy[1] ist ein Tool um die eigenen Privatsphären-Einstellungen (Reichweite) in Facebook zu überprüfen. Es wird als Lesezeichen in der Browserleiste gespeichert und kann dann in Facebook ausgeführt werden. Dieses Tool zeigt nach dem Ausführen eine Auswertung der Einstellungen (siehe Abbildung 2.2) und bietet eine Möglichkeit an, diese automatisch anzupassen. Zum Beispiel ist der dritte Eintrag mit einem *”caution”* versehen, da die Kontaktinformationen des Benutzers öffentlich sichtbar sind.

Dem Benutzer soll dadurch Arbeit abgenommen werden, da er sich nicht durch die zum Teil unübersichtlichen Facebook-Seiten klicken muss.

[1] `http://www.reclaimprivacy.org`, abgerufen am 21.02.2013

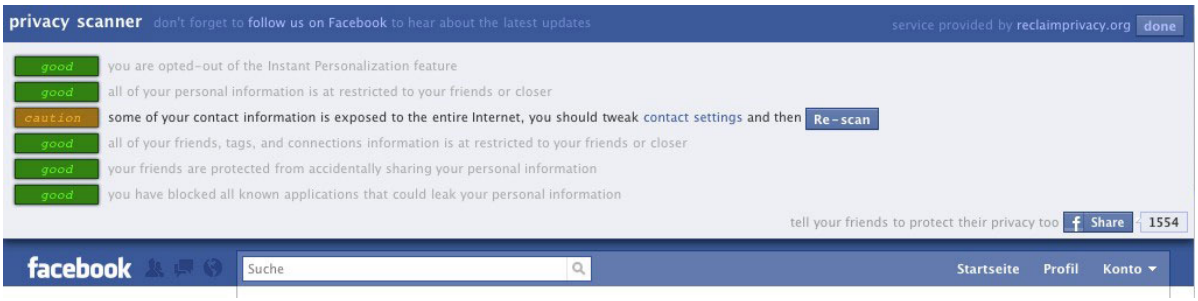

Abbildung 2.2: Screenshot der Überprüfungsergebnisse von Reclam Privacy [16]

Aktuell (Stand 2013) funktioniert das Tool nicht mehr, da Facebook Änderungen an der HTML-Struktur vorgenommen hat, sodass das Werkzeug die entsprechenden Informationen von Facebook nicht auslesen kann.

2.3 Privacyfix

Privacyfix[2] ist eine Erweiterung für den Browser Google Chrome. Diese überprüft, ähnlich wie Reclam Privacy, die Einstellungen von Facebook, aber auch Google+ wird unterstützt. Dabei werden allerdings nicht nur die Privatsphäre-Einstellungen überprüft, sondern ebenfalls Einstellungen für zum Beispiel Google-Indizierung des eigenen Profils. Zur Verbesserung der Einstellungen werden dann ”Fix”-Buttons angeboten (siehe Abbildung 2.3). Die Abbildung zeigt einige hinterlegte Bereiche, diese sind auf der linken Seite zum Teil orange. Das bedeutet: hier sind die Einstellungen nicht optimal für die eigene *Privacy* und sollten angepasst werden. Beim Klicken eines solchen ”Fix”-Buttons wird die entsprechende Einstellungsseite auf Facebook geöffnet und eine ausführliche Beschreibung zur entsprechenden Einstellung eingeblendet sowie eine Erläuterung, wieso Privacyfix diese ändern möchte. Diese Beschreibung geht über die von Facebook angegebene hinaus. Anschließend wird die Einstellung angepasst, sofern der Benutzer einverstanden ist. Auf der linken Seite sieht man das Ergebnis, nachdem man sein Profil hat ”fixen” lasen. Die orangenen Bereiche sind nun grün hinterlegt und kennzeichnen so, dass die Einstellungen jetzt in Ordnung sind. Der dritte Punkt in der Abbildung bedeutet zum Beispiel, dass alle Statusmeldungen, die verfasst werden, für **jeden** sichtbar sind. Die entsprechende Einstellung wurde durch das ”fixen” auf Freunde beschränkt. Neben dieser Funktionalität ist es möglich, dass Benachrichtigungen für Änderungen der Datenschutzerklärung von Facebook oder Google eingestellt werden können.

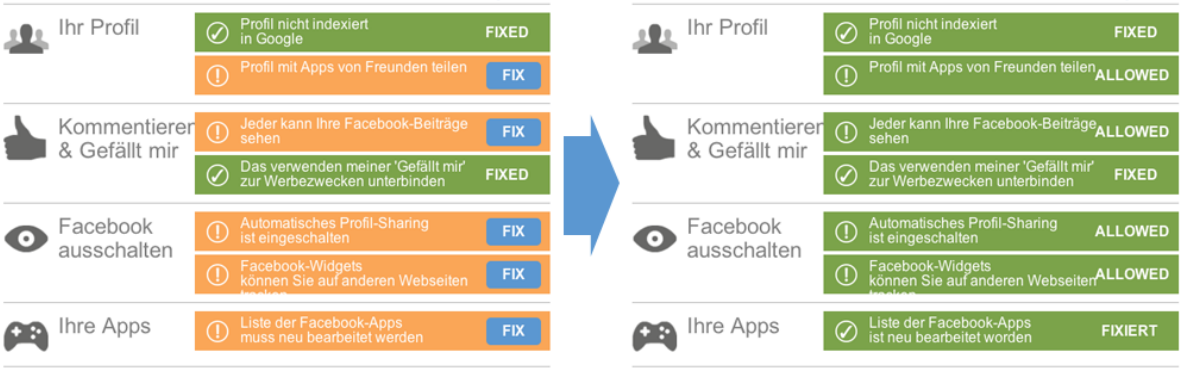

Abbildung 2.3: Screenshot der Überprüfung eines Facebook Profils durch Privacyfix und die anschließende Verbesserung

Zudem bietet die Erweiterung eine Möglichkeit an, dem Benutzer auszurechnen, welchen finanziellen Wert (in Dollar) das eigene Profil für verschiedene OSNs hat. So soll der Benutzer ein Bewusstsein dafür bekommen, dass die Nutzung der sozialen Netzwerke nicht wirklich kostenlos ist. Allerdings ist nicht offengelegt, wie dieser Wert berechnet wird.

2.4 Facebook Privacy Watcher

Der Facebook Privacy Wacher (FPW) [23, 24] ist eine Erweiterung für den Browser Firefox von Mozilla[3]. Auch diese Erweiterung dient der Anpassung von Privatsphäre-Einstellungen in Facebook. Allerdings ist der Unterschied zu den

[2] `www.privacyfix.com`, abgerufen am 21.02.2013

[3] `http://www.mozilla.org/`, aufgerufen am 21.02.2013

anderen Ansätzen, dass die Elemente in Facebook während des Benutzens bezüglich ihrer gruppenbezogenen Sichtbarkeit (Privatsphäre-Einstellungen) gefärbt und dem Nutzer so sichtbar gemacht werden (siehe Abbildung 2.4).

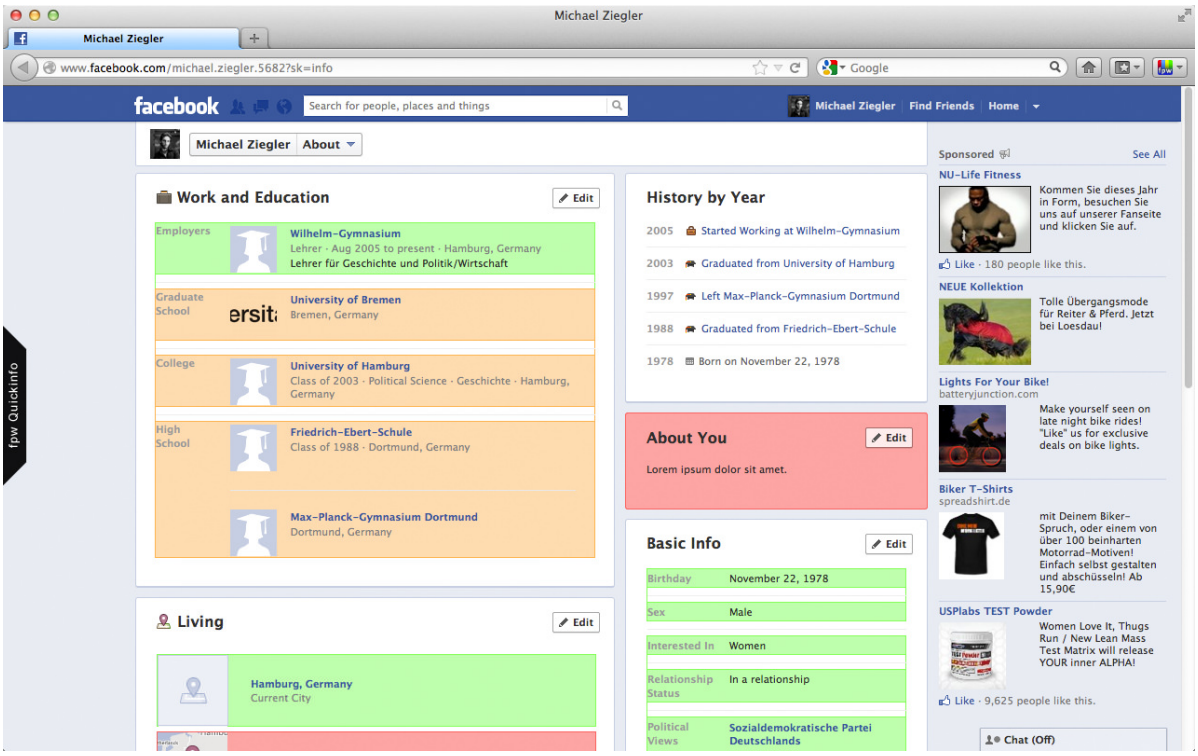

Abbildung 2.4: Screenshot der eingebetteten Färbung durch den Facebook Privacy Wacher [24]

Dafür gibt es vier Farbstufen, die folgende Einstellungen repräsentieren:

- Grün: Jeder in Facebook kann Ihre Daten sehen
- Orange: Nur für Freunde sichtbar
- Rot: Vor allen Benutzern versteckt
- Blau: Für eine Teilmenge Ihrer Freunde sichtbar

Der Vorteil gegenüber den beiden anderen Ansätzen ist, dass dabei das ständige Bewusstsein für die Privatsphäre-Einstellungen erhöht wird. Bei den anderen Ansätzen führt man das Tool in der Regel **einmal** aus und ändert dann entsprechend seine Einstellungen. Der FPW macht den Benutzer während des Benutzens von Facebook **ständig** auf die möglichen Einstellungen aufmerksam, da die farblichen Hervorhebungen von entsprechenden Bereichen fortwährend angezeigt werden.

2.5 Abgrenzung zu den existierenden Ansätzen

Die wissenschaftliche Arbeit (Lockr), siehe Abschnitt 2.1, versucht mittels Kryptografie die *Privacy* zu schützen, insbesondere vor unerwünschtem Zugriff auf eigene Daten. Die existierenden Ansätze beschäftigen sich alle mit den Privatsphäre-Einstellungen und der Erhöhung des Bewusstseins des Benutzers für diese Einstellungen (und deren Konsequenzen). Diese Arbeit beschäftigt sich nicht nur mit den Privatsphäre-Einstellungen und der Frage, wem es möglich sein soll, eine Information zu sehen, sondern versucht den Inhalt der Information zu überprüfen. Das bedeutet, der Benutzer soll während des Schreibens einer Statusmeldung unmittelbar Feedback bekommen, welche Abschnitte in dem Text der Statusmeldung potenziell privatsphärenkritisch sind und ob er den Text so veröffentlichen sollte. Dadurch soll verhindert werden, dass Privatsphäre gefährdende Inhalte überhaupt veröffentlicht werden. Das Kapitel 4 beschreibt die Idee und zeigt das Konzept für eine Browser-Erweiterung, die eine solche Überprüfung und Visualisierung anbietet.

Zunächst werden aber im folgenden Kapitel 3 die theoretischen Grundlagen für die *Privacy* Betrachtung erläutert und angewandte Techniken zur Textüberprüfung erklärt, die in dieser Arbeit eingesetzt werden.

3 Theoretische Grundlagen

Dieses Kapitel gibt einen Überblick über die theoretischen Grundlagen, die in dieser Arbeit genutzt wurden. Zunächst werden der Begriff *Privacy* und die damit verbundenen Probleme beschrieben. Anschließend werden allgemeine Konzepte zur Verbesserung der Privatsphäre erklärt. Im letzten Teil werden die in dieser Arbeit angewandten Techniken behandelt, mit einer Einführung zu Naive Bayes Klassifizierern und einer Erläuterung der Levenshtein Distanz.

3.1 Privacy Problem

Der Begriff *Privacy* (deutsch Datenschutz und Privatsphäre) ist nicht einheitlich definiert, jedoch gibt es einige Definitionsversuche. Diese Arbeit orientiert sich an dem Recht auf Selbstbestimmung, das aus dem Volkszählungsurteil in Karlsruhe 1983 [3] hervorgegangen ist.

Folgende Definition soll als Grundlage der Betrachtungen zur *Privacy* dienen.

> *„Der Bereich, in dem eine Person selbst bestimmt (oder bestimmen können sollte), wem sie wann und warum welche Information über sich selbst zugänglich macht." [18]*

In einer computerlosen Welt können nur Personen im direkten Umfeld Informationen über eine Person erlangen, entweder über direkten Austausch oder über Beobachtung. Das heißt, eine Person hat Einfluss auf ihre Privatsphäre durch die Wahl ihres Umfeldes. Es kann trotzdem passieren, dass dieses Umfeld die Information wiederum an dessen Umfeld weiterreicht und die betreffende Person somit in ihrer Privatsphäre verletzt wird. Allerdings geschieht diese Verbreitung von Informationen mit einer wesentlich geringeren Geschwindigkeit und Reichweite, als es in der heutigen Zeit des Internets der Fall ist. Das Internet spielt eine zunehmend zentralere Rolle in unserem Leben und immer mehr unserer Daten werden dort digital verarbeitet und gespeichert.

Als Benutzer die Kontrolle über diese Daten zu bewahren ist ein sehr komplexes Problem, denn sobald jemand anderes Zugriff auf diese Daten erhält, ist nicht mehr klar, was damit geschieht. Dieser müsste dafür sorgen, dass die Informationen adäquat gesichert und geschützt werden, dennoch kann es passieren, dass der Betreiber nicht für einen solchen Schutz sorgen kann oder möchte. Das Fehlen eines adäquaten Schutzes kann vom Betreiber einerseits unbewusst durch schlechte IT Sicherheit hervorgerufen werden, sodass Fremde Zugriff auf die Daten erlangen können. Andererseits kann der Betreiber die *Privacy* bewusst missachten, indem die Daten von internen oder externen Services (gegebenenfalls mit kommerziellem Hintergrund) verwendet werden. Hierzu sollte der Zweck der verwendeten Daten zwischen Benutzer und Betreiber genau festgelegt werden (*Purpose Binding*) wie auch der zukünftige Umgang mit den Daten [10]. Um diese Probleme zu vermeiden, wäre ein naiver Ansatz, die Daten erst gar nicht im Internet zu verwenden, aber dann müsste der Benutzer auf verschiedene Vorteile verzichten, die er durch das Teilen von Informationen erlangt (zum Beispiel einen auf Benutzer zugeschnittenen Service). Daher sollten die Betreiber in der Pflicht sein, vom Benutzer nur die Daten anzufordern, die für den Service wirklich nötig sind (Datenminimierung). Dennoch muss der Benutzer die zu erwartenden Vorteile gegenüber den Kosten (Verlust der Informationskontrolle) abwägen und entscheiden, ob es sich lohnt, die eigenen Informationen für die Vorteile "einzutauschen". Entgegen dem Paradigma, möglichst wenige Daten an Internetbetreiber zu geben, entwickelt sich ein Trend, dass das Teilen von Informationen besser ist als sie nicht zu teilen, sogar ohne ernsthaften Vorteil. Hier setzt diese Arbeit an und versucht, nicht das Teilen von Informationen zu verhindern, sondern vielmehr den Benutzer hinsichtlich privatsphärenkritischer Angaben zu sensibilisieren, sodass dieser selbst merkt, was er nicht im Internet mitteilen sollte [10].

Ein weiteres Problem in Verbindung mit *Privacy* ist der Kontext, in dem Informationen geteilt werden. Mit Kontext ist hierbei die Verbindung der Information zu einer Zielgruppe und einem Sinnzusammenhang gemeint. Eine Information wird in der Regel an eine bestimmte Zielgruppe (zum Beispiel enge Freunde) weitergegeben und oftmals in einem Zusammenhang, der für diese Zielgruppe klar ist. Wenn diese Information nun aber den Kontext wechselt (zum Beispiel können die Information auch Arbeitskollegen sehen), dann kann das zu unerwarteten (negativen) Auswirkungen für den Benutzer führen. Das bedeutet, wenn der Benutzer eine Information teilt, wäre es wünschenswert, wenn diese im von ihm definierten Kontext steht (*Context Binding*) [10].

Der nächste Abschnitt beschreibt sogenannte *Privacy Enhancing Technologies*, die der Verbesserung der Privatsphäre eines Benutzers dienen sollen.

3.2 Privacy Enhancing Technologies

Privacy Enhancing Technologies (PETs) [34] sind Tools, Programme und Mechanismen, die versuchen, die Privatsphäre von Online-Benutzern zu schützen.

Die Ziele von PETs sind unter anderem [37]:

- Kontrolle über die Daten durch den Benutzer erhöhen
- Sammlung der Daten durch Dienstleister minimieren
- Anonymisierung des Benutzers durch zum Beispiel Pseudonyme
- Aushandeln von Nutzungsbedingungen
- Technische Ausführung dieser vereinbarten Nutzungsbedingungen
- Verfolgung und Auflistung von Transfer der Benutzerdaten, sodass der Benutzer eine Übersicht seiner Daten hat
- Die gesetzlichen Rechte des Benutzers fördern

Beispiele für PETs [5] sind:

- **Automatische Datenanonymisierung nach einer bestimmten Zeit**, damit die verarbeiteten Daten technisch so aufbewahrt werden, sodass eine Identifizierung des Benutzers nur so lange stattfinden kann, wie die Daten zweckdienlich sind.
- **Verschlüsselung** von übertragenen Daten, damit diese während der Übermittlung nicht abgehört werden können.
- **„Cookie Cutters"** die das Missbrauchen von Cookies durch Betreiber verhindern sollen, indem Cookies für bestimmte Seiten abgeschnitten werden.
- **„Platform for Privacy Preferences" (P3P)**[1], ein Portal, dass dem Benutzer ermöglicht verschiedene Datenschutzbestimmungen von Webseitenbetreibern einzusehen und zu vergleichen. Des Weiteren können diese Politiken gegenüber den eigenen Privatsphärenpräferenzen verglichen werden.

Diese Arbeit beschäftigt sich mit dem Entwickeln einer PET, dabei handelt es sich um ein auf Facebook spezialisiertes Werkzeug, das den Benutzer aufmerksam machen soll, falls dieser potenziell privatsphärenkritische Statusmeldungen veröffentlichen möchte, siehe Kapitel 4.

3.3 Angewandte Techniken

In diesem Abschnitt werden die theoretischen Ansätze für einen Naive Bayes Klassifizierer (effizienter Klassifizierer, um Objekte der Klasse zuzuordnen, der sie mit der größten Wahrscheinlichkeit angehören) und die Levenshtein-Distanz (Maß für Textverschiedenheit) eingeführt, die in dieser Arbeit praktisch eingesetzt werden.

3.3.1 Naive Bayes

Zu einem großen Teil wird sich in dieser Arbeit mit der Analyse und Einstufung von Texten beschäftigt. Einerseits ist interessant, welche Teile des Textes kritisch sind (siehe Kapitel 4.1), andererseits ist aber gleichermaßen eine Bewertung des gesamten Textes interessant. Für diese Gesamtbewertung kommt in dieser Arbeit unter anderem ein Naive Bayes Klassifikator zum Einsatz. Hierfür werden sogenannte Trainingsdaten (Texte mit bekannter Klassifizierung) erfasst, anhand derer der Klassifikator lernen und zukünftige Texte einstufen kann.

Bayes Klassifikatoren im Allgemeinen ordnen, anhand bereits gelernter Zuordnungen von Objekten zu Klassen (Traningsdaten), ein beliebiges Objekt einer Klasse zu, der es mit der größten Wahrscheinlichkeit angehört. Diese Klassifikatoren basieren auf dem Bayestheorem (Gleichung 3.1), das die Berechnung von bedingten Wahrscheinlichkeiten beschreibt [17].

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)} \tag{3.1}$$

wobei

- $P(A)$ die A-priori-Wahrscheinlich für ein Ereignis A ist
- $P(B \cap A)$ die Wahrscheinlichkeit für ein Ereignis B unter der Bedingung ist, dass A eingetreten ist

[1] http://www.w3.org/P3P/

- $P(B)$ die A-priori-Wahrscheinlichkeit für ein Ereignis B ist

Der Naive Bayes Klassifizierer ist wegen seiner Einfachheit und guten Performance beliebt [19]. Die Grundannahme ist, dass jedes Attribut von seiner Klasse unabhängig ist (daher der Name **Naive** Bayes). Das bedeutet in einem Text, dass die Worte voneinander unabhängig sind. Diese Annahme ist selbstverständlich nicht korrekt, da Texte keinen Sinn machen würden, wenn die Worte voneinander unabhängig wären, dennoch führt diese Annahme in der Regel zu guten Ergebnissen [25].

Um die Funktionsweise des Naive Bayes Klasssifizierers in Bezug auf die Problemstellung dieser Arbeit zu erläutern, dient das folgende Beispiel zum Verständnis (angelehnt an [21]).

Angenommen wir möchten wissen, mit welcher Wahrscheinlichkeit ein Text T, der die Wörter w_1 und w_2 enthält, zur Klasse $kritisch$ gehört.

- 15 von 30 bisherigen Texten wurden als $kritisch$ markiert
- 20 von 30 Texten enthalten das Wort w_1
- 11 von den Texten mit w_1 wurden als $kritisch$ markiert
- 15 von 30 Texten enthalten w_2
- Davon wurden 12 als $kritisch$ markiert

Wie groß ist nun die Wahrscheinlichkeit, dass Text T zur Klasse $kritisch$ gehört, mit dem Wissen, dass der Text w_1 und w_2 enthält?

- **Gewöhnlicher Ansatz:**

$$\frac{P(w_1|kritisch \cap w_2) * P(w_2|kritisch) * P(kritisch)}{P(w_1|w_2)P(w_2)}$$

  Mit jedem zusätzlichen Wort wird die Formel komplexer, da neue bedingte Wahrscheinlichkeit (für die Wörter untereinander) berechnet werden müssten.

- **Naive Bayes Ansatz:**

  Annahme: Wörter unabhängig

$$\frac{P(w_1|kritisch) * P(w_2|kritisch) * P(kritisch)}{P(w_1)P(w_2)} = \frac{\frac{11}{15} * \frac{12}{15} * \frac{15}{30}}{\frac{20}{30} * \frac{15}{30}} = 0.88$$

  Demnach beträgt die Wahrscheinlichkeit, dass der Text T (mit unter anderem den Wörtern w_1 und w_2) zur Klasse $kritisch$ gehört, 88%.

Die Funktion, um einen beliebigen Text zu klassifizieren laute also:

$$classify(w_1, w_2, ..., w_n) = \arg\max_{klasse}(P(klasse) * \prod_{i=1}^{n} P(w_1|klasse)) \tag{3.2}$$

Man beachte, dass der Nenner aus der Formel 3.1 entfernt wurde, da dieser eine Konstante darstellt. Für die Berechnung der Wahrscheinlichkeiten an sich dürfte man den Nenner nicht entfernen, aber in diesem Fall sind wir nur an der wahrscheinlichsten Klasse interessiert, also der Sortierung.

Damit Trainingsdaten und deren enthaltene Worthäufigkeiten erlernt werden können, wird folgendes Verfahren verwendet: Sei W einer Menge von allen Wörtern eines Textes T und $klasse$ bezeichnet die Klasse, der dieser Text zugeordnet werden soll. Für jedes Wort w in W wird der Zähler words$[w][klasse]$ erhöht und für die Klasse $klasse$ wird der Zähler classes$[klasse][w]$ erhöht. Das heißt, im Lernprozess wird für jedes Wort gespeichert, wie häufig es einer Klasse zugeordnet wurde, und für jede Klasse wird gespeichert, wie häufig sie einem Wort zugeordnet wurde. Aus den so gelernten Daten lassen sich dann die Wahrscheinlichkeiten für die *classify* Funktion (Gleichung 3.2) ermitteln.

3.3.2 Levenshtein-Distanz

Es wird im Rahmen dieser Arbeit versucht, bestimmte Worte in einem Text wieder zu erkennen, während ein Benutzer diesen Text eintippt. Der naive Ansatz ist, den Text Wort für Wort durchzugehen und zu überprüfen, ob das gesuchte Wort mit dem Wort im Text übereinstimmt. Allerdings kann es vorkommen, dass im Text Tippfehler vorhanden sind, dann funktioniert eine Überprüfung auf Gleichheit zweier Worte nicht mehr. Daher wird in dieser Arbeit versucht, die Ähnlichkeit zweier Zeichenketten zu bestimmen, damit trotz Tippfehlern bestimmte Worte erkannt werden. Zur Bestimmung einer solchen Ähnlichkeit wird in dieser Arbeit mit der Levenshtein-Distanz [20] gearbeitet, welche häufig zur Bestimmung von Ähnlichkeiten zweier Zeichenketten eingesetzt wird.

Die Levenshtein-Distanz zweier Zeichenketten entspricht der minimal nötigen Anzahl an Operationen (Einfügen, Löschen und Ersetzen), damit die erste Zeichenkette in die Zweite umgewandelt werden kann [9, 20]. Ein Algorithmus zur Bestimmung der Levenshtein-Distanz zweier Zeichenketten ist in Listing 3.1 zu finden.

Ein Beispiel ist die Distanz zwischen ”Test” und ”Fest”, diese beträgt 1, denn es ist **eine** Ersetzung von T durch F nötig. Ein anderes Beispiel ist die Distanz zwischen ”Tier” und ”Tor”. Diese beträgt zwei: 1. Ersetze i durch o (Toer) 2. Lösche e (Tor).

Listing 3.1: Pseudocode-Implementierung der Levenshtein-Distanz-Funktion

```
1  int LevenshteinDistance(string s, string t)
2  {
3    int len_s = length(s), len_t = length(t), cost = 0
4
5    if(s[0] != t[0]) then cost = 1
6
7    if(len_s == 0) then      return len_t
8    else if(len_t == 0) then return len_s
9    else                     return minimum(LevenshteinDistance(s[1..len_s-1], t) + 1,
10                                           LevenshteinDistance(s, t[1..len_t-1]) + 1,
11                                           LevenshteinDistance(s[1..len_s-1], t[1..len_t-1]) + cost)
12 }
```

Dieses Kapitel hat grundlegend in die *Privacy*-Problematik eingeführt und verschiedene Maßnahmen zur Verbesserung der *Privacy*, sogenannte Privacy Enhancing Technologies, vorgestellt. Anschließend wurden Naive Bayes Klassifizierer und die Levenshtein-Distanz erklärt, die für diese Arbeit verwendet werden.

Das folgende Kapitel 4 beschreibt die genaue Funktionalität der entwickelten Browser-Erweiterung. Dabei wird darauf eingegangen, welche Aspekte eines Textes kritisch sein können und wie die Visualisierung der Überprüfungsergebnisse aussehen soll. Abschließend werden in Kapitel 4 noch einige wichtige Designentscheidungen aufgeführt.

4 Konzept

In dieser Arbeit wird eine Browser-Erweiterung mit dem Namen ”Facebook Post Checker” (FPC) für den Browser Chrome von Google entwickelt. Diese Erweiterung soll dem Benutzer die Möglichkeit geben, seine eigenen Statusmeldungen während des Verfassens (”on-the-fly”) in Facebook hinsichtlich Privatsphäre überprüfen zu lassen und dem Benutzer dann farblich kenntlich zu machen, an welchen Stellen der Text der Statusmeldung privatsphärenkritisch ist. Dafür werden verschiedene Zusatzinformationen verwendet und verarbeitet, wie zum Beispiel die Profil-Informationen des Benutzers (Wohnort, Hobbys und so weiter) und gelernte Einstufungen von verschiedenen Statusmeldungen. Eine konzeptionelle Darstellung ist in Abbildung 4.1 zu sehen.

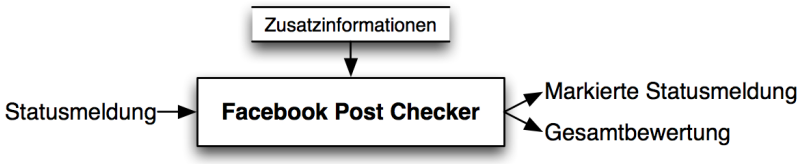

Abbildung 4.1: Konzept des Facebook Post Checker (FPC). Eine Statusmeldung wird eingegeben und vom FPC verarbeitet, das heißt mit Hilfe von Zusatzinformationen bewertet und markiert. Daraus resultiert eine Statusmeldung mit markierten Textpassagen und eine Gesamtbewertung.

Im nächsten Abschnitt 4.1 werden zunächst die aufgestellten Einstufungs-Klassen hinsichtlich kritischer Inhalte für Texte definiert und dargestellt welche Aspekte in einem Text zu überprüfen sind. Anschließend wird in Abschnitt 4.1.5 die sogenannte ”Analyse Pipeline” beschrieben, ein in dieser Arbeit entstandenes Verfahren zur modularen und erweiterbaren Überprüfung von verschiedenen Aspekten in einem Text. Im darauf folgenden Abschnitt 4.1.6 wird der Naive Bayes Klassifizierer erläutert, der zur Gesamtbewertung eines Textes beitragen soll. Die Idee der Gesamtbewertung wird in Abschnitt 4.1.7 erläutert. Nachfolgend wird in Abschnitt 4.2 das Konzept für die Visualisierung der Überprüfungsergebnisse erklärt und anhand von Bildern verdeutlicht. Abschließend werden für diese Arbeit wichtige Designentscheidungen unter Abschnitt 4.3 aufgeführt und begründet.

4.1 Überprüfung der Statusmeldungen

Der vom Benutzer eingegebene Text muss untersucht werden, um kritische Textpassagen ausfindig zu machen. Diese kritischen Textpassagen einer Statusmeldung werden in dieser Arbeit in drei Stufen klassifiziert, wobei sich die entsprechenden Farben an dem mentalen Modell der Ampel orientieren.

- **Sehr kritisch**: Diese Textpassage sollte nicht veröffentlicht werden und wird rot markiert.
- **Kritisch:** Diese Textpassage könnte kritisch sein, sollte aber vom Benutzer überprüft werden. Der Text wird gelb markiert.
- **unkritisch:** Diese Textpassage kann ohne größere Überprüfung durch den Benutzer geteilt werden. Der Text wird grün markiert.

Es gibt verschiedene Aspekte, die in einer Statusmeldung überprüft werden müssen, die nachfolgend erklärt werden.

4.1.1 Abweichung von Angaben im Profil hinsichtlich der Privatsphäre-Einstellung

Ein Benutzer kann in Facebook sein Profil ausfüllen, das umfasst persönliche Angaben in der eigenen ”Info”-Seite, sowie ”Gefällt-mir”-Angaben. Die ”Info”-Seite enthält Informationen über Arbeit und Ausbildung, Wohnorte, Beziehung, Familie, allgemeine Informationen (Geburtstag, Geschlecht, etc.), sowie Kontaktinformationen. Die ”Gefällt-mir”-Seite listet Lieblingsbücher, -filme, -serien, -spiele, -sportler, -sportmannschaften und favorisierte Aktivitäten, Interessen sowie inspirierende Personen auf. Des Weiteren sind ”Gefällt-mir”-Angaben zu Facebook-Seiten möglich, die nicht exakt unter eine dieser Kategorien fallen. Zum Profil eines Benutzers gehören ebenso die eigenen Freunde, das heißt, mit wem er auf Facebook verbunden ist.

Jede dieser Angaben hat eine bestimmte **Privatsphäre-Einstellung**: eine Menge von Benutzern, für die diese Informationen sichtbar sein sollen. Die voreingestellten Privatsphären-Einstellungen sind öffentlich, Freunde, Freunde ohne Bekannte und ”Nur ich”.

Auch für Statusmeldungen kann ein Benutzer festlegen, welcher Menge von Benutzern diese Meldung angezeigt werden soll. Nun kann es vorkommen, dass jemand Statusmeldungen schreibt, in denen Informationen aus seinem Profil erwähnt werden, siehe Beispiel in Abbildung 4.2. Wenn die Privatsphäre-Einstellung der Statusmeldung eine von der Information abweichende ist, dann liegt eine **Inkonsistenz** vor.

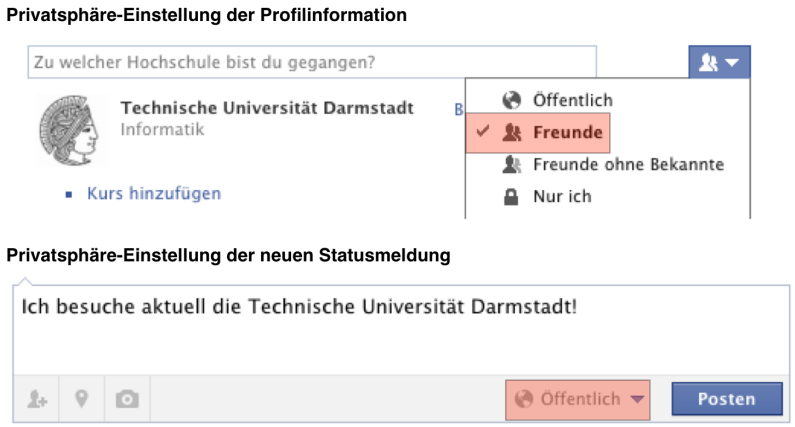

Abbildung 4.2: Profilinformation und deren Wiederverwendung in einer Statusmeldung. Die gewählten Privatsphären-Einstellungen (rot gefärbt) bilden eine Inkonsistenz, denn ”Technische Universität Darmstadt” sollte nur für Freunde sichtbar sein, die Statusmeldung, die diese Profilinformation enthält, ist jedoch auf ”Öffentlich” gesetzt.

Eine solche Inkonsistenz ist dem Benutzer als *sehr kritisch* (rot) darzustellen, andernfalls als unkritisch.

Damit die Inkonsistenzen festgestellt werden, müssen die relevanten Profil-Daten von Facebook ausgelesen werden sowie deren zugehörigen Privatsphäre-Einstellungen. Mehr dazu in Kapitel 5.3.

Um Profilinformationen trotz Tippfehler (zum Beispiel Darmstad, anstatt Darmstad**t**) korrekt zu identifizieren und einzustufen, kann eine Levenshtein-Option (siehe Abschnitt 3.3.2) gewählt werden.

4.1.2 Reguläre Ausdrücke

Reguläre Ausdrücke sind Muster, die nach syntaktischen Regeln eine Menge von Zeichenketten beschreiben (zum Beispiel beschreibt das Muster ”$a.*b$” alle Zeichenketten, die mit a beginnen und mit b aufhören). Diese regulären Ausdrücke können dazu verwendet werden, bestimmte Muster in einem Text zu erkennen [6].

Mit Hilfe dieser regulären Ausdrücke lassen sich Textinhalte erkennen, die von einem Benutzer nicht veröffentlicht werden sollten. Reguläre Ausdrücke bedürfen einer nachträglichen Überprüfung des Benutzers und sind als *kritisch* (gelb) einzuordnen.

Folgende reguläre Ausdrücke werden von der Browser-Erweiterung erkannt.

- Kreditkartennummern
- E-Mail-Adressen
- IBAN-Kontonummern (nationale Kontonummern sind nicht regulär)

Die Erweiterung ist so gestaltet, dass weitere reguläre Ausdrücke einfach hinzugefügt werden können, siehe Abschnitt 4.1.5.

4.1.3 Pronomen

Pronomen (Fürwörter) sind eine Wortklasse, die stellvertretend für ein Nomen stehen [4]. Oftmals können hierdurch unbeabsichtigt kritische Dinge mitgeteilt werden, ohne dass der Benutzer sich dessen bewusst ist.

In dieser Arbeit werden die zwei Unterklassen, Possessiv- und Reflexivpronomen, erkannt und als *kritisch* (gelb) eingeordnet.

- **Possessivpronomen:** mein(e), dein(e), unser(e)
- **Reflexivpronomen:** mich, dich, sich, uns, euch

Beispiele:

- Heute wieder so ein nerviger Tag auf der Arbeit, **mein** Chef ist echt das Letzte!
- Ich glaube ich habe gestern Nacht **mein** Fahrrad nicht abgeschlossen im Herrngarten vergessen!
- Hey Daniel, es gibt eine neue Gewinnaktion bei Vodafone, sag ihnen einfach, dass ich **dich** geworben habe, hier meine Daten: KundenNr.: 16729375673 , HandyNr.: 01754836183, wir teilen **uns** dann die Prämie!

Die Pronomen sind allerdings nur ein Hinweis auf kritische Inhalte und müssen vom Benutzer überprüft werden und werden daher als *kritisch* (gelb) eingestuft.

4.1.4 Städtenamen

Ortsangaben sind kritisch, da der Standort verraten werden kann, aber auch Abwesenheit signalisiert werden kann. Eine solche Ortsangabe wird *kritisch* (gelb) dargestellt.
Beispiel 1: "Ich bin gerade in **Frankfurt** und schaue das Fußballspiel", hierbei könnte zum Beispiel ein "Stalker" diese Information missbrauchen.
Beispiel 2: "Die nächsten Tage fahre ich nach **Berlin**", enthält Informationen über eine Abwesenheit und können zum Beispiel von Einbrechern missbraucht werden.

In dieser Arbeit wurde sich aus Gründen der Effizienz darauf beschränkt zu überprüfen, ob der Text Städtenamen enthält. Dafür werden die 100 größten Städte Deutschlands überprüft.

4.1.5 Analyse Pipeline

Die Analyse Pipeline soll der modularen Überprüfung eines Textes dienen. In diese Analyse Pipeline wird ein Text eingegeben, der dann von hintereinander geschalteten Modulen überprüft wird (siehe Abbildung 4.3).

Jeder Aspekt (siehe Abschnitt 4.1.1-4.1.4) kann in einem Modul definiert werden. Falls sich neue Aspekte ergeben, die in einem Text kritisch sein können, können diese in neuen Modulen beschrieben und in die Analyse Pipeline eingefügt werden.

Am Anfang wird der vom Benutzer eingegebene Text an Modul 1 übergeben, und anschließend bis Modul n auf die verschiedenen Aspekte hin untersucht und markiert. Hierbei erhält jedes Modul als Eingabe eine Liste seines Vorgängermoduls (Modul 1 bekommt eine einelementige Liste mit dem gesamten Text), wobei jeder Eintrag in der Liste aus einem Text und optional aus Markierungsinformationen ($Highlight$) besteht. Das jeweilige Modul iteriert dann über die Einträge in der Liste und untersucht jeden **nicht markierten** Eintrag auf einen Aspekt hin (zum Beispiel auf einen bestimmten regulären Ausdruck). Es werden nur die **nicht** markierten Einträge beachtet, weil Textpassagen sonst mehrfach untersucht und markiert würden und die Performance darunter leiden würde. Falls ein Modul innerhalb eines Eintrags eine kritische Textpassage findet, wird dieser Eintrag innerhalb der Liste in drei Teile aufgebrochen. Der erste Teil entspricht dem Text vor der kritischen Textpassage, der zweite Teil ist die kritische Textpassage selbst (versehen mit Markierungsinformationen) und der dritte Teil ist der Text nach der kritischen Textpassage. Nachdem die Iteration über alle Einträge abgeschlossen wurde, wird die Liste an das nächste Modul weitergegeben. Dieses geht dann wieder die Einträge (welche keine Markierungen enthalten) durch und so weiter. Das letzte Modul gibt dann seine Liste an die Ausgabe der Pipeline.

Beim Entwerfen der Pipeline sollten die Module so aneinandergereiht werden, dass das Modul mit der höchsten Priorität den Anfang der Pipeline bildet und das Modul mit der niedrigsten Priorität das Ende. In dieser Arbeit ist der Vergleich von Profilinformationen und Text höher priorisiert als reguläre Ausdrücke, da wir bei Ersterem konkrete Hinweise haben, dass der Benutzer etwas mit seinem Profil Inkonsistentes veröffentlichen möchte.

Aus den gewonnenen Markierungsinformationen in der Liste können im späteren Verlauf die farblichen Hervorhebungen in der Oberfläche konstruiert werden, wie in Abschnitt 4.2 beschrieben.

4.1.6 Naive Bayes Klassifizierer

Der FPC enthält einen Naive Bayes Klassifizierer, der aufgrund von gelernten Trainingsdaten klassifiziert, ob eine Statusmeldung **wahrscheinlich** die Privatsphäre verletzt oder nicht. Hierfür wird der theoretische Ansatz (siehe Kapitel 3.3.1) umgesetzt und anhand von Trainingsdaten ermittelt, ob eine **gesamte** Statusmeldung wahrscheinlich **kritisch** oder **nicht**

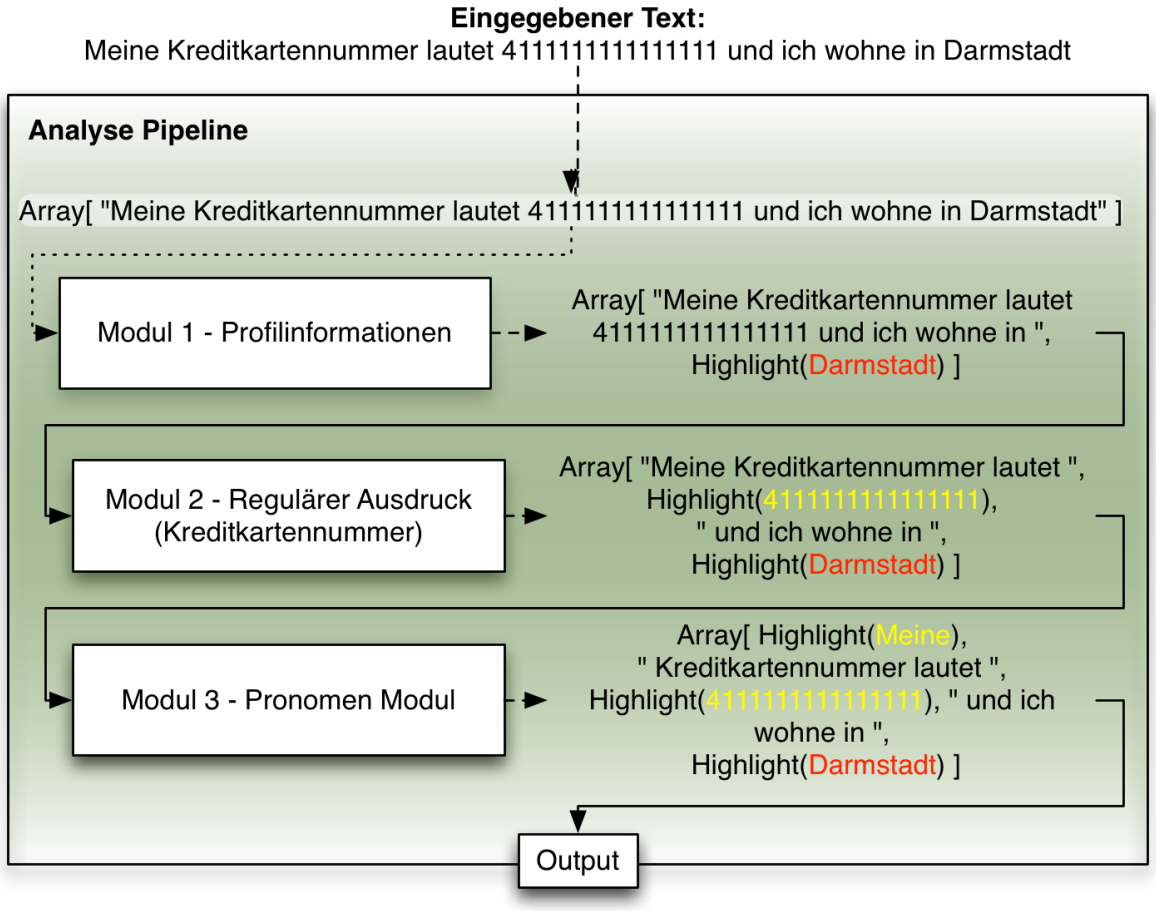

Abbildung 4.3: Zeigt den Aufbau und die Funktionsweise der Text-Analyse-Pipeline. Hierbei wird ein Text durch verschiedene Module geschickt, die den Text auf verschiedene Aspekte untersuchen und entsprechend markieren.

kritisch ist. Die Klassifizierung stuft eine Statusmeldung als problematisch ein, wenn $P(kritisch) > P(unkritisch)$, sonst als unkritisch. Um diesen Grenzwert besser bestimmen zu können, sind Realdaten nötig, die aber aktuell nicht verfügbar sind. In dieser Arbeit wird das Lernen von Trainingsdaten über die Eingabe durch ein Textfeld gelöst, in das der Benutzer Beispiele für Statusmeldungen eingibt und auswählen kann, ob diese potentiell problematisch oder unproblematisch sind. Das Naive Bayes Model wird nach jedem Trainingsvorgang persistent in einer Datenbank abgelegt, somit ist, über mehrere Browser-Sitzungen hinweg, gewährleistet, dass die gelernten Daten nicht verloren gehen. Ein Hinweis auf das Erfassen von qualitativen und realen Trainingsdaten wird in Kapitel 7.2 gegeben.

4.1.7 Gesamtbewertung

Am Ende jeder Textüberprüfung soll eine Gesamtbewertung über den geschriebenen Text abgeben werden. Ein Ablaufdiagramm ist in Abbildung 4.4 zu sehen. Ist der gesamte Text voraussichtlich *sehr kritisch, kritisch oder unkritisch*. Diese Gesamtbewertung wird beeinflusst durch das Ergebnis der Analyse Pipeline (Markierungen) und durch die Einstufung des Naive Bayes Klassifizierers, siehe Abbildung 4.4. Wenn die Pipeline mindestens ein $Highlight$ ausgibt, das rot ist, wird der gesamte Text als *sehr kritisch* bewertet. Wenn die Pipeline mindestens ein $Highlight$ ausgibt, das gelb ist, wird der gesamte Text als *kritisch* bewertet. Wenn der Naive Bayes Klassifizierer für die Klassen *nicht kritisch* und *kritisch* unterschiedliche Wahrscheinlichkeiten ausgibt sowie die Wahrscheinlichkeit für *kritisch* höher ist als für *nicht kritisch*, dann wird der gesamte Text als privatsphärenkritisch bewertet.

4.2 Visualisierung der Überprüfungsergebnisse

Nachdem die Überprüfung abgeschlossen ist, müssen die Ergebnisse für den Benutzer sichtbar gemacht werden. Hierfür wird in Abschnitt 4.2.1 erläutert, wie die Passagen des Textes, die in der Analyse Pipeline markiert wurden, im Eingabefeld

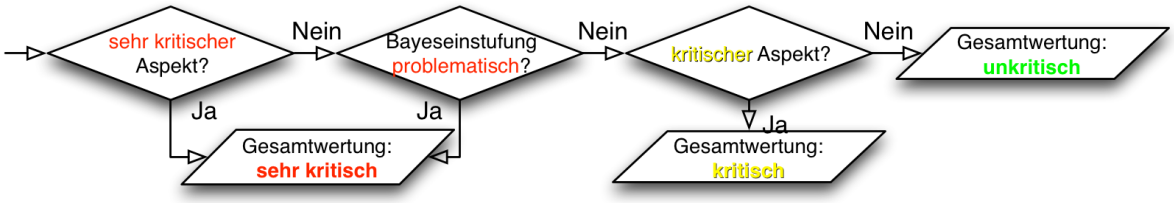

Abbildung 4.4: Ablauf der Gesamtbewertung einer Statusmeldung, also wann die gesamte Statusmeldung in welche Stufe (*sehr kritisch, kritisch, unkritisch*) eingestuft wird.

der Statusmeldung durch Färbungen visualisiert werden. Anschließend wird in Abschnitt 4.2.2 erläutert, wie die erfolgte Gesamtbewertung für den Benutzer kenntlich gemacht wird.

4.2.1 Kritische Textpassagen

Die kritischen Textpassagen werden direkt in dem Eingabefeld (auf Facebook) farblich hinterlegt (rot / gelb / grün), siehe Abbildung 4.5.

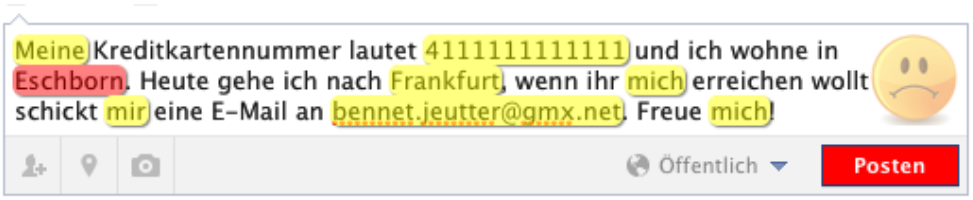

Abbildung 4.5: Screenshot einer markierten Statusmeldung in Facebook mit Gesamtbewertung *sehr kritisch* (Privatsphäre-Einstellung: öffentlich; Profilinformation "Eschborn" allerdings "Nur ich")

Beim Bewegen des Mauszeigers über eine farblich markierte Textpassage erscheint zudem ein kleines Popup (siehe Abbildung 4.6), in dem beschrieben wird, was das (potenzielle) Problem ist. Das Popup in Abbildung 4.6 bezieht sich auf eine Inkonsistenz mit einer Profilinformation und zeigt dem Benutzer, welche Einstellung für diese Profilinformation gewählt wurde sowie an welchem Ort diese Information auf Facebook hinterlegt ist. Zudem wird ein Hinweis an den Benutzer gegeben, welche Privatsphäre-Einstellung er für die Profilinformation wählen sollte.

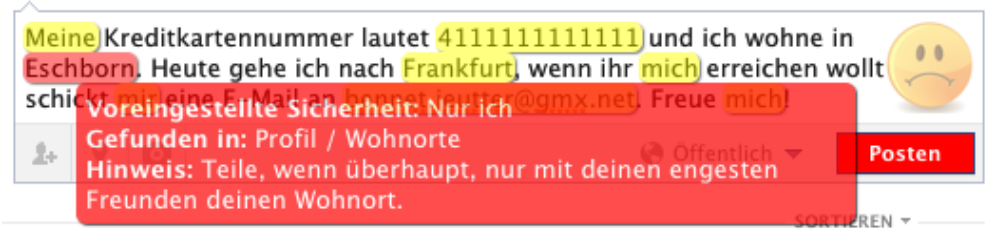

Abbildung 4.6: Screenshot einer markierten Statusmeldung in Facebook und dem Popup für das *sehr kritische* Wort "Eschborn" (Wohnortangabe im Profil)

Bei kritischen Pronomen wird ein Hinweis mit folgendem Inhalt angezeigt: "Pronomen (Fürwörter) können Hinweise auf sehr persönliche Dinge sein." Ortsangaben, die kritisch markiert wurden, liefern beim Bewegen der Maus über die Markierung folgenden Hinweis: "Ortsangaben sollten nicht gepostet werden, da sie Informationen über deinen Standort oder deine Abwesenheit (Einbruchgefahr) enthalten können." Der Hinweis für reguläre Ausdrücke kann für jeden einzelnen Ausdruck definiert werden. Für Kreditkarten-Nummern lautet er zum Beispiel "Regulärer Ausdruck: Kreditkartennummer".

4.2.2 Gesamtbewertung

Nach jeder Überprüfung einer Statusmeldung wird eine Gesamtwertung ermittelt (siehe Abschnitt 4.1.7), diese kann wieder *sehr kritisch, kritisch* oder *unkritisch* sein. Für die Visualisierung dieser Gesamtbewertung wird im Hintergrund der Textbox ein Smiley angezeigt (siehe Abbildung 4.7). Dieser Smiley wird auch in drei Stufen visualisiert: Traurig schauen (*sehr kritisch*), Kritisch schauen (*kritisch*), Fröhliche schauen (*unkritisch*).

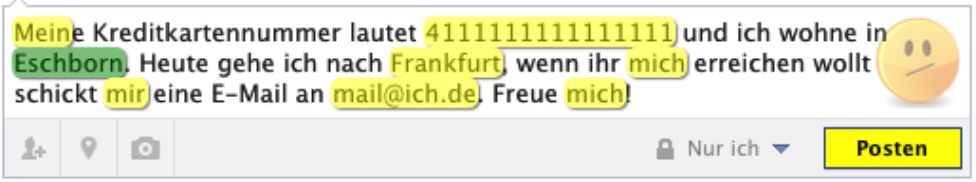

Abbildung 4.7: Screenshot einer markierten Statusmeldung in Facebook mit Gesamtbewertung *kritisch*, da kritische Informationen enthalten sind, die von Facebook genutzt werden können, auch wenn die Privatsphäre-Einstellung auf "Nur ich" gesetzt ist (Privatsphäre-Einstellung: Nur ich; Profilinformation "Eschborn" auch "Nur ich")

Neben der Visualisierung der Gesamtbewertung durch den Smiley wird der "Posten"-Button unter der Textbox farblich angepasst. Dabei erscheint der Button bei einer *sehr kritischen* Gesamtbewertung rot, bei einer kritischen Gesamtbewertung gelb, andernfalls bleibt die Farbe unverändert (blau). Tabelle 4.1 zeigt eine Übersicht über die drei Stufen der Gesamtbewertung und die entsprechenden Darstellungen von Smiley und Button.

Tabelle 4.1: Zeigt eine Übersicht über die drei Stufen der Gesamtbewertung und die entsprechenden Darstellungen von Smiley und Button

Klassifizierungstufe	Darstellung Smiley	Darstellung "Posten"-Button
Sehr kritisch		
Kritisch		
Unkritisch		

Falls die Gesamtbewertung durch den Naive Bayes Klassifizierer beeinflusst wurde, dann erscheint neben dem "Posten"-Button noch eine Textmeldung, dass diese Gesamtbewertung durch Lerndaten erzielt wurde (siehe Abbildung 4.8).

Abbildung 4.8: Screenshot einer durch den Naive Bayes Klassifizierer als "kritisch" eingestuften Nachricht.

4.3 Designentscheidungen

Die getroffenen Designentscheidungen dieser Arbeit werden nachfolgend erläutert.

4.3.1 Browser-Erweiterung versus externes Programm

Anfänglich war die Idee, ein Programm (anstatt der Browser-Erweiterung) zu entwickeln, das die Statusmeldungen überprüft und die Profilinformationen ausliest.

Der Vorteil wäre, vom Browsertyp (Chrome, Firefox, Internet Explorer, etc.) unabhängig zu sein. Jedoch überwiegen die Nachteile eines externen Programms. Die Umsetzung als zusätzliches Programm wäre nicht benutzerfreundlich, da der Benutzer für das Verfassen seiner Statusmeldungen ein zusätzliches Programm hätte starten müssen. Zudem gibt es technische Probleme, denn die offizielle Facebook-API liefert keine Informationen über die Privatsphäre-Einstellungen zu den Profilinformationen. Eine alternative Idee, anstatt der Nutzung der Facebook API, war, mit Hilfe eines internen HTTP-Clients das HTML-Dokument von Facebook anzufragen und dort die Privatsphäre-Einstellungen heraus zu extrahieren. Allerdings setzt Facebook unter anderem JavaScript ein, sodass das bloße HTML-Dokument keine Informationen liefern würde.

Hingegen kann mit einer Browser-Erweiterung das Document Object Model (DOM) [39], eine Schnittstelle, um auf HTML Dokumente zuzugreifen, ausgelesen und so die Privatsphäre-Einstellungen erlangt werden. Des Weiteren kann das optische Feedback zu einer Statusmeldung während des ”normalen” Nutzens von Facebook an den Benutzer geliefert werden.

Nachteil ist jedoch, dass, wenn Facebook Änderungen an der HTML-Struktur vornimmt, dann die Browser-Erweiterung angepasst werden muss.

4.3.2 Überprüfung und Markierung des Textes während der Eingabe

In der entwickelten Erweiterung werden kritische Textpassagen direkt während der Eingabe überprüft und markiert. Eine mögliche Alternative wäre, die Überprüfung und Visualisierung erst nach dem Klick auf den ”Posten”-Button durchzuführen und den Benutzer dann zu fragen, ob er wirklich veröffentlichen möchte. Hierbei ergibt sich das Problem, dass der Benutzer unüberlegt ”weiter” klickt (ähnlich wie bei Nutzungsbedingungen von Programmen). Zudem ist anzunehmen, dass sich der Benutzer wohlmöglich mehr mit dem Text beschäftigt und das Bewusstsein für den Inhalt erhöht wird, wenn er während des Schreibens Feedback bekommt und nicht am Ende den kompletten Text überarbeiten muss. Ein Nachteil dieser Lösung ist die Performance, da bei jedem neuen Zeichen eine Überprüfung stattfindet und nicht nur einmal am Ende des Verfassens. Allerdings ist diese geringere Performance während der Benutzung nicht ”spürbar”.

4.3.3 Erkennung von Ortsangaben

Eine alternative Idee zur Erkennung von Ortsangaben, gegenüber den Städtelisten, ist die Nutzung eines Kartendienstes, wie zum Beispiel Google Maps[1]. Dadurch wäre es möglich die Entfernungen zwischen dem Wohnort und einem erwähnten Ort feststellen und so eine genauere Einstufung zu treffen. Allerdings wurde diese Idee nicht umgesetzt, da durch den entstehenden Datenverkehr eine stark verschlechterte Performance der Auswertung zu erwarten ist. Demnach wäre keine Überprüfung der Statusmeldung in Echtzeit mehr möglich. Durch den Datenaustausch über den Google-Service wären zudem negative *Privacy*-Implikationen zu erwarten.

Eine lokale Lösung mit einer großen Geo-Datenbank (z.B. OpenGeoDb[2]) wäre alternativ denkbar, aber die Größe der Browser-Erweiterung würde dadurch stark zunehmen, denn alleine der SQL-Dump für Deutschland ist aktuell 60MB groß. Darüber hinaus bringt ein solcher Ansatz technische Probleme mit sich, denn die Daten müssten in ein vom Browser verarbeitbares Datenbank-Format gebracht werden, die aber beschränkter sind, als übliche Datenbank-Formate (wie z.B. MySQL).

4.3.4 Verwendung von Bibliotheken zur Sprachverarbeitung

Es wurde überlegt, eine Bibliothek einzubinden, um natürliche Sprachverarbeitung durchführen zu können. Eine populäre Java-Bibliothek zur Sprachverarbeitung ist LingPipe[3]. Damit ist es zum Beispiel möglich, Texte zu klassifizieren, Wortkorrekturen vorzunehmen und vieles mehr. Diese Bibliothek wurde in Java entwickelt und kann über eine Java-API angesprochen werden.

Tests in einem ähnlichen Szenario ergaben jedoch, dass die Bibliothek große Mengen an Arbeitsspeicher (> 1000MB) benötigt, was zu einer Verlangsamung des kompletten Rechners führt.

[1] https://maps.google.de

[2] `http://opengeodb.org/`

[3] http://alias-i.com/lingpipe/

Ein weiteres Problem ist die Nutzung einer Java-API in JavaScript. Da momentan keine umfangreichen Bibliotheken zur natürlichen Sprachverarbeitung in JavaScript existieren, muss auf andere Sprachen zurückgegriffen werden. Hierfür müsste ein Server eingerichtet werden, der über einen Webservice Zugriff auf diese Java-API bietet. Dafür müsste bei jeder Überprüfung einer Statusmeldung eine Abfrage an diesen Server durchgeführt werden und, unabhängig von der verwendeten Bibliothek, ein gewisser Kommunikations-Overhead in Kauf genommen werden, der zur Verschlechterung der Performance führen würde. Die Benutzerfreundlichkeit könnte ebenso darunter leiden, da der Benutzer möglicherweise noch den Server installieren und starten müsste, bevor er die Browser-Erweiterung vollständig nutzen könnte, da eine Mitlieferung in der Browser-Erweiterung technisch nicht möglich ist.

4.4 Kapitelzusammenfassung

Das Kapitel 4 hat die Idee der Browser-Erweiterung erklärt und die zu implementierenden Funktionalitäten aufgestellt. Dafür wurde zunächst detailliert erläutert, welche Aspekte in einem Text kritisch sein können und wie diese eingestuft werden sollen (*sehr kritisch / kritisch / unkritisch*). Anschließend wurde erklärt, wie die Überprüfung des Textes modular, anhand einer Analyse Pipeline sowie eines Naive Bayes Klassifizierers stattfindet und wie die Ergebnisse dieser Überprüfung für den Benutzer dargestellt werden. Abschließend wurden einige Designentscheidungen für die Browser-Erweiterungen diskutiert.

Die explizite Umsetzung des in diesem Kapitel vorgestellten Konzepts wird im nachfolgenden Kapitel 5 beschrieben. Dabei wird die Architektur erläutert und ein JavaScript-Framework zur effizienteren Entwicklung vorgestellt. Zudem wird erklärt, wie Daten von Facebook bezogen werden, um die konkrete Umsetzung der Browser-Erweiterung für Facebook mit Informationen für die Überprüfung zu versorgen. Dann werden einige Details der Implementierung für die Echtzeit-Überprüfung und Visualisierung der Ergebnisse besprochen.

5 Implementierung

Dieses Kapitel enthält eine Beschreibung der konkreten Umsetzung der Browser-Erweiterung und dessen Komponenten. Abbildung 5.1 zeigt alle zu entwickelnden Komponenten und deren Relationen untereinander. Zum einen wird gezeigt, welche Profilinformationen von Facebook ausgelesen werden müssen, das bedeutet all die Seiten, die Informationen enthalten, die vom Benutzer über Privatsphäre-Einstellungen konfigurierbar sind. Hierfür werden zum einen die allgemeinen Informationen wie Wohnort, Arbeitsstätte etc., ausgelesen, aber auch die Freunde-Liste. Es werden zudem die ”Gefällt-Mir”-Angaben ausgelesen, also welche Filme, Musik, Bücher, etc. ein Benutzer mag. Diese Informationen werden durch die Profilanalyse automatisiert ausgelesen und in einer Datenbank abgelegt, samt Privatsphäre-Einstellungen. Zum anderen werden die beiden Überprüfungs-Stränge gezeigt, das heißt die Analyse Pipeline (Untersuchung auf verschiedene Aspekte, siehe Abschnitt 4.1.5) und der Naive Bayes Klassifzierer. Diese Überprüfungsergebnisse resultieren dann in einer markierten Statusmeldung und der Visualisierung einer Gesamtbewertung.

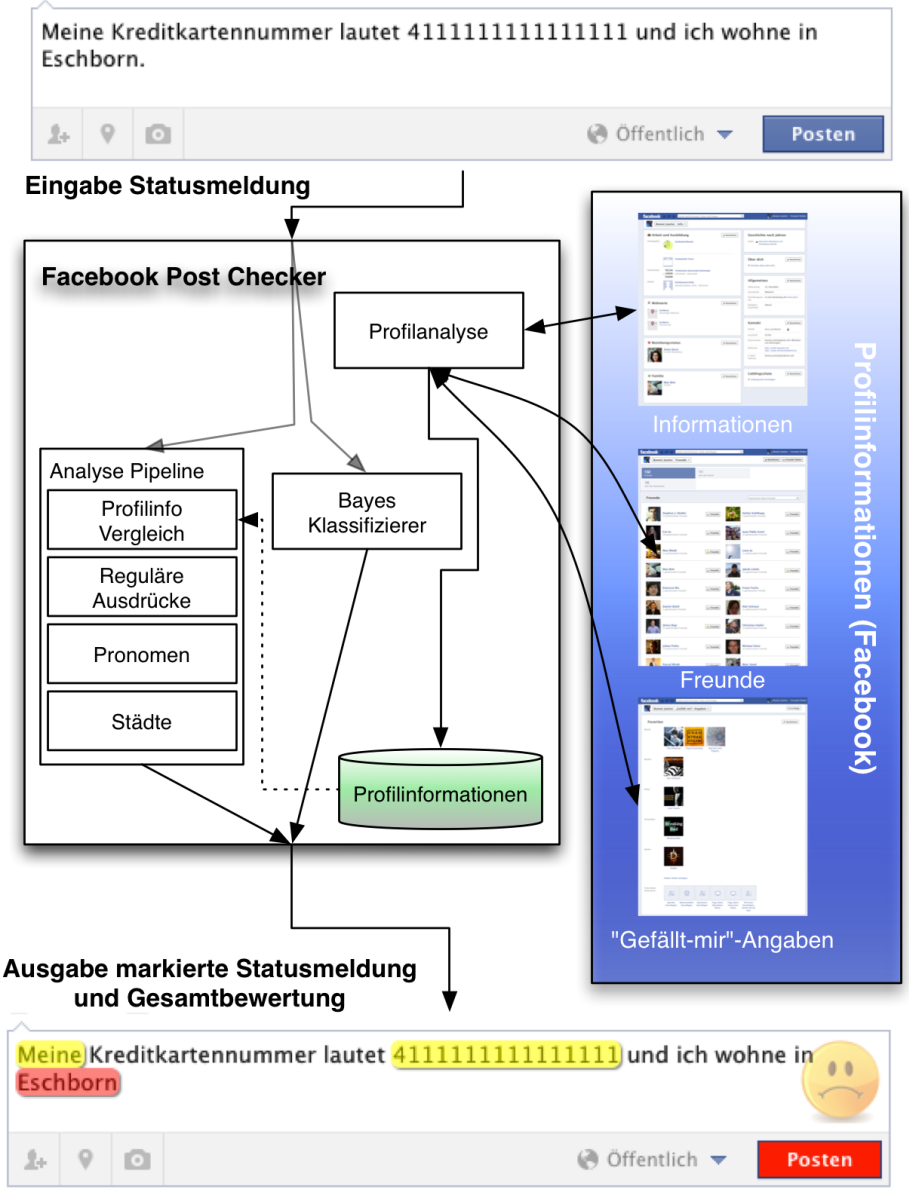

Abbildung 5.1: Die verschiedenen Komponenten der Browser-Erweiterung, die implementiert werden müssen und wie sie miteinander verbunden sind.

Im ersten Abschnitt 5.1 wird die Architektur der Google Chrome Erweiterung erläutert. Danach folgt in Abschnitt 5.2 eine kurze Einführung zum Open-Source JavaScript Framework jQuery [27]. Anschließend wird in Abschnitt 5.3 erklärt,

wie und welche, Daten von Facebook bezogen und gespeichert werden. Der darauf folgende Abschnitt 5.4 beschäftigt sich mit den Implementierungsdetails der Echtzeit-Überprüfung der eingegebenen Statusmeldungen in Facebook. Im letzten Abschnitt 5.5 wird die technische Umsetzung der Visualisierung der Überprüfungsergebnisse erläutert.

5.1 Architektur der Google Chrome Erweiterung

Das Entwickeln einer Google Chrome Erweiterung birgt hinsichtlich der Softwarearchitektur Eigenheiten und Beschränkungen, die hier eingeführt werden. Google Chrome unterscheidet zwischen Content-Scripts, Background-Pages (inkl. Script) und Popup-Pages.

Background Pages haben ihren eigenen Scope (Bereich, in dem Variablen sichtbar sind) und laufen im Hintergrund, solange der Browser an ist, unabhängig davon, auf welcher Seite man sich gerade befindet. Diese Background Pages haben deutlich mehr Privilegien, das bedeutet Zugriffsrechte, auf verschiedene Browser-Komponenten als die anderen Typen (Content-Script / Popup-Page, siehe unten) und können zum Beispiel Tabs und Datenbanken verwalten.

Content-Scripts hingegen werden auf der Seite (zum Beispiel Facebook) injiziert und laufen im Scope der Seite, auf der sie injiziert wurden. Diese haben einen Zugriff auf das HTML-Dokument (DOM [39]) dieser Seite und können dieses auslesen und manipulieren.

Eine **Popup-Page** erzeugt neben der Adress-Leiste ein Icon für die Erweiterung, bei dessen Klick ein Popup mit dem Inhalt der Popup-Page angezeigt wird. Hier könnten zum Beispiel Konfigurationen angezeigt werden. In der entwickelten Browser-Erweiterung wird in der Popup-Page ein Button zum Analysieren des Profils sowie ein Eingabefeld, um den Naive Bayes Lerner zu trainieren, angezeigt, siehe Abbildung 5.2.

Abbildung 5.2: Screenshot des Browser-Erweiterung-Popups des FPC.

Aus den unterschiedlichen Privilegien der verschiedenen Seitentypen ergibt sich, dass Content-Scripts und Background-Pages einzeln wenig Anwendungsraum für umfangreichere Erweiterungen bieten und somit zusammenarbeiten müssen. Es ist nicht direkt möglich, aus einem Content-Script heraus JavaScript-Methoden der Background-Page aufzurufen. Deswegen müssen API-Methoden von Google Chrome zur Kommunikation verwendet werden (zum Beispiel chrome.extension.sendMessage [14]), womit unkompliziert Nachrichten zwischen den einzelnen Komponenten hin und hergeschickt werden können. Dazu wurde ein Prozess konstruiert, welcher das Aufrufen von Methoden in der Background-Page nachbildet. In unserer Implementierung besteht jede Nachricht an die Background-Page aus einem $messageType$ und einem $data$-Objekt, wobei $messageType$ für den gewünschten Methoden-Namen und das $data$-Objekt die Parameter für die Methode enthält. Abbildung 5.3 zeigt, welche Pages / Scripts dieser Arbeit welche Zugriffsrechte haben und die nötige Kommunikation untereinander.

Die Popup-Page (siehe Abbildung 5.2) dient lediglich dem Starten der Profilanalyse sowie dem Trainieren des Naive Bayes Lerners. Dabei werden Methoden der Background-Page aufgerufen, die dann über entsprechende Methoden der Tab-Verwaltung verschiedene Content-Scripts in Facebook ausführt. Die verschiedenen Content-Scripts könnten auf den DOM (das HTML-Dokument) von Facebook zugreifen und dieses manipulieren. Die Content-Scripts zur Profil-Analyse geben dann Informationen an die Background-Page zurück, welche die erfassten Daten mit Hilfe der Datenbank-Privilegien ablegen kann.

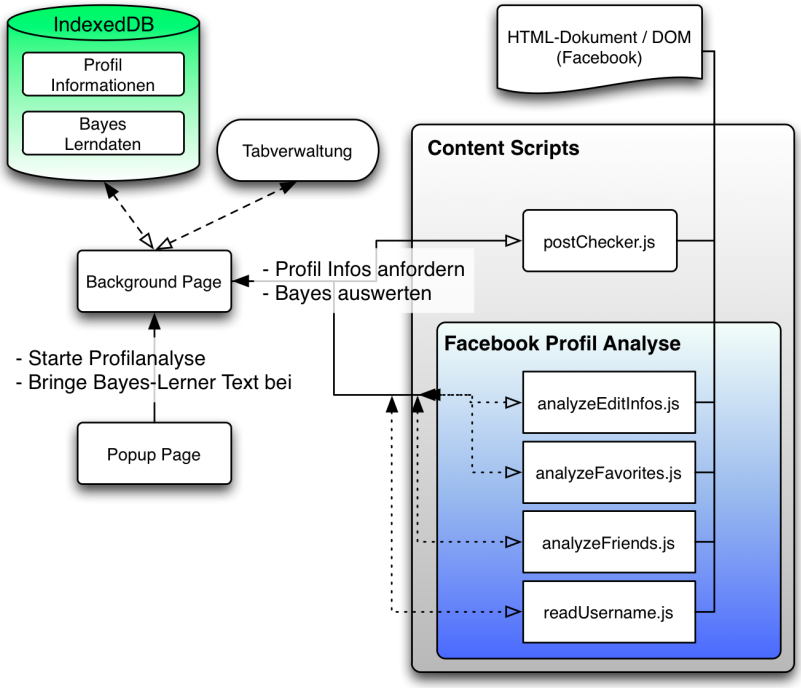

Abbildung 5.3: Architektur und Aufteilung der Google Chrome Erweiterung

Im Anhang dieser Arbeit ist eine Übersicht der Ordner- und Dateistruktur der Browser-Erweiterung zu finden.

Um zunächst die vielen Daten aus Facebook zu lesen und später das optische Feedback an den Benutzer zu geben, wird jQuery verwendet, das im folgenden Abschnitt vorgestellt wird.

5.2 jQuery Library

jQuery [27] ist ein weitverbreitetes, kostenloses Open-Source JavaScript-Framework, das eine effiziente Entwicklung von JavaScript Anwendungen ermöglicht. Es vereinfacht das Traversieren von HTML Dokumenten, das Event Handling, die Animierung von Objekten und asynchrone Web Anfragen. Das Framework ist vollständig CSS3 kompatibel sowie auf browserübergreifende Funktionalität ausgelegt.

Ein Grund für die Verwendung von jQuery in dieser Arbeit ist, dass eine große Anzahl DOM-Zugriffe (lesend und schreibend) durchgeführt werden müssen. Zum Beispiel müssen die Profilinformationen des Benutzers sowie die dazugehörigen Privatsphäre-Einstellungen aus dem HTML-Dokument von Facebook extrahiert werden. Mit jQuery lassen sich gesuchte DOM-Knoten mit einer Zeile Code finden, was die Übersichtlichkeit und Kompaktheit des Codes fördert.

In Listing 5.1 ist ein Beispiel zur Reduzierung des Codes mittels jQuery zu sehen. Dabei handelt es sich um ein simples Beispiel.

Listing 5.1: Beispiel zur Färbung des Textes aller Elemente mit der Klasse "box", mittels jQuery [36]

```
1  // jQuery
2  $('.box').css('color', 'red');
3
4  // Übliches JavaScript (moderne Browser)
5  [].forEach.call( document.querySelectorAll('.box'), function(el) {
6    el.style.color = 'red'; // or add a class
7  });
8
9  // Legacy JavaScript (browserkompatibel)
10 var box = document.getElementsByClassName('box');
11 var i = 0, len;
12 for ( len = box.length; i < len; i++ ) {
13    box[i].style.color = 'red';
14 }
```

Ein weiterer Vorteil ist, dass jede Methode eines jQuery Objekts wieder dieses Objekt zurückgibt, somit lassen sich lange Ketten von Anforderungen erstellen, siehe Listing 5.2. Dadurch werden Arbeitsspeicher und Leistung gespart, da

der DOM nicht ständig neu durchsucht werden muss, sondern nur einmalig bei Aufrufen des Selektors durch jQuery. Selektoren sind Muster, die zur Suche im DOM verwendet werden, können [32].

Listing 5.2: Beispiel, in dem mit einer jQuery-Code-Zeile 1. ein span-Element erzeugt 2. einen Text (inneres HTML) setzt 3. zwei CSS-Klassen hinzufügt 4. die Hintergrundfarbe festlegt 5. ein Mouse-Hover-Event-Listener definiert wird

```
1 $("<span></span>").html(text).addClass("fbword h").css('background-color', 'red').
2 hover( hoverFn, unHoverFn).appendTo(body);
```

5.3 Daten von Facebook auslesen und speichern

Ein wichtiger Aspekt der Erweiterung ist das Auffinden von Inkonsistenzen zwischen den Privatsphäre-Einstellungen der verfassten Statusmeldung und denen der Profilinformationen (siehe Abschnitt 4.1). Damit diese Detektion durchgeführt werden kann, müssen zunächst alle Profilinformationen und deren Privatsphäre-Einstellungen vorliegen. Die erste Überlegung war, die offizielle Facebook-API zu verwenden, um an diese Daten zu gelangen, allerdings war sehr schnell klar, dass so nicht alle relevanten Daten auszulesen sind. Insbesondere die Privatsphäre-Einstellungen lassen sich mit der API nicht auslesen. Daher wird in dieser Arbeit mittels der Chrome Erweiterung auf das Facebook-Profil zugegriffen, das bedeutet die Seiten, die Profilinformationen enthalten, werden in verschiedenen Tabs geöffnet und mit einem speziellen Content-Script versehen. Das jeweilige Content-Script liest dann mittels jQuery die entsprechenden Bereiche der Seite aus.

Besonderheiten beim Auslesen

Auf jeder Seite, die Profilinformationen enthält, muss automatisiert auf ”Bearbeiten” geklickt werden, auf diese Weise werden die Privatsphäre-Einstellungen sichtbar. Die entsprechenden Informationen sind vorher auch nicht versteckt im Quellcode vorhanden, sondern werden erst nach dem Klicken auf ”Bearbeiten” über AJAX-Anfragen (Technik zur asynchronen Datenübertragung zwischen Browser und Server [8]) im Hintergrund geladen. Das führt dazu, dass eine unbestimmte Zeit (in der Regel nur wenige Sekunden) nach dem Klick auf ”Bearbeiten” gewartet werden muss, bis die entsprechenden Informationen im DOM verfügbar sind. Das Warten auf die DOM-Objekte wird mittels Timeout-Methoden [29] realisiert (Beispiel siehe Listing 5.3). Hierbei prüft man alle x Sekunden, ob das DOM-Objekt verfügbar ist. Es gibt zwar Browser-Events (z.B. DOMNodeInserted), allerdings werden diese nicht von allen Browsern unterstützt und gelten mittlerweile wieder als *deprecated* [31].

Listing 5.3: Zeigt den Einsatz von Timeout-Methoden um aufbestimmte DOM-Objekte zu warten

```
1  function moveOnCrawlEmbededSection(section) {
2    // In dem Pagelet nach einem Input-Element suchen
3    // (dieses ist erst verfügbar, wenn die AJAX-Anfrage erfolgreich verarbeitet wurde)
4    var inputs = $("#pagelet_" + section).find("input");
5
6    // Wurde ein solches Element nicht gefunden?
7    if(inputs.length == 0) {
8      // Dann starte diese Methode in 1000 Millisekunden
9      // mit dem Parameter section erneuert
10     setTimeout(moveOnCrawlEmbededSection, 1000, section);
11     return;
12   }
13 ...
14 }
```

Damit die Privatsphäre-Einstellungen ausgelesen werden können, ist es möglich, an bestimmten Stellen mit dem Selektor ([32]) $div.audienceSelector\ select$ zu arbeiten - diese Elemente sind im Quellcode versteckt und unsichtbar. Bei der Suche nach Elementen mit diesem Selektor wird ein Select-Element (Auswahlliste, hier aber nur als Informationscontainer verwendet) zurückgegeben, wobei dessen numerischer Wert wie folgt zu interpretieren ist:

80 entspricht **öffentlich**.
40 entspricht **Freunde**.
127 entspricht **Freunde ohne Bekannte**.
10 entspricht **nur ich**.
Nachfolgend werden die einzelnen Seiten und deren Struktur beschrieben.

Info-Seite

Die Info Seite[1] enthält grundlegende Informationen, wie zum Beispiel Arbeitgeber und Wohnort.

[1] URL für allgemeine Profil Informationen: `https://www.facebook.com/benutzername/info`

Auf dieser Seite muss zwischen zwei Arten von Informationscontainern unterschieden werden: Pagelet und Popup.

Pagelets enthalten verschiedene Informationen, binden ihre Formularfelder sowie Privatsphäre-Einstellungen aber erst in die bestehende Seite ein, nachdem auf ”Bearbeiten” geklickt wurde. Diese sind im DOM über eindeutige IDs auslesbar, zum Beispiel $section\_eduwork$. Pagelets werden (aktuell) lediglich für die Sektion ”Arbeit und Ausbildung” verwendet, der Rest wird über Popups realisiert, das bedeutet neu geladene Boxen mit Informationen. Popups werden beim Klick auf ”Bearbeiten” komplett neu erzeugt. Beim Auslesen dieser Popups gibt es eine Besonderheit, die beachtet werden muss: Zu jedem Zeitpunkt kann nur ein Popup offen sein (beim Versuch zwei Popups zu öffnen wird das Erste geschlossen), das heißt, es ist nicht möglich, nach dem Laden der Seite alle ”Bearbeiten”-Buttons zu klicken, sodass alle Popups geladen werden. Stattdessen muss ein Popup nach dem anderen geöffnet und ausgelesen werden. Hierzu wurde eine Queue eingerichtet die es erlaubt, nach dem Laden der Seite einmal alle gewünschten Popups zu definieren, die dann nacheinander abgearbeitet werden.

Listing 5.4: Ausschnitt der Popup-Auslese-Methoden hinsichtlich der Queue Funktionalität

```
/**
* Startet den Prozess des Auslesens einer Popup Section
*/
this.crawlPopupSection = function(section) {
  // Diese Anfrage dem Queue hinzufügen
  popupSectionQueue.push(section);

  // Wenn nur diese Sektion in der Queue ist, dann mit dem Auslesen beginnen
  if(popupSectionQueue.length == 1) {
    // "Bearbeitenmodus!" für Bereich öffnen
    clickLink($("#"+section+"_edit_button"));

    // Weiterarbeiten
    moveOnCrawlPopupSection(section);
  }
}


function moveOnCrawlPopupSection(section) {
  [...]
  // Bearbeiten beenden, das heißt bisheriges Popup schließen
  closeEditPopupSection();

  popupSectionQueue.shift(); // Aktuelle Sektion aus der Queue löschen

  // Falls weitere Sektion vorhanden
  if(popupSectionQueue.length != 0) {
    // Die nächste Sektion abarbeiten lassen
    self.crawlPopupSection(popupSectionQueue.pop());
  }
}
```

Favoriten-Seite

Die Favoriten-Seite[2] enthält Informationen über die eigenen Interessen (definiert über ”Gefällt mir”-Angaben), zum Beispiel, welche Band gehört oder welcher Filme gemocht wird. Für jede dieser Kategorien (also Musik, Filme, Spiele und so weiter) kann jeweils eine eigene Privatsphäre-Einstellung festgelegt werden. Es gibt auf dieser Seite nur einen ”Bearbeiten”-Button, der alle Kategorien zum Bearbeiten freigibt (und somit die Privatsphären-Einstellungen anzeigt). Dann lassen sich die jeweiligen Profilinformationen auslesen, indem über alle tr-Elemente (Tabellenzeilen) mit der CSS-Klasse $editLikesRow$ im DOM iteriert und anschließend von den gefundenen Elementen das $aria-label$-Attribut ausgelesen wird. Jedes dieser $editLikesRow$ Elemente enthält wiederum ein th-Kind-Element (Tabellenkopf-Zeile) mit der Klasse $label$, welches den Kategorienamen enthält (z.B. Spiele).

Eine Besonderheit auf der Favoriten-Seite stellen noch sonstige Favoriten dar, denn hier werden die jeweiligen ”Gefällt mir”-Angaben erst nach dem Klick auf ”Andere Seiten, die dir gefallen” in einer ”Profile-Browser-List” geladen, die dann iterativ über den Selektor $div.fbProfileBrowserList\ ul\ li\ a[data-gt]$ ausgelesen werden kann.

[2] URL für Favoriten-Seite: `https://www.facebook.com/benutzername/favorites`

Freunde-Seite

Die Freunde-Seite[3] enthält eine Liste aller Facebook-Freunde. Eine besondere Schwierigkeit beim Auslesen dieser Seite liegt darin, dass anfänglich nur 14 Freunde in der Liste geladen sind. Die restlichen Freunde werden erst nach und nach beim Runterscrollen nachgeladen. Daraus ergibt sich, dass die Erweiterung automatisiert ständig nach unten scrollen muss (siehe Listing 5.5), bis keine weiteren Freunde mehr geladen werden. Die Seite enthält alle Freunde, sobald ein DOM-Element mit der Klasse *fbProfileBrowserNoMoreItems* existiert.

Listing 5.5: Zeigt den Code-Abschnitt um die Seite so lange scrollen zu lassen, bis keine neuen Freunde mehr geladen wurden.

```
1   function scrollUntilNoMoreItemsAndReadFriends() {
2       // Seite bis ganz unten scrollen
3       window.scrollBy(0, 100000);
4
5       // Prüfen ob noch mehr Profile geladen werden müssen
6       if($(".fbProfileBrowserNoMoreItems").length == 0) {
7           // Wenn ja kurz warten und dann die Methode neu starten
8           setTimeout(scrollUntilNoMoreItemsAndReadFriends, 1000);
9       } else {
10          // Ansonsten mit dem Auslesen beginnen
11          startReadingFriends();
12      }
13  }
```

Datenbank

Für die Datenspeicherung wurde IndexedDB [33] verwendet. Es ist aktuell (Stand 2013) der empfohlene Ansatz zur Speicherung von großen Datenmengen auf der Client-Seite im Browser. Andere Ansätze wie WebSQL gelten als *deprecated*.

Es ist keine relationale Datenbank, sondern es werden Objekte in sogenannten ”Stores” (ähnlich zu Tabellen) abgelegt, die dann wieder abgerufen werden können. Jeder ”Store” bekommt einen Schlüssel zugewiesen, der einem einzigartigen Attribut in einem Objekt entspricht (zum Beispiel die Profil-Informationen haben den Schlüssel ’name’). Bei der Suche nach bestimmten Objekten kann dann exakt nach diesem Schlüssel gesucht werden oder ein ”Cursor” initialisiert werden, mit dem über alle Einträge im ”Store” iteriert werden kann. Solch ein ”Cursor” kann optional mit Einschränkungen versehen werden, sodass zum Beispiel nur nach den Objekten gesucht wird, deren Schlüssel mit einem bestimmten Buchstaben anfangen. In dieser Arbeit wurden lediglich uneingeschränkte ”Cursors” verwendet, um die kompletten ”Stores” auszulesen, da es keinen Fall gab, in dem spezielle Einträge benötigt wurden.

Für die in dieser Arbeit entwickelte Browser-Erweiterung werden zwei Stores angelegt:

1. **profile_informations**: Hier werden alle Objekte der Klasse $ProfileInformation$ abgelegt, die während des Auslese-Prozesses entstehen. Für die Überprüfung werden sie dann wieder ausgelesen.

2. **bayes:** Dieser Store enthält das $Bayes$-Objekt und wird bei jedem Lern-Prozess gespeichert. Beim Start der Extension wird das Objekt ausgelesen und kann so verwendet werden.

5.4 Statusmeldungen überprüfen

Der wesentliche Bestandteil der entwickelten Erweiterung ist die Analyse des vom Benutzer eingegebenen Textes auf privatsphärenkritische Inhalte. Der naive Ansatz war zunächst alle Wörter im Text durchzugehen und dann einen Query in der Datenbank abzusetzen, ob das jeweilige Wort in den gespeicherten Profilinformationen existiert. Dafür wird der Text mit bestimmten Trennzeichen (Leerzeichen, Punkte, Komma und so weiter) getrennt und anschließend über die einzelnen Wörter iteriert. Dabei ergab sich das Problem, dass häufig Profilinformationen gespeichert werden, die genau solch ein Trennzeichen enthalten (zum Beispiel ”The Offspring” enthält ein Leerzeichen und kann somit nicht im Text gefunden werden).

Dieses Problem lässt sich lösen, indem nicht über die Wörter im Text iteriert wird, sondern über die gesammelten Profilinformationen. Dafür werden diese Profilinformationen beim Öffnen von Facebook geladen und anschließend bei jeder Textänderung iteriert. Hierbei wird überprüft, ob sich der Text dieser Profilinformation im eingegebenen Text wieder findet. Es ergab sich nach einigen Tests, dass dieser Ansatz effizienter funktioniert und zu mehr Treffern im Text führt.

Der nächste im Konzept definierte Suchaspekt sind reguläre Ausdrücke. Um diese zu erkennen, wurde ein Prozess entwickelt, um die Analyse möglichst modular aufzubauen, sodass jederzeit neue Analyseverfahren (zum Beispiel das Pronomen Modul) einfach implementiert werden können, siehe Kapitel 4.1.5. Die Idee ist, dass sich beliebig viele Analysemodule definieren lassen, die ein bestimmtes Interface implementieren.

[3] URL für Freunde-Seite: `https://www.facebook.com/benutzername/friends`

Analyse Pipeline

Jedes Modul erbt von der abstrakten *AnalyzeModule*-Klasse und muss somit eine *analyze*-Methode implementieren. Diese Module können dann instanziiert werden und einem Objekt der Klasse *AnalyzePipeline* übergeben werden. Die Pipeline kann dann mit der Methode *analyzeText* sowie *rate* angesprochen werden und lässt den Text dann durch die übergebenen Module laufen (siehe Abbildung 4.3). Technisch funktioniert es so, dass es ein Array gibt, in dem am Anfang der Pipeline nur ein *string*-Element enthalten ist: der eingegebenen Text. Dieses Array wird dann an Modul 1 gegeben, das in der Folge auf diesem Array arbeitet. Das Modul sucht nun nach kritischen Textpassagen und bricht die Elemente im Array auf, die kritischen Text enthalten (siehe Listing 5.6).

Listing 5.6: Zeigt den Code-Abschnitt um das Array aufzubrechen und eine Markierung einzufügen

```
1  [...]
2  var pos = ...; // pos enthällt die Textposition des kritischen Textes
3  [...]
4
5  // Text vor dem gesuchten Wort
6  var before = currSection.substr(0, pos);
7
8  // Gesuchter Text
9  var middle = currSection.substr(pos, currValue.length)
10
11 // Nach dem gesuchten Text
12 var after = currSection.substr(pos + currValue.length);
13
14 // Sektionen aufsplitten
15 sections.splice(
16   j,
17   1,
18   before,
19   new Highlight(
20     middle,
21     this.generateHoverText(),
22     AnalyzeModule.WORD_STATE.BAD
23   ),
24   after
25 );
```

Der erste Teil entspricht einem *string*-Objekt, das den Text **bis** zum kritischen Text enthält. Der zweite Teil entspricht einem *Highlight*-Objekt, das den Text des kritischen Texts enthält sowie eine genauere Beschreibung des Problems und eine Sicherheits-Stufe (Grün / Gelb / Rot). Der dritte Teil entspricht einem *string*-Objekt, dass den Text **nach** dem kritischen Text enthält. Das nachfolgende Modul bekommt nun dieses aufgebrochene Array und geht wieder nur die *string*-Objekte durch. Dadurch werden keine Textpassagen doppelt untersucht (Effizienz) oder markiert, da die *Highlight*-Objekte nicht durchsucht werden. Das letzte Module gibt am Ende ein Array an den Pipeline-Output, welches aus *string* -und *Highlight*-Objekten besteht. Dieses kann dann von der Oberfläche visualisiert werden.

In der aktuellen Implementierung (Stand April 2013) dieser Arbeit ist die Reihenfolge der Module wie folgt festgelegt: **1.** Vergleich von Profilinformationen hinsichtlich Privatsphäre-Einstellungen **2.** Regulärer Ausdruck: Kreditkartennummern **3.** Regulärer Ausdruck: E-Mail-Adresse **4.** Regulärer Ausdruck: IBAN Kontonummer **5.** Pronomen Modul **6.** Städtemodul.

Die regulären Ausdrücke sind in Listing 5.7 zu finden.

Listing 5.7: Verwendete reguläre Ausdrücke

```
1 // Kreditkartennummern
2 (?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])
      [0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})
3
4 // E-Mail-Adressen
5 ([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})
6
7 // IBAN-Kontonummern
8 [a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}([a-zA-Z0-9]?){0,16}
```

Naive-Bayes

Die Implementierung basiert auf der Implementierung von Gary Dusbabek[4] und wurde bis auf leichte Anpassungen (insbesondere das Eingliedern der *guess*-Methode in die Klasse) in die Erweiterung integriert. Es gibt eine Klasse "Bayes" mit den Haupt-Methoden *teachText* und *guess*.

teachText bekommt als Parameter einen Text und eine Klasse, der er zugeordnet werden soll. Dabei werden alle Wörter durchlaufen und deren Häufigkeit für eine Klasse erhöht sowie die Häufigkeit der Klasse für ein Wort.

Listing 5.8: Zeigt die Lernmethode des Naive Bayes Klassifizierers

```
1 teach : function(wrd, cls) {
2         this.sanitize(wrd, cls); // Speicher in den Arrays für das Wort reservieren
3         this.words[wrd][cls]++;
4         this.words[wrd].__length++;
5         this.classes[cls][wrd]++;
6         this.classes[cls].__length++;
7         this.classes.__length++;
8     }
```

guess bekommt als Parameter einen Text und gibt ein Array der Wahrscheinlichkeiten für alle trainierten Klassen zurück (zum Beispiel 'nicht kritisch': 20%, 'kritisch': 80%)

Die Persistenz der trainierten Daten wird über einen IndexedDB-Store gewährleistet, in dem das Bayes-Objekt nach jedem Training gespeichert wird.

5.5 Farbliche Hervorhebung der gefundenen Textpassagen

Herkömmliche HTML-Textfelder unterstützen keine im Text differenzierte Formatierung, das bedeutet, einzelne Textpassagen lassen sich nicht einzeln formatieren, lediglich dem kompletten Text kann ein Stil zugewiesen werden. Daher musste ein Workaround entwickelt werden, um farbliche Hervorhebungen von bestimmten Wörtern / Textabschnitten durchzuführen. Zunächst war der Ansatz, den Hintergrund des Textfeldes transparent zu machen und für jeden zu markierenden Textabschnitt eine Box (DIV-Element) mit entsprechender Hintergrundfarbe zu generieren und an der entsprechenden Textposition hinter das Textfeld zu legen. Wie sich herausstellte, ist die Berechnung der Positionierung und Größe der Highlight-Boxen für jedes Wort komplex und fehleranfällig, da die Font-Informationen (wie zum Beispiel Zeichenabstand) nicht bekannt sind und somit geschätzt werden müssen. Ein anderer Ansatz [26] ist es, über das Textfeld eine Box mit transparentem Hintergrund zu legen, in diese Box den Text des Textfeldes zu kopieren und die Formatierungen dann in dieser Box durchzuführen. Die Formatierung geschieht mittels gezielter Einfassung von Textpassagen in *span*-Elementen, die dann formatiert werden. In dieser Arbeit werden genau die Textpassagen mit dieser Technik formatiert, die laut Analyse-Pipeline einem *Highlight*-Objekt entsprechen. Die entsprechenden *span*-Elemente werden dann mittels der CSS-Klasse *h* (für Highlight) versehen, die die Ecken abrundet und einen leichten Schatten erzeugt. Die Hintergrundfarbe der jeweiligen Markierung wird mittels dem *style*-Attribut pro Element zugewiesen. Insgesamt hat diese Methode zu einem sehr guten Ergebnis geführt, jedoch nicht ganz ohne Probleme, wie (unter anderem) nachfolgend beschrieben.

5.6 Aufgetretene Probleme

Auslesen der Privatsphären-Einstellungen

Ursprünglich wurden die Privatsphäre-Einstellungen über die CSS-Klassen des Privatsphäre-Einstellung-Buttons ermittelt. Allerdings stellte sich raus, dass diese Klassen randomisiert sind und somit keine Aussage über die Privatsphäre-Einstellung haben. Wie oben beschrieben wurde das durch das Auslesen des versteckten Select-Elementes gelöst (siehe Abschnitt 5.3, Besonderheiten beim Auslesen).

Reguläre Ausdrücke für Telefonnummern

Für die Arbeit wurden diverse reguläre Ausdrücke definiert, zum Beispiel für E-Mail-Adressen. Beim Versuch einen regulären Ausdruck für Telefonnummern zu erstellen, zeiget sich, dass die Aufgabe sehr komplex ist, denn das Aufschreiben von Telefonnummern ist nicht regulär [13]. Manche Personen trennen Telefonnummern durch Bindestriche, andere durch Schrägstriche. Manche geben Vorwahlen mit führenden Nullen an, andere ohne und so weiter. Diese unterschiedlichen Schreibweisen lassen sich nicht alle in einem regulären Ausdruck beschreiben, denkbar wäre allerdings ein Modul, das verschiedene Schreibarten prüft, indem verschiedenste reguläre Ausdrücke gesucht werden.

[4] http://www.dusbabek.org/~garyd/bayes/

change-Event der Textbox nicht nutzbar

Mittels JavaScript kann auf Events von HTML-Elementen reagiert werden. Zum Beispiel kann man auf einem beliebigen Objekt einen *click*-Event-Listener registrieren, der ausgelöst wird, sobald das entsprechende Objekt angeklickt wurde. In dieser Arbeit ist zum Beispiel das Event interessant, wenn sich der Text ändert (*change*-Event), denn so kann bei jeder Textänderung eine erneuerte Überprüfung stattfinden. Allerdings verhindert eine (unbekannte) Facebook-Komponente, dass die Erweiterung dieses Event registrieren kann, daher wurde ein Workaround entwickelt. Jede halbe Sekunde wird mittels *setTimeout* eine check-Methode aufgerufen, die überprüft, ob sich der Text geändert hat und, falls ja, diesen überprüft.

Probleme bei der farblichen Markierung des Textes

Zunächst kam es zu dem Problem, dass in manchen Fällen der Text der Overlay-Box anders umgebrochen wurde als im Textfeld. Das Problem kam dadurch zustande, dass Texteingabe-Felder mehrere Leerzeichen hintereinander anzeigen, in den restlichen HTML-Elementen werden aber mehrere Leerzeichen stets als ein Leerzeichen interpretiert. Dasselbe gilt für Umbrüche, denn diese werden im Texteingabe-Feld als Umbruch angezeigt, in sonstigen Elementen (also der Overlay-Box) aber nicht. Die Lösung beider Probleme war es, zwei Leerzeichen hintereinander und Leerzeilen durch deren HTML-Code-Äquivalente zu ersetzen, siehe Listing 5.9.

Listing 5.9: Zeigt den Code-Abschnitt, um die doppelten Leerzeichen und Umbrüche in der Overlay-Box korrekt darzustellen

```
1 var htmlText = text.replace(new RegExp("\r\n", 'g'), '<br/>')
2 .replace(new RegExp("\n", 'g'), '<br/>')
3 .replace(new RegExp("\r", 'g'), '<br/>')
4 .replace(new RegExp("  ", 'g'), "  ");
```

Dieses Kapitel 5 hat die konkrete Umsetzung der Browser-Erweiterung erläutert. Dabei wurde die Architektur dargelegt und auf Implementierungsdetails des Profil-Auslese-Prozesses, Überprüfung, sowie Visualisierung eingegangen. Des Weiteren wurden im letzten Abschnitt Probleme erklärt, die während der Umsetzung aufgetreten sind.

Das folgende Kapitel 6 zeigt die technische Evaluierung dieser Umsetzung. Dafür wird eine Testreihe aufgestellt und die Ergebnisse der Durchführung dieser Testreihe werden beschrieben.

6 Technische Evaluation

Dieses Kapitel beschreibt die Evaluierung der Funktionalität der Browser-Erweiterung. Der Browser Google Chrome ist betriebssystemunabhängig und ebenso die dazugehörenden Erweiterungen, jedoch wurde die entwickelte Erweiterung sicherheitshalber auf zwei Maschinen (Mac und Windows) in Google Chrome installiert und getestet. Damit mögliche Fehler bei der Sprachwahl ausgeschlossen werden, wurden die Tests der Erweiterung mit den Spracheinstellungen Deutsch und Englisch auf Facebook durchgeführt. Anschließend wurden die Anforderungen an die Erweiterung Schritt für Schritt mit zwei unterschiedlichen Facebook-Profilen überprüft und getestet. Insgesamt gab es somit acht Testläufe.

Das Testen der implementierten Levenshtein-Distanz-Messung hat es nicht in diese Testläufe geschafft und wurde herausgenommen. Nach einer Implementierung der Levenshtein-Distanz für diese Arbeit stellte sich nach einigen kleineren Tests heraus, dass die Performance des Algorithmus (siehe Listing 3.1) nicht annähernd ausreichend ist, da die Ergebnisse in Echtzeit benötigt werden (während der Eingabe). Sobald etwas längere Zeichenketten (etwa 8 Zeichen) im Vergleich verwendet wurden, dauerte die Berechnung, bedingt durch eine Komplexität von $\mathcal{O}(nm)$ deutlich, zu lange und der Algorithmus musste mehrere Sekunden (zum Teil sogar Minuten) rechnen. Daher werden Tippfehler aktuell nicht erkannt, siehe Abschnitt 7.2.

Sonst wurden in jedem Testlauf folgende Anwendungsfälle überprüft:

1. **Analyse des Profils**

   Kurzbeschreibung: Die Analyse des Profils muss nach der Installation, wie in Kapitel 5 beschrieben, durchgeführt werden. Dabei werden die entsprechenden Daten und Privatsphäre-Einstellungen ausgelesen.

   Vorbedingung: Benutzer ist in Facebook eingeloggt.

   Auslöser: Benutzer drückt den Button ”Facebook Profil analysieren” im Popup-Fenster der Erweiterung, neben der Adressleiste von Google Chrome.

   Prozess: Das Profil des Nutzers wird automatisiert erfasst. Die Erweiterung durchsucht jede Profilseite (Freunde, eigenes Profil, etc.) und übernimmt relevante Daten in die Datenbank. Der Prozess im Detail öffnet verschiedene Fenster und schließt diese anschließen wieder.

   Nachbedingung: Alle relevanten Daten sind in der Datenbank. Alle im Prozess geöffneten Fenster sind nach Beendigung geschlossen.

   Ergebnis: Bei allen getesteten Systemen und Sprachen wurden keine Auffälligkeiten festgestellt. 8 / 8 Testläufe erfolgreich.

2. **Überprüfung von Statusmeldungen**

   Kurzbeschreibung: Die Überprüfung der Statusmeldungen muss, wie in Kapitel 5 beschrieben, durchgeführt werden. Hierbei wird der Text auf verschiedene Aspekte hin untersucht und markiert.

   Vorbedingung: Benutzer ist in Facebook eingeloggt und befindet sich auf der Startseite.

   Auslöser: Der Benutzer gibt einen Text in die Statusmeldung-Textbox ein.

   Prozess: Die Erweiterung analysiert automatisch den eingegebenen Text und findet kritische Textpassagen.

   Nachbedingung: Die kritischen Textpassagen im Text sind farblich entsprechend markiert und zeigen ein Fenster mit Hinweisen an, wenn der Mauszeiger über die jeweilige Markierung bewegt wird.

   Ergebnis: Bis auf ”Selektierung im Textfeld”, siehe Kapitel 7.2, ließen sich in allen getesteten Systemen und Sprachen keine Auffälligkeiten feststellen. Es wurden folgende Module getestet.

   - Profilinformation Modul - entsprechend verschiedene Privatsphäre-Einstellungen getestet
   - Pronomen Modul - verschiedene Texte mit Pronomen getestet
   - RegEx Modul - Die im Konzept vorgestellten regulären Ausdrücke wurden getestet
   - Städte Modul - Stichprobentest aus den 100 größten Städten

3. **Überprüfung der Gesamtbewertung**

   Kurzbeschreibung: Es muss eine Gesamteinschätzung der eingegebenen Statusmeldung gegeben werden, wie in Kapitel 5 beschrieben.

   Vorbedingung: Benutzer ist in Facebook eingeloggt und befindet sich auf der Startseite.

Auslöser: Der Benutzer gibt einen Text in die Statusmeldung-Textbox ein.

Prozess: Die Erweiterung analysiert den Text und führt eine Gesamtbewertung des Textes durch. Der Text wird als ”unkritisch” eingestuft, falls keine kritischen Textpassagen gefunden wurden. Falls eine *sehr kritische* Textpassage gefunden wurde, wird der Text als *sehr kritisch* eingestuft, und falls eine ”kritische” Textpassage gefunden wurde, dann wird er als ”kritisch” eingestuft. Sollte eine kritische Einstufung durch den Naive Bayes Klassifizierer erfolgt sein, dann wir der Text als *sehr kritisch* eingestuft.

Nachbedingung: Je nach Einstufung wurde der Smiley im Hintergrund der entsprechenden Textbox gesetzt und der Posten-Button farblich angepasst. Falls der Naive Bayes Klassifizierer für die kritische Einstufung verantwortlich war, dann zeige neben dem Posten-Button einen Hinweis, dass die Gesamtbewertung durch den Klassifizierer beeinflusst wurde.

Ergebnis: Bei allen getesteten Systemen und Sprachen wurden keine Auffälligkeiten festgestellt. 8 / 8 Testläufe erfolgreich.

4. **Überprüfung der Lernfunktionalität des Naive Bayes Klassifizierers**

   Kurzbeschreibung: Die Daten, die dem Naive Bayes Klassifizierer eingegeben werden, müssen entsprechend gelernt und gespeichert werden.

   Vorbedingung: Google Chrome ist geöffnet.

   Auslöser: Der Benutzer öffnet das Popup-Fenster der Erweiterung, gibt dort im Textfeld einen Beispieltext ein und wählt eine Klassifizierungsstufe (problematisch / nicht problematisch). Anschließend klickt er auf ”Beibringen”.

   Prozess: Dem Naive Bayes Klassifizierer wird der Text eingegeben und die statistischen Daten entsprechend aktualisiert. Danach werden die kompletten Lerndaten des Klassifizierers in die Datenbank übernommen.

   Nachbedingung: Es erscheint im Popupfenster eine Nachricht ”Text erfolgreich beigebracht”. Die Lerndaten wurden aktualisiert und in die Datenbank übernommen.

   Ergebnis: Bei allen getesteten Systemen wurden keine Auffälligkeiten festgestellt (Sprache irrelevant, da der Lernprozess nichts mit Facebook zu tun hat). 8 / 8 Testläufe erfolgreich.

5. **Überprüfung der Einstufung durch den Naive Bayes Klassifizierer**

   Kurzbeschreibung: Ein vom Benutzer eingegebener Text muss entsprechend der Lerndaten des Naive Bayes Klassifizierers eingestuft werden.

   Vorbedingung: Der Benutzer ist in Facebook eingeloggt und es sind Trainingsdaten für den Naive Bayes Klassifizierer vorhanden.

   Auslöser: Der Benutzer gibt einen Text in die Statusmeldung-Textbox ein.

   Prozess: Der Klassifizierer analysiert den Text und überprüft ob Wörter vorhanden sind, die er als problematisch / nicht problematisch kennengelernt hat. Wenn er solche Wörter findet, dann wird der Text entsprechend eingestuft.

   Nachbedingung: Die Gesamtbewertung wird beeinflusst (siehe Usecase 3) und der entsprechende Hinweis erscheint.

   Ergebnis: Bei allen getesteten Systemen und Sprachen wurden keine Auffälligkeiten festgestellt. 8 / 8 Testläufe erfolgreich.

Bis auf ”Selektierung im Textfeld”, siehe Kapitel 7.2, ließen sich in den acht Testläufen keine Fehler feststellen und es wurde entsprechend der oberen Auflistung die Funktionalität geprüft und als ordnungsgemäß befunden.

Um die Performance zu testen, wurde folgender Test durchgeführt: Gebe einen Text mit vielen kritischen Textpassagen in die Statusmeldung-Textbox ein und kopiere diesen vielfach in die Textbox hinein.

Dabei wurde das Ergebnis (Markierung und Gesamtbewertung) in etwa einer halben Sekunde ermittelt, was für die Erweiterung ein gutes Ergebnis ist.

Die *Usuability* wurde in der parallel durchgeführten Studie [7] belegt. Die Schritte, die der Benutzer durchführen muss, sind gering. Lediglich die Installation der Erweiterung und das initiale Analysieren des Profils müssen vom Benutzer durchgeführt werden. Die Überprüfung und Markierung der eingegebenen Statusmeldungen erfolgt automatisch in Echtzeit.

7 Fazit und zukünftige Arbeiten

Dieses Kapitel gibt ein Fazit, indem die Ergebnisse der Arbeit zusammengefasst werden und der entstandene Nutzen reflektiert wird. Abschließend werden zukünftige Aufgaben vorgestellt, die sich aus dieser Arbeit ergeben haben und Raum für weiterführende Arbeiten schafft.

7.1 Fazit

Die *Privacy* von Internetbenutzern wird heute durch soziale Netzwerke gefährdet, da der Benutzer die Kontrolle über seine eingegebenen Daten und Informationen verliert. Facebook und viele andere soziale Netzwerke bieten keine eigenen Werkzeuge, um die *Privacy* der Benutzer zu schützen. Allerdings müssen die OSNs sich an die gesetzlichen Bestimmungen halten, daher könnte die Politik aktiv werden und versuchen die Benutzer stärker zu schützen, indem gewisse *privacy*-schützende Maßnahmen Pflicht werden. Trotzdem liegt es zum aktuellen Zeitpunkt in der Hand des Benutzers, ob und in welchem Umfang er seine Daten veröffentlicht (oder nicht) und für wen er diese anschließend freigibt. Zudem muss der Benutzer vor dem Veröffentlichen entscheiden, ob die enthaltenen Informationen seine *Privacy* gefährdet oder nicht. Hier fehlt häufig das Bewusstsein über *privacy*-kritische Inhalte der Benutzer, um solche Entscheidungen korrekt fällen zu können.

Es wurden existierende Ansätze zur Verbesserung der *Privacy* in sozialen Netzwerken untersucht und festgestellt, dass diese in der Regel versuchen, die Privatsphäre-Einstellungen in den sozialen Netzwerken zu optimieren, das heißt zum Beispiel die Anzahl der Leute, an die eine entsprechende Information geteilt werden soll, einzuschränken. Allerdings wird in keinem der Ansätze untersucht, welche Inhalte konkret veröffentlicht werden und ob diese *privacy*-kritisch sind.

Die in dieser Arbeit erstellte Browser-Erweiterung ”Facebook Post Checker” setzt an diesem Problem an. Dafür werden die vom Benutzer eingegebenen Statusmeldungen in Echtzeit überprüft und kritisch eingeschätzte Passagen farblich markiert. Zudem werden beim Bewegen des Mauszeigers über diese Markierungen Hinweise angezeigt, die dem Benutzer das Problem erklären. Diese Visualisierung kann dem Benutzer dann helfen, eine Entscheidung zu fällen, ob er den Text tatsächlich in dieser Fassung veröffentlichen soll. Das Bewusstsein für den Inhalt kann so oftmals gesteigert werden, allerdings stellt die aktuelle Version der Browser-Erweiterung einen Prototyp dar und kann nicht garantieren, dass die Einstufung der eingegebenen Statusmeldung 100% korrekt funktioniert. Jedoch konnte die Machbarkeit nachgewiesen werden und durch die modulare Architektur sind viele Erweiterungen sowie andere Nutzungsgebiete (zum Beispiel Internetforen oder beim E-Mail-Versand) möglich.

Aktuell ist die Überprüfung sehr statisch, das heißt, es wird nach bestimmten Wörtern / Zeichenketten gesucht und diese werden anschließend eingestuft. Die Browser-Erweiterung wird zwar auf den Benutzer zugeschnitten (und ist in dem Sinne dynamisch), indem zum Beispiel seine Profildaten bei der Überprüfung verwendet werden. Jedoch kann die Erweiterung nicht die Semantik von ganzen Sätzen erkennen und Wörter, die etwas Gleiches bedeuten (Synonyme). Hierbei ist das Problem, dass untersuchte Ansätze der natürlichen Sprachverarbeitung deutlich zu unperformant arbeiten, um in einer Echtzeit-Überprüfung eingesetzt werden zu können. Dabei ist ein großes Problem, dass es aktuell keine entsprechenden Bibliotheken für JavaScript gibt und die API über einen Server angesprochen werden müsste.

Mögliche Erweiterungen und Verbesserungen für den FPC werden im nachfolgenden Abschnitt 7.2 beschrieben.

7.2 Zukünftige Arbeiten

Nachfolgend werden offene Probleme und Ideen für diese Arbeit beschrieben.

Selektierung im Textfeld

Wie in Abschnitt 5.5 erklärt, wird über das Textfeld eine Box gelegt. Das führt dazu, dass diese Overlay-Box alle Maus-Events (wie zum Beispiel Klicks) empfängt und nicht das Textfeld. Eine schnelle Lösung ist, dass beim Klicken der Overlay-Box mittels der focus()-Methode [30] das Textfeld aktiviert wird. Dann kann der Text mit der Tastatur bearbeitet werden, aber Markierungen mit der Maus sind nicht möglich.

Alternativer Algorithmus zur Bestimmung der Ähnlichkeit zweier Zeichenketten

Die Levenshtein-Distanz als Maß der Ähnlichkeit zweier Zeichenketten wurde implementiert und getestet, allerdings stellte sich dabei heraus, dass die Performance für Echtzeituntersuchungen zu gering ist (siehe Kapitel 6). Es wird also ein effizienter Algorithmus zur Bestimmung der Ähnlichkeit zweier Zeichenketten benötigt, damit gleichermaßen falsch eingetippte Wörter erkannt werden können.

Editier-Link in Popup Fenster von Markierungen bezüglich Profilinformationen

Die aktuelle Browser-Erweiterung bietet die Möglichkeit, mit dem Mauszeiger über eine Textmarkierung zu fahren und dadurch ein Popup aufzurufen, das Informationen über die Gründe der Markierung enthält. Falls die Markierung durch das Modul zur Auffindung von Profilinformationen im Text (*ProfileInfoCompareModule*) gesetzt wurde, wäre ein Link sinnvoll, um die entsprechende Profilinformation direkt anzupassen.

Sammeln von geeigneten Trainingsdaten für den Klassifizierer

Die Genauigkeit der Einschätzung eines Klassifizierer ist stark abhängig von seinen Lerndaten. Aktuell gibt es die Möglichkeit, im Popup der Erweiterung den Naive Bayes Klassifizierer zu trainieren, jedoch ist das lediglich ein Entwickler-Tool, um den Klassifizierer zu testen. Für die Zukunft sind Überlegungen sinnvoll, wie eine umfangreiche Sammlung qualitativer Trainingsdaten erlangt werden könnte.

Eine Idee ist es, in den sozialen Netzwerken ein Spiel zu etablieren. Dabei würden auf der Startseite neben jeder angezeigten Statusmeldung von anderen Benutzern zwei Buttons erscheinen: ”Kritisch” und ”Nicht kritisch”. Ein Benutzer kann durch Klicken eines dieser Buttons eine Einschätzung abgeben, ob er eine bestimmte Meldung privatsphärenkritisch findet oder nicht. Dadurch trainiert der Benutzer seinen Klassifizierer mit Hilfe der Realdaten seiner Freunde. Um dem Ganzen einen spielerischen Charakter geben zu können, wäre es möglich, jede Woche Punktelisten zu veröffentlichen. In diesen würde stehen, wer die besten (unkritischsten) Meldungen schreibt oder am häufigsten bewertet. Dieses Feature der Ranglisten müsste man aber bezügliche *Privacy* evaluieren.

Ob die dadurch gesammelten Daten (partielle Statusmeldungen) in einen von allen Benutzern geteilten Lerndaten-Pool (ähnlich zu Spam Filter bei großen E-Mail-Anbietern) gespeichert werden sollten, ist unklar, da sich durch das Speichern Datenschutz-Probleme ergeben würden. Eine Option wäre sinnvoll, bei der man lediglich seinen eigenen Klassifizierer verwenden könnte oder aber den ”durchschnittlichen” aller Benutzer.

Ansätze für natürliche Sprachverarbeitung

Wie im Fazit erwähnt wäre es sinnvoll, die Erweiterung mit natürlicher Sprachverarbeitung auszustatten, um so zum Beispiel die Semantik eines Satzes verstehen zu können. Jedoch sind aktuelle Ansätze und Biblotheken in einer Browser-Erweiterung nicht in Echtzeit einsetzbar (siehe Kapitel 4.3, Verwendung von Bibliotheken zur Sprachverarbeitung).

Daher müssten weitere Verfahren analysiert und entwickelt werden, insbesondere in Hinsicht auf die Anwendbarkeit in Echtzeit.

Unterstützung für weitere soziale Netzwerke und andere Browser

Es wäre sinnvoll, die Browser-Erweiterung so weiterzuentwickeln, dass neben Facebook weitere soziale Netzwerke, wie zum Beispiel Google+, unterstützt werden. Der modulare Code erlaubt, dass zum Beispiel die Überprüfung und Speicherung von Daten wiederverwendet werden kann. Ebenso die Visualisierung der Überprüfungsergebnisse kann zum größten Teil übernommen werden. Neu hinzukommen müsste ein Ausleseverfahren für die Profildaten des neuen sozialen Netzwerks. Das ist oftmals komplex, da die Seitenstrukturen, die Profildaten enthalten, erst manuell analysiert und dann entsprechende Routinen zum automatisierten Auslesen entwickelt werden müssen.

Des Weiteren wäre es nützlich, die Erweiterung für andere Browser anzubieten, allerdings sind die Strukturen für Erweiterungen in anderen Browsern oftmals sehr unterschiedlich und dementsprechend müssten möglicherweise einige Anpassungen vorgenommen werden.

Abbildungsverzeichnis

Listings

8 Anhang

Übersicht über die Ordner- und Dateistruktur der Browser-Erweiterung

- **background**
  - background.html - Bindet benötigte JS-Dateien ein und anschließend das Background-Script
  - background.js - Enthält das Background-Script, welches die Tabs verwaltet und eine *Database*-Instanz verwaltet
- **content**
  - AnalyzePipeline
    - Modules
      - CitiesModule.js - Definiert die Klasse *CitiesModule* (erbt von *SimpleWordCompareModule*) und dient dem Auffinden von Städtenamen im Text
      - ProfileInfoCompareModule.js - Enthält die Klasse *ProfileInfoCompareModule*, welche den Text nach Profilinformationen durchsucht und markiert
      - LevenshteinWordCompareModule.js - Alternative Implementierung für *ProfileInforCompareModule* mit zusätzlicher Erkennung von Tippfehlern durch die Levenshtein-Distanz
      - PronounModule.js - Beschreibt die Klasse *PronounModule*, die Pronomen im Text findet und markiert
      - RegExModule.js - Beschreibt die Klasse *RegExModule*, welche beliebige reguläre Ausdrücke im Text findet und markiert
    - AnalyzeModule.js - Enthält die abstrakte Basis-Klasse *AnalyzeModule*, von der alle Module erben
    - AnalyzePipeline.js - Enthält die *AnalyzePipeline*-Klasse die für das Zusammenschalten der verschiedenen Module zuständig ist
    - Highlights - Enthält die *Highlight*-Klasse, die Markierungs-Informationen enthält.
    - SimpleWordCompareModule.js - Abstrakte Implementierung eines Moduls, die bestimmte Wörter im Text sucht
  - css - Enthält die CSS-Dateien um das Aussehen festzulegen
  - helpers
    - FacebookCrawler.js - Enthält die *FacebookCrawler*-Klasse mit einigen Methoden zum Auslesen spezifischer Informationen aus Facebook
    - LoadingIndicator.js - Verwaltet die ”Laden”-Anzeige, die beim Profil analysieren angezeigt wird
  - profile - Dieser Ordner enthält alle Content-Scripts, die in Seiten mit Profil-Informationen eingefügt werden
    - analyzeEditInfos.js - Liest die Daten der Profil-Info-Bearbeiten-Seite
    - analyzeFavorites.js - Liest die Daten der Favoriten-Seite
    - analyzeFriends.js - Liest die Daten der Freunde-Seite
    - readUsername.js - Liest den Benutzernamen auf der Startseite von Facebook aus.
  - postChecker.js - Diese Datei wird auf der Facebook-Startseite eingebunden und kümmert sich primär um optische Darstellung der Überprüfungsergebnisse.
- **img** - Ordner der alle, von der Erweiterung genutzten, Bilder enthält.
- **models**
  - NaiveBayes
    - Bayes.js - Enthält eine Naive-Bayes-Implementierung (Klasse *Bayes*) mit Lern -und Schätzfunktionen

  - PrivacySettings.js - Definiert eine Klasse $PrivacySettings$, die einen numerischen Wert (Privatsphäre-Einstellung) enthält und eine $violatesSetting$-Methode zur Überprüfung, ob eine andere Einstellung verletzt wird
  - ProfileInformation.js - Beschreibt eine Klasse $ProfileInformation$, die einen Wert, eine $PrivacySetting$ sowie eine Quelle enthält

- **persistency**
  - Database.js - Definiert die Klasse $Database$ für Lese -und Schreibzugriffe auf die Datenbank
- **vendor** - Enthält Libraries (jQuery)
- popup.html - Definiert das Popup, das beim Klicken auf das Erweiterungs-Icon erscheinen soll
- popup.js - Beschreibt die Aktionen der Popup-Seite, also Bayes unterrichten und Profil analysieren

Literaturverzeichnis

[1] F. Belfort, A. Meier, and D. Zumstein. Geschäftsmodelle für Social Network Sites am Beispiel von Facebook, Seminararbeit, University of Fribourg, 2010.

[2] Danah Boyd and Eszter Hargittai. Facebook privacy settings: Who cares? *first monday*, 15, 2010.

[3] Bundesverfassungsgericht. Volkszählung: Recht auf informationelle Selbstbestimmung, Urteil v. 15. Dezember 1983, Az. 1 BvR 209, 269, 362, 420, 440, 484/83, 1983.

[4] Duden. Pronomen, http://www.duden.de/node/700716/revisions/1140535/view, aufgerufen am 15.01.2013, 2013.

[5] Union European. Technologien zum Schutz der Privatsphäre. 265(2003):1–4, 2007.

[6] Jeffrey Friedl. *Mastering regular expressions*. O'Reilly Media, Incorporated, 2006.

[7] Sebastian Funke. *Benutzerstudie zur Visualisierung von privatsphärenkritischen Nachrichten in sozialen Netzwerken*. Bachelor-thesis, TU Darmstadt, 2013.

[8] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[9] Michaela Geierhos. Minimale Editierdistanz nach Levenshtein, http://www.cis.uni-muenchen.de/~micha/praesentationen/rechtschreibkorrektur/Levenshtein.html, aufgerufen am 03.01.2013.

[10] Marco Ghiglieri, Hervais Simo, Michael Waidner, Universität Darmstadt, FB Informatik, and FG Sit. Technical Aspects of Online Privacy. Technical report, 2012.

[11] Forschungswerk GmbH. Studie: Social Communities. 2009.

[12] Tabreez Govani and Harriet Pashley. Student awareness of the privacy implications when using facebook. *Evaluation*, 17(Whelan):105–110, 2005.

[13] Häßler, U. Javascript RegExp oder GREP - Reguläre Ausdrücke, http://www.mediaevent.de/javascript/Javascript-Regulaere-Ausdruecke-1.html, aufgerufen am 02.01.2013, 2008.

[14] Google Inc. chrome.extension - Google Chrome API, http://developer.chrome.com/extensions/extension.html, aufgerufen am 05.02.2013.

[15] Nico Kirch. 584 Millionen Menschen sind Facebook-süchtig, http://www.socialmediastatistik.de/584-millionen-meschen-sind-facebook-suechtig/, aufgerufen am 11.02.2013, 2012.

[16] Christian Klaß. Facebook Privacy Scanner sorgt für mehr Privatsphäre, http://www.golem.de/1005/75158.html, aufgerufen am 16.01.2013, 2010.

[17] David D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval. *Machine Learning ECML98*, 1398:4–15, 1998.

[18] Lutz Prechelt. Vorlesung "Anwendungssysteme": Privatsphäre, http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-AWS-2011/31_Privatsphaere.pdf, aufgerufen am 24.01.2013.

[19] A McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. *Dimension Contemporary German Arts And Letters*, 752(1):41–48, 1998.

[20] Rainer Merkl and Stephan Waack. *Bioinformatik Interaktiv: Algorithmen und Praxis*. Wiley-VCH, 2003.

[21] Alexandru Nedelcu. How To Build a Naive Bayes Classifier, https://www.bionicspirit.com/blog/2012/02/09/howto-build-naive-bayes-classifier.html, aufgerufen am 21.01.2013, 2012.

[22] Focus Online. Party-Horror 2.0: Zwei Facebook-Partys eskalieren, http://www.focus.de/panorama/welt/party-horror-2-0-randale-und-muell-durch-facebook-partys_aid_833457.html, aufgerufen am 23.03.2013, 2012.

[23] Thomas Paul, Martin Stopczynski, Daniel Puscher, Melanie Volkamer, and Thorsten Strufe. C4PS - Helping Facebookers Manage Their Privacy Settings. Technical report, 2012.

[24] Daniel Puscher. Facebook privacy watcher, http://www.daniel-puscher.de/fpw/, aufgerufen am 16.01.2013, 2012.

[25] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[26] Nicolae Surdu and David Rodrigues. Style text inside textarea like facebook does, http://stackoverflow.com/questions/8438371/style-text-inside-textarea-like-facebook-does, aufgerufen am 04.12.2012, 2011.

[27] The jQuery Foundation. jquery: The write less, do more, javascript library, http://jquery.com, aufgerufen am 27.12.2012.

[28] Amin Tootoonchian and Stefan Saroiu. Lockr: better privacy for social networks. In *5th international conference on Emerging networking experiments and technologies*, pages 169–180, 2009.

[29] Verein SELFHTML. Window Objekt - Javascript Objektreferenz, http://de.selfhtml.org/javascript/objekte/window.htm, aufgerufen am 02.01.2012.

[30] Verein SELFHTML. focus() Methode - Javascript Objektreferenz, http://de.selfhtml.org/javascript/objekte/elements.htm#focus, aufgerufen am 04.12.2012, 2012.

[31] W3C. Document Object Model (DOM) Level 3 Events Specification, http://www.w3.org/TR/DOM-Level-3-Events/#event-type-DOMNodeInserted, aufgerufen am 26.01.2013, 2012.

[32] W3C. Selectors Level 3, http://www.w3.org/TR/selectors/, aufgerufen am 26.01.2013, 2012.

[33] W3C. Indexed database api, http://www.w3.org/TR/IndexedDB, aufgerufen am 02.01.2013, 2013.

[34] Yang Wang and Alfred Kobsa. Privacy-enhancing technologies. *Social and Organizational Liabilities in Information Security*, pages 203–227, 2006.

[35] Yang Wang, Pedro Giovanni Leon, Gregory Norcie, Alessandro Acquisti, and Lorrie Faith Cranor. "I regretted the minute I pressed share": A Qualitative Study of Regrets on Facebook. *Methodology*, 28:1, 2011.

[36] Jeffrey Way. From jQuery to JavaScript- A Reference | Nettuts+, http://net.tutsplus.com/tutorials/javascript-ajax/from-jquery-to-javascript-a-reference, aufgerufen am 27.12.2012, 2012.

[37] Wikipedia. Privacy-enhancing technologies, http://en.wikipedia.org/wiki/Privacy-enhancing_technologies, aufgerufen am 24.01.2013, 2012.

[38] Wikipedia. Usage share of web browsers, http://en.wikipedia.org/wiki/Usage_share_of_web_browsers, aufgerufen am 23.03.2013, 2013.

[39] Lauren Wood. Programming the web: the w3c dom specification. *Internet Computing, IEEE*, 3(1):48–54, 1999.