Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Medieninformatik

Visuelle Segmentierung von Webseiten mit Hilfe von Crowdsourcing

Bachelorarbeit

Florian Kneist
geb. am: 26.12.1992 in Mülheim an der Ruhr

Matrikelnummer 110547

1. Gutachter: Prof. Dr. Benno Stein
2. Gutachter: Dr. Andreas Jakoby

Datum der Abgabe: 18. April 2016

Erklärung

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

Weimar, 18. April 2016

.....
Florian Kneist

Zusammenfassung

Die automatische Segmentierung von Webseiten erlaubt es, den Inhalt von Webseiten auf die Kategorisierung vorzubereiten. Nach der Kategorisierung kann die gezielte Suche nach Informationen eingegrenzt werden. Suchmaschinen und Methoden des Information Retrievals profitieren von einer systematischen Reduzierung des zu durchsuchenden Datenvolumens, da unnützes Markup und Noise im Dokument entfernt werden. Eine zweite Motivation ist das Aufbereiten der Webseite zur Darstellung auf kleinen Geräten. Außerdem profitieren Screenreader davon, wenn sie zuordnen können welchem Zweck ein Textblock dient.

Wir entwickelten eine Pipeline zur manuellen Segmentierung von Webseiten mit dem Ziel eine skalierbare Methode zu schaffen, mit der große Corpora segmentierte Webseiten erstellt werden können. Crowdsourcing bietet dabei den einfachen Zugang zu menschlichen Annotatoren. Deren Aufgabe besteht darin zusammengehörige Inhaltssegmente auf Webseiten zu identifizieren und zu markieren. Bisherige Corpora in diesem Feld sind relativ klein (≤ 600 Webseiten), wie in Kapitel 2 beschrieben und enthalten veraltete Webseiten. Der Zugang zu großen Corpora ist für die Entwicklung und den Vergleich von automatischen Methoden notwendig.

Zur Vorbereitung der Segmentierung wurden die Webseiten heruntergeladen und Screenshots von ihnen erstellt. Bei dem Archivierungsvorgang legten wir Wert darauf, Webseiten mit dynamisch generiertem Inhalt reproduzierbar zu speichern. Die Screenshots der archivierten Seiten werden von den Annotatoren durch ein von uns erstelltes Web-Interface segmentiert. Segmente können bei dieser Aufgabe Navigationselemente, Absätze zu verschiedenen Themen oder nutzergenerierter Inhalt wie Kommentaren und Bewertungen sein; ebenso Grafiken, Videos oder interaktive Inhalte.

Gestalterisch liegen den Webseiten oft rechteckige Komponenten zu Grunde. Man denke an Grid-Systeme in Front-End-Frameworks, Tabellen, Listen, Grafiken, Menüs, Videos. Unser Ansatz beschränkt sich auf das Finden von rechteckigen Segmenten. Optisch als zusammengehörig erkennbare Inhalte sind meist auch inhaltlich zusammengehörig. Der visuelle Aufbau von gerenderten Webseiten kann also einen guten Aufschluss über die Gruppierung des Inhalts einer Webseite geben. Menschen haben es hier leicht, durch visuelle Hinweise wie Leerzeilen, Positionierung und Farbe solche Gruppen schnell zu bilden. Für das Segmentieren durch Menschen spricht, dass Webseiten schließlich für Menschen gestaltet werden.

Der Segmentierungsvorgang kann durch unser Web-Interface überwacht werden. Das Arbeiter Interface zur Segmentierung zeichnet die Interaktionen des Annotators mit dem Interface, sowie das Endresultat auf. Jede Seite kann von verschiedenen Annotatoren bearbeitet werden und diese verschiedenen Lösungsvorschläge können danach vereinheitlicht werden. Die Ausgabe wären gruppierte Fragmente der Seiten. Der letzte Analyse-Schritt zur Schaffung eines Korpus bestünde darin, auf die Knoten in der DOM-Struktur zu schließen, die den einzelnen Segmenten zu Grunde lagen. Dies ist für die Extrahierung von Inhalten und deren weitere Verarbeitung notwendig.

Inhaltsverzeichnis

Kapitel 1

Einführung

In dieser Arbeit wird ein System zur Ansammlung und Verarbeitung von manuellen Webseiten-Segmentierungen vorgestellt. Es ist für die Annotation durch Arbeiter der Plattform Amazon Mechanical Turk ausgelegt. Die entwickelten Bestandteile umfassen eine Lösung zur Archivierung von Webseiten, einem Interface für die Arbeiter und einem Interface für die Arbeitgeber zur Bewertung der Ergebnisse.
Für das Thema Webseiten-Segmentierung tat sich ein großer Design-Space auf, bei dem viele Entscheidungen bezüglich des Vorgehens getroffen werden mussten. Die erste Fragestellung war, wie den Arbeitern die archivierten Webseiten präsentiert werden sollten. Als Lösung wurden Screenshots der archivierten Seiten erstellt und diese in überlappende Abschnitte unterteilt. Aufgrund der unterschiedlichen Länge von Webseiten bieten die Screenshot-Abschnitte den Vorteil, dass Webseiten portioniert werden können und den Arbeitern Aufgaben von vergleichbarem Umfang präsentiert werden können. Zudem kann die versehentliche Interaktion mit der zu segmentierenden Webseite ausgeschlossen werden.
Die Annotationsmethoden dieser Arbeit beschränken sich auf rechteckige Segmente, da Webseiten meist aus rechteckigen Bestandteilen aufgebaut sind. Als Bezugsquelle für die zu archivierenden URLs wurde in dieser Arbeit, aufgrund seiner Größe und freien Verfügbarkeit, der Common Crawl herangezogen. Durch die Umsetzung des Arbeitgeber-Interfaces als Browsererweiterung wurde Flexibilität und Systemunabhängigkeit erreicht. Diese Erweiterung visualisiert die Segmentierungsvorschläge der Arbeiter und erleichtert somit deren Bewertung. Bei dem vorbereitenden Thema Archivierung war die visuell originalgetreue Speicherung und Rekonstruierbarkeit von Webseiten wichtig. Hier wurden verschiedene Lösungen getestet und schließlich eine Kombination aus Headless-Browser und Proxy-Server verwendet um auch dynamisch geladenen Inhalt zu erfassen.

Kapitel 2

Related Work

Das Problem der Webseiten Segmentierung wurde bereits aus zahlreichen Perspektiven untersucht, unter der Einbeziehung verschiedener Bestandteile von Webseiten. Nennenswert sind hier die HTML-Struktur, das visuelle Erscheinungsbild und der Textinhalt von Webseiten.
Cai et al. [2003] stellen einen Algorithmus vor, der versucht die visuelle Wahrnehmung des Internetnutzers nachzubilden, einen sogenannten Vision-based Page Segmentation Algorithm (VIPS). Das Ziel der Segmentierung ist es, nichtüberlappende Blöcke, die durch horizontale und vertikale Separatoren getrennt sind, zu bilden. Idealerweise möchte man inhaltlich möglichst kohärente Blöcke finden. Der letzte Schritt ist das Strukturieren dieser Blöcke. Der DOM-Baum wird top-down iterativ nach Separatoren untersucht. In jedem Durchgang wird die Seite feiner granuliert unterteilt. Es werden dabei 13 unterschiedliche heuristische Regeln verwendet, um zu entscheiden, ob ein Knoten geteilt werden soll. Die Regeln werden in unterschiedlichen Kombinationen auf ein festes Set von HTML-Tags angewendet. So sollen beispielsweise Knoten deren Kinder überwiegend Text-Knoten sind, nicht geteilt werden oder ein Knoten soll geteilt werden, wenn einer seiner Kind-Knoten das HTML-Element „<hr>“ enthält. Tags zur Erstellung von Tabellen finden dabei besonders große Beachtung.
Früher wurden Tabellen oft auch zur Gestaltung von Webseiten verwendet, worin diese Gewichtung begründet sein dürfte. Die feste Verzahnung mit HTML-Tags und „bad practices“ dieser Zeit machen den Algorithmus aus heutiger Sicht etwas unflexibel beziehungsweise anpassungsbedürftig. Andere Kriterien, die in die Feststellung der Unterschiedlichkeit von Webseitenabschnitten einfließen, sind Hintergrundfarbe und Textdekoration. Das Identifizieren und Wegschneiden von Blöcken ohne Inhalt ist möglich. Auch haben Cai et al. [2003] ihren VIPS von Menschen evaluieren lassen. 600 Internetseiten wurden durch den VIPS-Algorithmus segmentiert und alle Ergebnisse von fünf Menschen bewertet. Es gab dabei die vier Kategorien perfekt, befriedigend,

mittelmäßig und schlecht, von denen für uns vor allem die dritte interessant ist (circa sechs Prozent). Es handelte sich um Blöcke, die unzureichend visuelle Trennungsmerkmale für den Algorithmus aufwiesen, deren Trennbarkeit von Menschen aber erkannt werden konnte. Fälle aus dieser Kategorie möchten wir mit unserem Ansatz erkennen. Cai et al. [2003] beschreiben visuelle Ansätze der Segmentierung, als gut geeignet für Detektion und Filterung von Noise.

Vineel [2009] bezieht die visuelle Gestaltung nicht in seinen Ansatz der Segmentierung mit ein, sondern arbeitet ausschließlich mit dem HTML-Quelltext der jeweiligen Seite. Der Autor charakterisiert Knoten im DOM-Baum durch Ihre Größe und ihre Entropie. Als Größe wird die Anzahl der unterschiedlichen Worte natürlicher Sprache pro Knoten und seiner Kinder beschrieben und soll die relative Wichtigkeit eines Knotens angeben. Mit der Entropie beschreibt er die Verteilung von verschiedenen Tag-Namen beispielsweise *table*, *tr* und *td* in den Unterbäumen. Laut Annahme entsprechen wiederkehrende Muster in den Tag-Namen einer Verteilung nahe einer Gleichverteilung, was für eine hohe Entropie sorgt. Die Entropie hängt in dem Ansatz ausschließlich von Tag-Namen ab, nicht vom Textinhalt. Weiterhin besteht die Annahme, dass einzelne Tags meistens nur bei einer gewissen Tiefe im DOM-Baum eingesetzt werden. Zum Beispiel Hyperlink-Elemente, also a-Tags nur in der Nähe von Blattknoten.
Das Vorgehen des Algorithmus lässt sich grob als ein Stutzen des DOM-Baumes beschreiben bis nur noch Knoten übrig bleiben, die als Segment gewertet werden. In einem 2D-Feature-Space aus Knotengröße und Entropie wird eine Zielregion festgelegt, begrenzt durch Schwellwerte. Die Autoren haben 400 Seiten untersucht und menschlich evaluiert. Probleme traten bei per Skript generierten Inhalten auf, die durch reine Analyse des HTML-Quellcodes nicht gefunden werden konnten. Ebenso schlägt der Algorithmus bei flachen HTML-Hierarchien fehl, wozu der Autor im Ausblick das Zusammenführen von Knoten vorschlägt. Die größte Einschränkung des Ansatzes ist, dass Segmente immer genau einem DOM-Knoten entsprechen müssen.

Kohlschütter and Nejdl [2008] präsentieren einen linguistischen Ansatz. Die Autoren argumentieren gegen die reine Untersuchung der HTML-Struktur auf Basis der Art der HTML-Tags, mit der Begründung, dass viele Tags keine eindeutigen Indikatoren für das Separieren von Inhalten seien. Sie betrachten Textblöcke als die atomare Einheit der Untersuchung und sehen HTML-Tags nur als potentielle Positionen für die Trennung, unabhängig von der Art des jeweiligen Tags. Je nach Unterschieden der statistischen Eigenschaften der Textblöcke werden sie als Separator bestätigt oder außer Acht gelassen. Ein Tag wird als Separator identifiziert, wenn es einen Wechsel im Textfluss gibt, etwa

von kurzen Phrasen zu langen Sätzen. Es wird auf die Parallelen zur Analyse von Schreibstilen hingewiesen. Sie beschreiben ihren Ansatz als die Lösung eines eindimensionalen Textdichte-Problems.

Da diese Arbeit die Vorbereitung eines großen Korpus von segmentierten Webseiten darstellt, folgt eine Übersicht über die Größe der Testsets von Webseiten der erwähnten Ansätze.

Ansatz von	Umfang Webseiten
Vineel [2009]	400
Cai et al. [2003]	600
Kohlschütter and Nejdl [2008]	111

Wir stellen einen Ansatz vor, der das, was Cai et al. [2003] automatisch modelliert, von Menschen ausführen lässt. Segmente sind in unserem Ansatz ebenfalls rechteckig, sind aber nicht auf die Repräsentation durch nur einen DOM-Knoten beschränkt. Wir schlagen die Repräsentation als eine Menge von DOM-Fragmenten vor. Der manuelle visuelle Ansatz ist besonders dann von Vorteil, wenn sich die Tag-Struktur von der optischen Gliederung unterscheidet. Der Ansatz ist unabhängig von Tag-bezogenen Heuristiken wie sie Vineel [2009] beschreibt. Die Positionierung der erwähnten Ansätze hinsichtlich Quelltext-Analyse und visueller Analyse zeigen wir in Abbildung 2.1. In Bezug auf die benötigte Zeit und Skalierbarkeit liegt der manuelle Ansatz im Hintertreffen gegenüber allen erwähnten automatischen Verfahren.

Nahe verwandt mit dem Problem der Webseiten Segmentierung ist die Content-Extraction. Kovačič [2012] gibt hier eine gute Übersicht und vergleicht akademische und kommerzielle Ansätze. Gottron [2008] bietet eine sehr umfassende Darstellung bestehender Methoden.

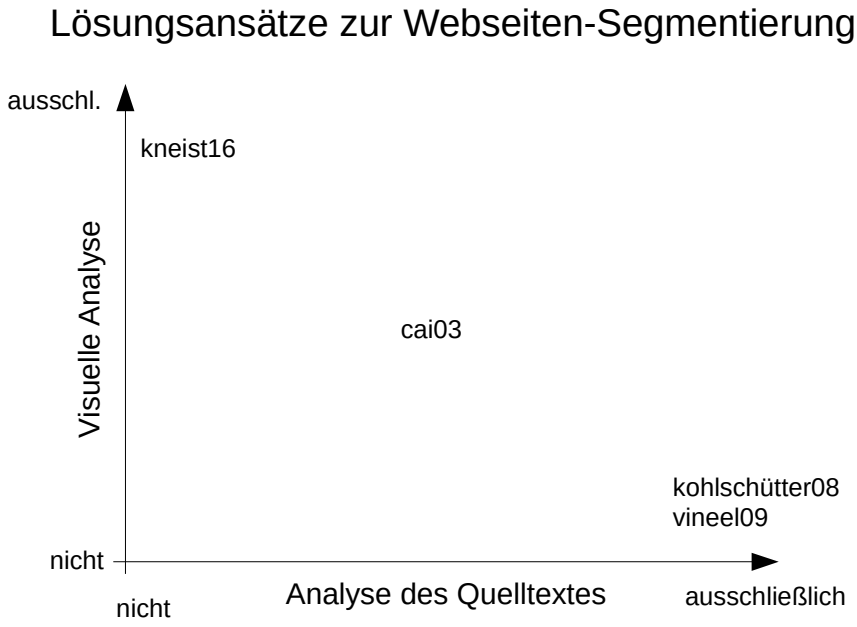

Abbildung 2.1: Positionierung der vorgestellten Ansätze zueinander in Bezug auf Analyse der Quelltextes und visuelle Analyse bei der Segmentierung.

Kapitel 3

Benutzerschnittstelle

Auf der Plattform Amazon Mechanical Turk (AMT) können von Arbeitgebern Aufgaben (HITs, Human Intelligence Tasks) in Auftrag gegeben werden. Bearbeitet werden diese von den dort registrierten Arbeitnehmern. Typische HITs sind Umfragen oder das Kategorisieren von Bildern. Generell ist jede Aufgabe geeignet, die sich in einem Browser erledigen lässt. Ein Auftrag kann durch eines der verfügbaren Templates, durch ein selbsterstelltes Template oder durch den Verweis auf eine eigene URL generiert werden. Die Bezahlung, die verfügbare Bearbeitungszeit und die Anzahl der Bearbeitungen pro Aufgabe lassen sich vom Requester frei bestimmen.
Fertiggestellte Aufträge lassen sich auf AMT per Webinterface in Tabellenform einsehen oder als CSV-Datei (Comma Separated Values) herunterladen. Auch die Entscheidung über die Bezahlung der Arbeiter findet in dieser Ansicht statt.
Im Rahmen dieser Arbeit wurden zwei Benutzerschnittstellen konstruiert: Eine Schnittstelle für die Arbeitnehmer auf AMT zur Segmentierung der Screenshot-Abschnitte und eine Schnittstelle für uns, als Auftraggeber zur effizienten Bewertung der Ergebnisse.
Das Arbeiter-Interface musste den Zweck erfüllen, komfortabel Rechtecke auf Screenshots-Abschnitten zu zeichnen, die später zu Segmentierungen verarbeitet werden, zu sehen in Abbildung 3.1, Schritt 2. Außerdem werden möglichst viele Informationen über den Browser des Users gesammelt. Durch das Erstellen und Testen von Bookmarklets mit verschiedenen Auswahltechniken gelangten wir zur Entscheidung für frei zeichenbare Auswahlrechtecke. Das Speichern von eingestellter Sprache und Zeitzone hilft, sich ein Bild über die Herkunft der Arbeiter zu machen. Durch das Speichern von technischen Details wie Bildschirmauflösung und Betriebssystem erhält man Aufschluss über die Ausstattung der Arbeiter, was bei der Erstellung zukünftiger HITs helfen kann.

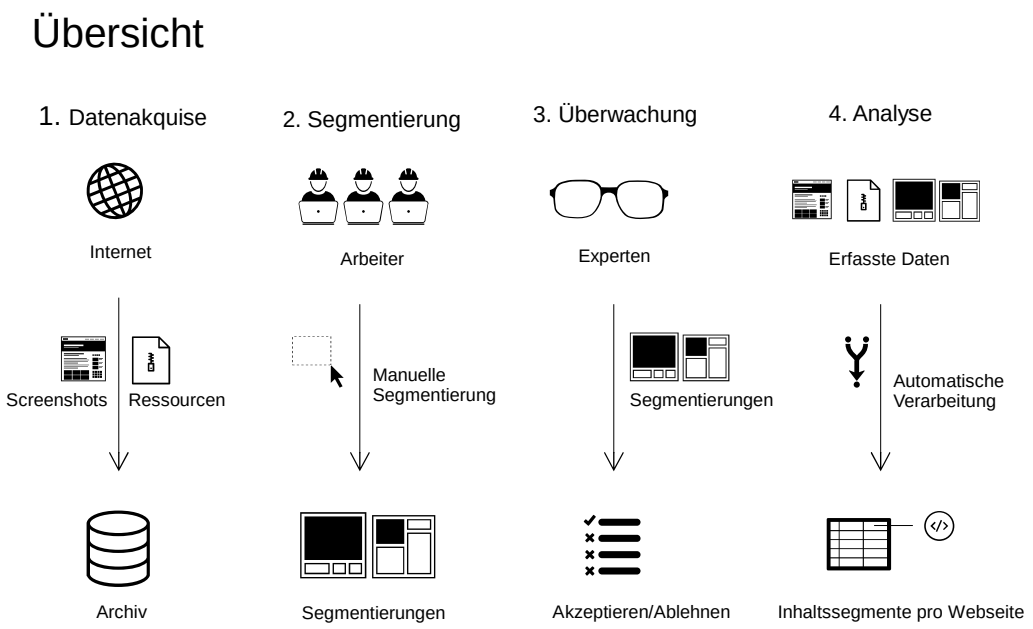

Abbildung 3.1: Eine Übersicht über die Phasen unserer Arbeit. Das Arbeiter-Interface wird in Punkt 2 (Segmentierung) verwendet und das Arbeitnehmer-Interface in Punkt 3 (Überwachung).

Das Arbeitgeber-Interface stellte die gesammelten Daten visuell dar und bot einen Überblick über den jeweiligen Arbeitnehmer; nennenswert sind hierbei historische Informationen über seinen Arbeitserfolg (Approval Rate), die Bearbeitungszeit und der Vergleich von Stichproben mit einem von uns erstellten "Goldstandard".

3.1 Arbeiter Interface

3.1.1 Entstehung

Als erste Inspiration für das Arbeiterinterface dienten uns ein Feature der Entwicklertools von Google Chrome (Google [2016]). Im Abschnitt "Inspecting Elements" sieht man den Modus beschrieben, in dem per Maus auf der gerenderten Webseite Elemente ausgewählt und farblich durch eine transparente Box gekennzeichnet werden können. Diese Rechtecke sind die Visualisierung

der Boundingboxen des entsprechenden DOM-Knotens. Wir haben dieses Verhalten in einem ersten Versuch per Bookmarklet nachgebildet. Eine Webseite wird aufgerufen und nach Aktivierung des Bookmarklets wird jeweils die Boundingbox, die direkt unter der Maus liegt, markiert.
Auf unseren Ansatz bezogen hat diese Methode als Markierungsart zwei Nachteile: Die Webseite muss online verfügbar sein, also aus Gründen der Reproduzierbarkeit von uns gespiegelt werden und sie schränkt die Freiheit ein, Segmentierungen frei festzulegen. Es kam dabei oft zu Hervorhebung von unvermuteten, nicht sichtbaren Elementen, was Nutzer verwirren und stören könnte. Weiterhin kann es sein, dass ein Segment nicht nur aus einem DOM-Knoten besteht oder dass ein Segment nur aus einem Teil eines DOM-Knotens besteht; Vineel [2009] weisen in Abschnitt VII auf eben diese Einschränkung ihres Ansatzes hin, dass Segmente nur einem gesamten DOM-Knoten zuordenbar seien. Aus diesem Grund entwickelten wir ein Interface, dass es erlaubte Rechtecke per Maus frei auf Screenshot-Abschnitten zu zeichnen. Vergleichbar ist diese Methode mit dem Auswahlwerkzeug gängiger Bildbearbeitungsprogramme oder der Funktion "imrect" von MATLAB Mathworks [2016].
Die Verwendung von MATLAB schied für unseren Ansatz aus, da die HITs im Browser bearbeitet werden sollten. Die Übertragung der arbeitnehmergenerierten Rechteckdaten auf DOM-Fragmente findet später durch eine automatische Methode statt.
Den Arbeitnehmern sollte es leicht fallen, zusammengehörige Segmente zu finden indem sie bewusst oder unterbewusst einigen Prinzipien der "Gestalt Theory" (Wertheimer [1923]) folgen. Nennenswert sind "Proximity", "Closure" und "Similarity". Übertragen auf unsere Problemstellung, erklären sich diese wie folgt:

- "Proximity": nahe beieinander liegender Inhalt bildet ein Segment.
- "Closure": gemeinsam umrandeter Inhalt bildet ein Segment.
- "Similarity": ähnlich gestalteter Inhalt bildet ein Segment.

Ein HIT enthält zehn Screenshot-Abschnitte und wird zehnmal bearbeitet. Pro Screenshot erhalten wir also zehn Segmentierungsvorschläge. Schematisch ist die Bearbeitung des gleichen Screenshot-Abschnitts in Abbildung 3.2 dargestellt.

Neben den Segmentierungsvorschlägen in Form von Rechteckdaten speichern wir einige weitere Informationen über den Browser der Arbeitnehmer gespeichert. Dazu zählen Auflösung, Betriebssystem, Zeitzone, Browser-Plugins und eingestellte Sprache. Hierzu wurde die Bibliothek "ClientJS" verwendet

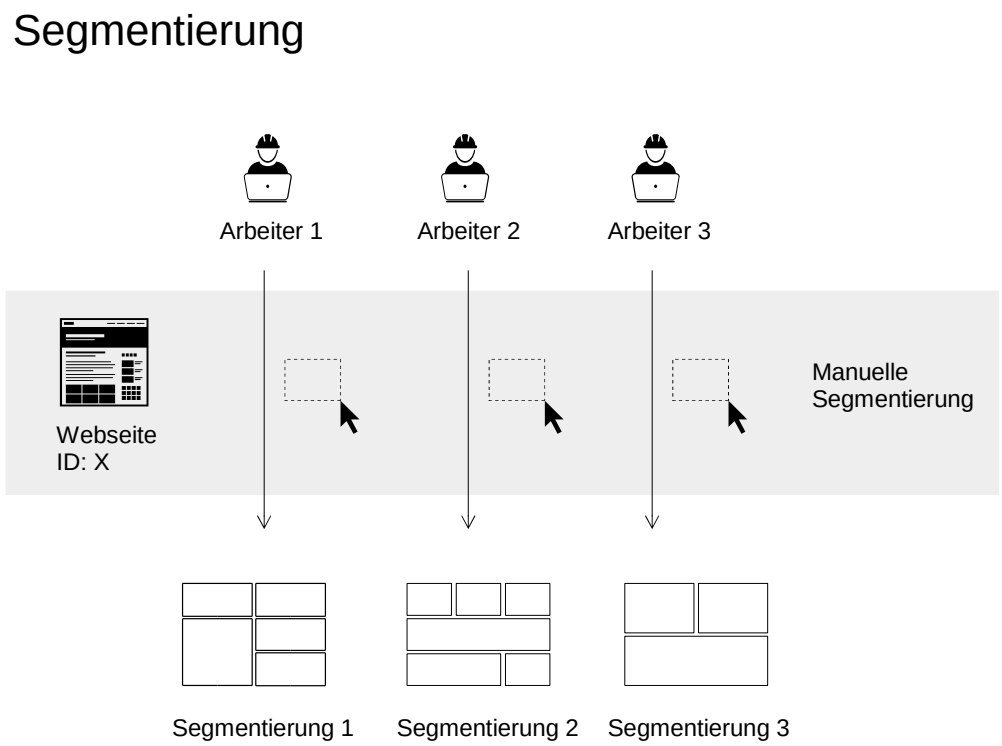

Abbildung 3.2: Schematische Darstellung des Annotationsprozesses eines Screenshot-Abschnittes. Verschiedene Arbeiter erhalten den gleichen Screenshot-Abschnitt zur Segmentierung und liefern voneinander abweichende Ergebnisse.

(Spirou [2016]). Was durch das Arbeiter-Interface erstellt wird kann schon wenige Augenblicke nach Abgabe des HITs im Arbeitgeber-Interface begutachtet werden, wie in Abschnitt 3.2 beschrieben wird.

3.1.2 Einbindung in AMT

Das Worker-Interface für die AMT-Arbeiter besteht aus einer Anleitung (Abschnitt B, Abbildung 3.3), einem Arbeitsbereich und einem Feedbackbereich für Kommentare. Die Arbeiter werden durch eine Arbeitsanweisung sowie Positiv- und Negativbeispiele auf den Task vorbereitet (Abschnitt B2, Abbildung 3.3). Darunter können sie mit der Maus Rechtecke auf dem Screenshot im Arbeitsbereich zeichnen. Die Erstellung der Rechtecke ist dem Selektionswerkzeug gängiger Bildbearbeitungssoftware nachempfunden. Die Rechtecke können verschoben, in der Größe angepasst und gelöscht werden. Die Funktionalität zur Interaktion stammt größtenteils aus jQueryUI. Zur optischen Aufbereitung des Interfaces wurde Bootstrap verwendet, was standardmäßig in AMT-Templates eingebunden wird. Nachdem der Arbeiter "Accept HIT" (Abschnitt A, Abbildung 3.3) klickt, kann er mit der Bearbeitung des HITs beginnen.

3.2 Arbeitgeber Interface

In der von Amazon bereitgestellten Weboberfläche von AMT werden die Ergebnisse jedes Tasks lediglich in Text und Tabellenform dargestellt. Unser angepasstes Arbeitgeber-Interface ist daher notwendig, um nutzergenerierte Daten zu visualisieren. Diese Visualisierung dient der Akzeptierung bzw. Ablehnung eines Segmentierungsvorschlages. So wird verhindert, dass bewusster Betrug bei der Erstellung zu Bezahlung führt. Der Vergleich mit dem von uns erstellten Goldstandard wird im Arbeitgeber-Interface durchgeführt, so dass Arbeiter unter keinen Umständen Zugriff auf diese erhalten können.
Zur Umsetzung boten sich die Verwendung der offiziellen API oder die Entwicklung einer Browser-Extension als Optionen. Unter der Voraussetzung, dass die Informationen schon in Tabellenform im Browser verfügbar waren, haben wir eine Erweiterung entwickelt. Dafür sprach zudem, dass die zu visualisierenden Daten durch unser Arbeiter-Interface in einem Format übertragen werden, dass sich durch Webtechnologie leicht darstellen lässt, nämlich Position und Größe der Boundingboxen von den Auswahlrechtecken der Arbeiter. Was auf lange Sicht gegen eine Erweiterung spricht, ist die Abhängigkeit vom Design und Aufbau der AMT-Seite; API-Änderungen sind weniger wahrscheinlich als Anpassungen des Webdesigns, meist angekündigt und besser dokumentiert.

Das Arbeiter Interface auf Amazon Mechanical Turk

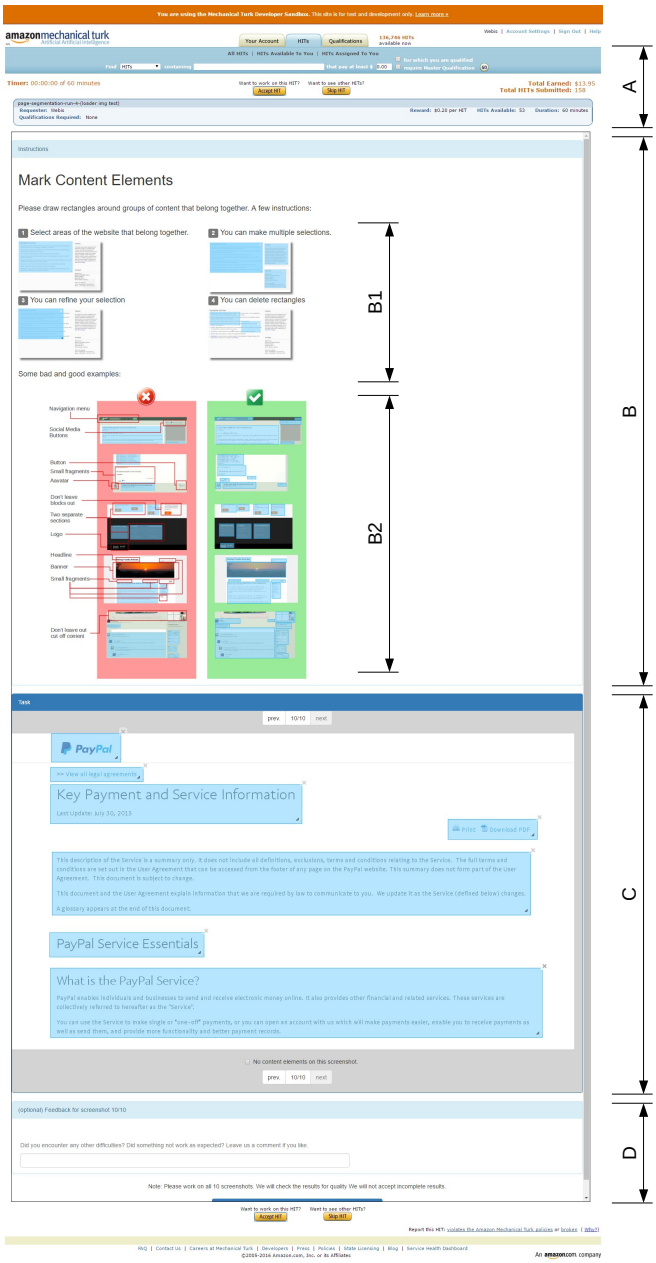

Abbildung 3.3: Screenshot des Arbeiter-Interfaces. Der HIT wird in Bereich A angenommen, Bereich B bietet eine Anleitung, in Bereich C wird die Segmentierung der Screenshot-Abschnitte vorgenommen und in Abschnitt D kann der Arbeiter optionales Feedback hinterlassen und den HIT schlussendlich übermitteln.

Die Browser-Erweiterung liest den Inhalt der Tabelle mit den HIT Ergebnissen dynamisch aus und bereitet die Informationen aussagekräftig auf. Pro Segmentierungsvorschlag wird dasselbe Bild angezeigt, das auch von dem Arbeiter bearbeitet wurde. Darauf werden die Rechtecke transparent eingezeichnet. Der Auftraggeber muss nun lediglich ein Häkchen setzen um Segmentierungsvorschläge zu akzeptieren oder abzulehnen. Zur Ablehnung haben wir die Auswahl von vorgefertigten Begründungen mit häufigen Fehlern oder einem frei beschreibbaren Textfeld eingebaut. Pro Screenshot sieht man die Bearbeitungszeit und einen optionalen Kommentar des Arbeiters.
Die Verwendung von beiden Interfaces haben wir in der sogenannten Sandbox von AMT getestet, die für Arbeiter sowie für Auftraggeber existiert. So kam der Goldstandard zustande, der in Abschnitt 3.3 beschrieben wird.

3.3 Qualitätssicherung durch Goldstandard

Um zu beurteilen, ob ein Arbeiter die Aufgabenstellung verstanden und ernsthaft bearbeitet hat, ermöglichen wir den Vergleich von Teilen seiner Abgabe mit Expertenlösugen, also von vorher manuell erstellten Segmentierungen. Dies dient der Qualitätssicherung der akquirierten Daten und soll die Frage beantworten: Hat der Arbeiter die Aufgabenstellung verstanden und ernsthaft bearbeitet? Wir segmentierten Screenshot-Abschnitte selbst und bildeten daraus eine Menge von Vergleichseinheiten, den Goldstandard. Jeder HIT, der zur Bearbeitung freigegeben wird besteht aus acht unbearbeiteten Screenshot-Abschnitten und zwei aus dem Goldstandard. Bei der Bewertung im Arbeitgeber-Interface errechnen wir ein Maß der Ähnlichkeit von Goldstandard und Arbeiter-Abgabe (Test-Outcome). Als Einheit dient die "Entity Detection Metric" (siehe Abschnitt 3.3.2). Wir segmentierten 113 Webseiten selbst, was zu 571 Screenshot-Abschnitten führte. Auf diese Weise entstanden für jeden Screenshot-Abschnitt drei Segmentierungsvorschläge.

3.3.1 Erzeugung des Goldstandards

Zuerst wurde eine Untermenge der archivierten Webseiten gebildet, die als Goldstandard dienen sollte. Wir wählten die visuell perfekt archivierten Einträge mit einer Mismatch-Rate von Null aus (siehe 4.1.2), weil diese auf jeden Fall ihren Weg in den Korpus finden sollten. Unter diesen sortierten wir manuell offensichtlich unbrauchbare Einträge aus, wie Webseiten, die als einzige Botschaft eine Fehlermeldung beinhalten. Von 182 visuell einwandfreien Einträgen gelangten wir so zu 113. Diese 113 Webseiten waren in 571 Screenshot-Abschnitte unterteilt. Wir haben diese Webseiten durch das gleiche Interface

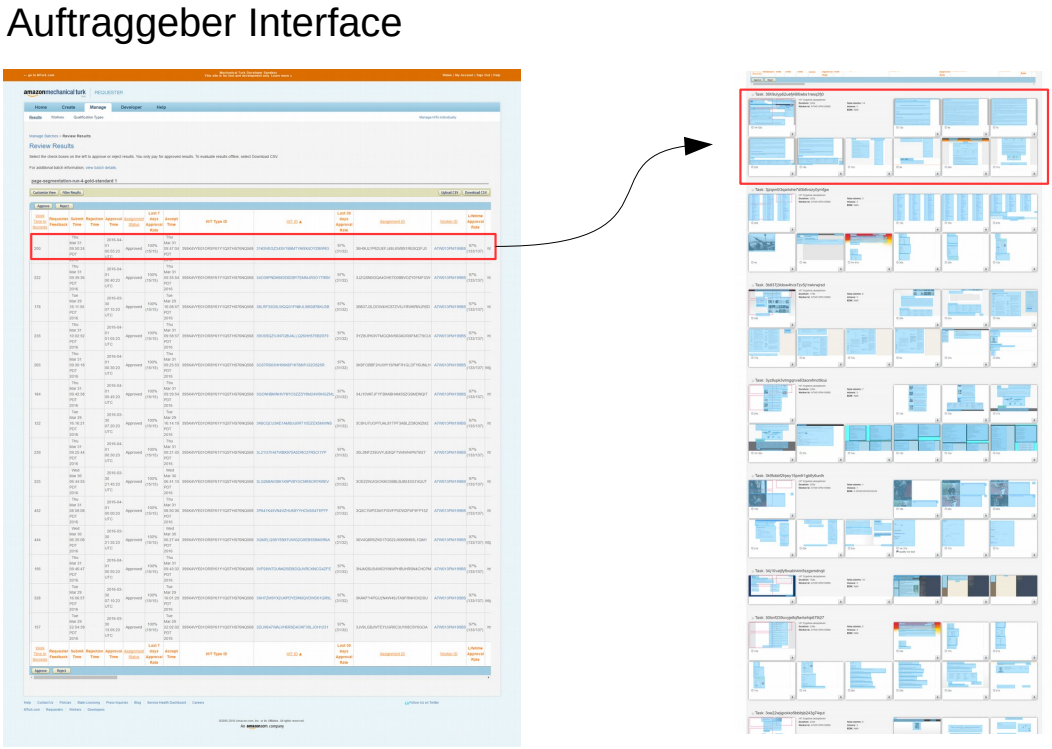

Abbildung 3.4: Das Auftraggeber Interface. Nach Aktivierung der Browser-Erweiterung werden die Segmentierungsvorschläge der Arbeiter dargestellt und ein Überblick über Arbeiter-Informationen gegeben. Anhand dieser Informationen wird über die Bezahlung des Auftrages entschieden.

segmentiert, wie es auch den Arbeitern präsentiert wurde. Der Vorgang geschah in der Arbeitnehmer-Sandbox und führte zu keinen Kosten.

3.3.2 Entity Detection Metric

Der Vergleich von Ground-Truth Segmentierungen mit den Arbeiter-Segmentierungen ist die Bestimmung der Ähnlichkeit von zwei Sets aus Rechtecken für die gleiche Zielfläche. Die Autoren Phillips et al. [1997] schlagen in Abschnitt 4.2 ein Protokoll vor dem wir folgten, das richtige Matches, Misses und False-Alarms einbezieht und die Accuracy der Abgaben bestimmt. Die Autoren beschäftigen sich mit dem Matching einiger Formen, von denen für unseren Ansatz nur Rechtecke relevant sind; von den erwähnten Autoren "Text-text Matching" genannt. Typische zweideutige Fälle, die bei dem Vergleich auftreten können sind, dass mehrere Goldstandard Elemente als ein Test-Outcome Element markiert wurde oder, dass anders herum mehrere Test-Outcome Elemente eigentlich ein Goldstandard Element bilden. Der erste Fall wird als One-to-many Match und der zweite Many-to-one Match bezeichnet. Korrekte Zuweisungen werden One-to-one Matches genannt.

Aus den drei zuletzt genannten Maßen mit Gewichtung entwickelten Antonacopoulos et al. [2007] die "Entity Detection Metric" (EDM) zum Vergleich von automatischen Segmentierungsverfahren von gescannten Textseiten. Über die "Detect Rate"

$$DetectRate_i = w_1 \frac{one2one_i}{N_i} + w_2 \frac{g\_one2many_i}{N_i} + w_3 \frac{g\_many2one_i}{N_i} \quad (3.1)$$

und die "Recognition Accuracy"

$$RecognAccuracy_i = w_4 \frac{one2one_i}{N_i} + w_5 \frac{d\_one2many_i}{N_i} + w_6 \frac{d\_many2one_i}{N_i} \quad (3.2)$$

gelangten sie zur EDM:

$$EDM_i = \frac{2DetectRate_i RecognAccuracy_i}{DetectRate_i + RecognAccuracy_i}. \quad (3.3)$$

Hier ist N_i die Anzahle der Ground-Truth Elemente von Rechtecks-Set i. Die Gewichte w_1 bis w_6 werden je nach Bedarf der Anwendung gesetzt und beschreiben Wichtigkeit, welche man der jeweiligen Art von Fehler für seinen Einsatzzweck zuspricht.

3.3.3 Ergebnisse

Beim Erstellen des Goldstandards fiel uns auf, dass wir schon untereinander abweichende Lösungen produzierten. One-to-many und Many-to-One Fälle traten häufig auf. Aus dieser Tatsache lässt sich der Bedarf für klare Regeln bei

der Segmentierung ableiten. Sind alle Kommentare beispielsweise ein Segment oder bildet jedes Kommentar ein eigenes? Wir versuchen die Unklarheiten in Beispielen zu klären und lassen eventuell mehrere Ergebnisse zu; wenn zum Beispiel feingranularer annotiert wurde als wir im Goldstandard festlegten, und die kleinteiligeren Annotationen von unserer gröberen Annotation eingeschlossen werden. Entscheidungen dieser Art werden in zukünftigen Phasen des Projektes fallen.

3.3.4 Alternative Methode: Werbefilterlisten

Eine Alternative zum Goldstandard wäre das automatische Überprüfen, ob Werbeblöcke als Segment erkannt wurden. Filterlisten von Werbeblockern sind frei zugänglich, wie etwa die EasyList EasyList [2016] und Parser für solche, wie den abp-filter-parser Bondy [2016]. Mit diesen Mitteln ließen sich auf Webseiten, die Werbung einblenden, die Komponenten identifizieren, die von Werbenetzwerken geladen werden. Über deren Boundingbox wüsste man über die Position und Größe des Werbeblocks Bescheid und könnte feststellen ob dieser vom Arbeiter eingerahmt wurde. Was uns gegen weitere Verfolgung dieses Ansatzes bewog war die Tatsache, dass Probleme bei der Wiedergabe der archivierten Webseiten vor allem bei Werbeinhalten auftraten (siehe 4.1.2). Die Arbeiter könnten die Werbeinhalte also visuell nicht wahrnehmen und somit auch nicht markieren.

Kapitel 4

Korpuskonstruktion

Der Korpus wird mit unserem Ansatz in drei Schritten erstellt: Erhebung der Rohdaten, manuelle Segmentierung und automatische Zusammenfassung der Ergebnisse. Bis jetzt fanden viele Tests mit den Interfaces sowie die Erstellung des Goldstandards statt. Die massenhafte Erhebung von Segmentierungen durch AMT-Arbeiter wird in einer zukünftigen Phase des Projektes erfolgen.
Bei der Rohdatenerhebung haben wir eine Methode gesucht, Webseiten möglichst vollständig, also inklusive aller HTML, JavaScript, CSS und anderen Quell- und Mediendaten zu archivieren, um sie reproduzierbar offline rendern zu können.
Die Screenshots dieser archivierten Webseiten werden anschließend in die HITs bei AMT eingebunden und von den Arbeitern durch unser Arbeiter-Interface segmentiert. Das wichtigste Resultat dieses Prozesses werden die Segmentierungen pro Screenshot-Abschnitt, also Rechteckdaten sein.
Wir lassen jeden Screenshot zehnfach auf AMT bearbeiten. Danach müssen die Ergebnisse pro Screenshot-Abschnitt zusammengefasst werden. Um abschließend die vollständige Segmentierung einer Webseite zu erhalten, werden die überlappenden Abschnitte zusammengefasst und Segmentierungen für die Screenshot-Abschnitte zusammengeführt.

4.1 Rohdatenerhebung und Vorverarbeitung

Vor dem Annotationsprozess wurden 1000 Webseiten archiviert und Screenshots von ihnen produziert. Als Input diente eine Liste von gesampelten URLs aus dem Common Crawl (siehe Elbaz [2011]), speziell dem Stand von November 2015. Der Output bestand aus einer strukturierten Sammlung von archivierten Webseiten und Informationen über den Archivierungsprozess. Die wichtigsten Komponenten des Outputs waren hierbei die Screenshots und die Quelldateien der Seiten im WARC-Format.

Visuelle Ansätze zur Segmentierung sind ohne Rendern der Seite nicht möglich, da die Gestaltung der Seite durch CSS-Regeln nicht beachtet werden kann. Elemente, die im DOM nahe beieinander liegen, können durch CSS-Regeln an völlig unterschiedlichen Positionen platziert, anderweitig verformt oder ausgeblendet werden. Durch die Verwendung von Headless-Browsern ist man bei der Segment-Findung nicht ausschließlich auf den DOM oder nur auf die Quelldateien einer Webseite angewiesen. Die Rohdaten wurden derart gesammelt, dass jede einzelne Webseite auch offline isoliert reproduzierbar gerendert und untersucht werden kann.
Die zwei dazu verwendeten Komponenten sind ein Headless-Browser, der die Seite rendert und Zugriff auf deren DOM bietet und ein Proxy-Werkzeug, welches alle Anfragen speichert und archiviert. In Abbildung 4.1 mittig oben werden diese zwei Komponenten im Kontext des Archivierungsprozesses dargestellt.
Headless-Browser können meist über die Kommandozeile angesteuert werden und Aufgaben mit der gerenderten Version einer Seite automatisiert erledigen. Sie werden häufig für End-to-End Tests in der Webentwicklung verwendet. Für die Webseiten-Segmentierung profitiert man durch Zugriff auf den DOM erheblich gegenüber Werkzeugen, die nur die einzelnen Quelldateien einer Seite untersuchen. Letztgenannte können solchen Inhalt nicht interpretieren, der per JavaScript erzeugt wurde. Dieser ist somit später nicht reproduzierbar. Während der Nachteil früher möglicherweise nur einzelne Bausteine der Webseite betraf, kann es sich heute um die ganze Webseite handeln. Die Verwendung von Client-seitig rendernden Frameworks wie AngularJS (Google [2010]) ermöglicht es, das Rendern der Webseite komplett auf den Client auszulagern. Im Falle von AngularJS wird ein, mit Variablen und Direktiven versehenes, HTML-Skelett per JavaScript mit Inhalt gefüllt. Dies steht im Gegensatz zu klassischen Praktiken, bei denen eventuell einzelne Teile einer serverseitig gerenderten Webseite per JavaScript mit Funktionalität versehen wurden.
Google reagiert auf diesen Trend durch Anpassung der Crawl Methode in der Hinsicht, dass diese nun "[...] generell in der Lage ist [...] Webseiten wie moderne Browser zu rendern und zu verstehen." Nagayama [2015]. Wie im Google Webmaster Central Blog beschrieben verabschiedet sich Google damit von der Empfehlung aus dem Jahr 2009 an Webmaster immer einen gerenderten HTML-Snapshot für jeden Zustand der Webseite vorzuhalten.
Der Headless-Browser PhantomJS (Hidayat [2010]) im Speziellen, wird über JavaScript-Dateien gesteuert und bietet eine eigene API an. So kann per Script das Scrollverhalten gesteuert, der „User-Agent" gesetzt, die Auflösung festgelegt und Screenshots erstellt werden. Die Funktionsweise und Verwendung beschreiben wir in 5.1.
Für die zukunftsfähige Archivierung von Webseiten reicht es nicht, einmalig

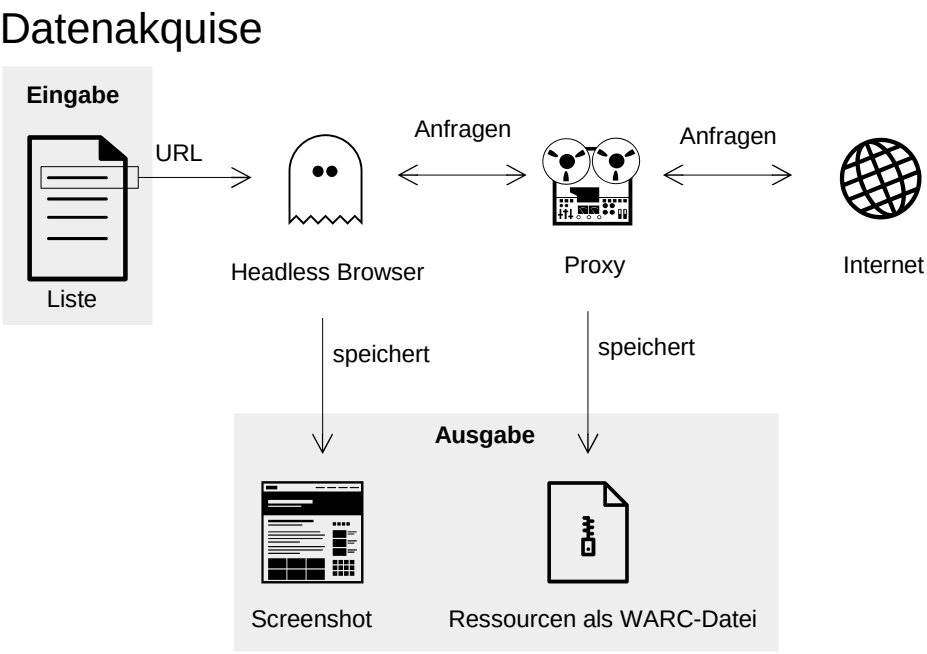

Abbildung 4.1: Zusammenspiel der einzelnen Komponenten, die zur Archivierung der Webseiten benötigt werden.

programmatisch Zugriff auf den DOM und das Erscheinungsbild zu haben. Es muss die Möglichkeit bestehen, diesen Zustand auch reproduzieren zu können; unabhängig davon ob die Originalressourcen (HTML,JS,CSS und Mediendaten) online verfügbar sind. Das Werkzeug warcprox (Proxy) fängt Anfragen von PhantomJS (Headless-Browser) wie in der Abbildung 4.1 ab und archiviert sie als WARC-Dateien. Die Funktionsweise wird in 5.1.2 beschrieben.

Würde man im Gegensatz zu den oben genannten Werkzeugen, Programme verwenden, die lediglich einzelne Ressourcen archivieren, oder den Einstiegspunkt einer Webseite (wie z.B. index.html) parsen und dort erwähnte Ressourcen nachladen, ergäben sich zwei Probleme. Zum einen bestünde die Unsicherheit alle Ressourcen archiviert zu haben, etwa wenn Links zu diesen dynamisch durch Javascript generiert werden. Zum anderen hätte man ein Problem bei der reproduzierbaren Wiedergabe, wenn Pfade absolut angegeben sind und noch auf Ressourcen im Internet verweisen.

WARC-Dateien können nicht im Browser geöffnet werden. Deswegen wird

ein Werkzeug zur Wiedergabe dieses Formats benötigt. Es wurde von uns PythonWayback (PyWB) verwendet. Es fungiert als lokaler Webserver und bietet dem Browser Zugang zu den Inhalten der WARC-Dateien. Es betreibt "Rewriting", das heißt, es schreibt alle Pfade in den ausgelieferten Ressourcen so um, dass sie auf den lokalen Server verweisen. Um die Screenshots der archivierten Seiten zu erstellen wurde mit PhantomJS auf die von PyWB bereitgestellten Seiten zugegriffen (siehe 5.1.3).

Die Darstellung eine Webseite wird maßgeblich durch Breite des Browserfensters beeinflusst. Responsive Designs passen sich der Browserbreite an und haben häufig Breakpoints, also Schwellwerte für die Breite, an denen sie zu einem anderen Seitenaufbau wechseln. Bei nicht-responsiven Designs passt der Browser die Seite an und bricht beispielsweise Textzeilen um. Wir suchten für die Screenshots eine Auflösung aus, in denen die meisten Webseiten korrekt dargestellt werden. Wir haben uns für 1366px * 768px pro Screenshot-Abschnitt entschieden, da diese Auflösung laut Awio Web Services [2016] im Bearbeitungszeitraum mit einem Anteil von 17,05% als die am weitesten Verbreitete im Internet galt. Die Höhe einer Webseite ändert sich meist mit Änderung der Browserfensterbreite automatisch.

Aufgrund der recht großen Höhe in Pixeln, die zahlreiche Webseiten aufweisen, haben wir uns dazu entschieden die Archiv-Screenshots mit Überlappung vertikal aufzuteilen, so dass jeder AMT-Arbeiter einen möglichst vergleichbaren Arbeitsaufwand pro HIT hat.

4.1.1 Auswahl der Webseiten: Top Million des Webs vs. Zufall

Die Auswahl an archivierten Webseiten sollte möglichst relevante Einträge enthalten. Um dies zu gewährleisten und gleichzeitig einen etablierten Datensatz zu nutzen, wurde das Alexa Ranking und der Common Crawl Elbaz [2011] mit dem Stand von November 2015, folgendermaßen untersucht: Wir extrahierten aus der Gesamtmenge von URLs im Common Crawl die URLs, die auch in der Top Million des Alexa Rankings vorkommen. Das Alexa Ranking ist ein globales Traffic Ranking von Webseiten, aus dem abzulesen ist, wie eine Webseite im Vergleich zu allen anderen Webseiten im Netz abschneidet. Das Ranking wird über eine proprietäre Methode mit historischen Daten der letzten drei Monate erhoben, basierend auf den Daten der Nutzer von 25000 verschiedenen Browser-Erweiterungen. Ebenfalls werden Daten von Webseiten einbezogen, die ein bereitgestelltes Skript auf ihre Seite eingebunden haben (vergleiche Alexa Internet, Inc [1996]). Dabei ergab sich die Feststellung, dass

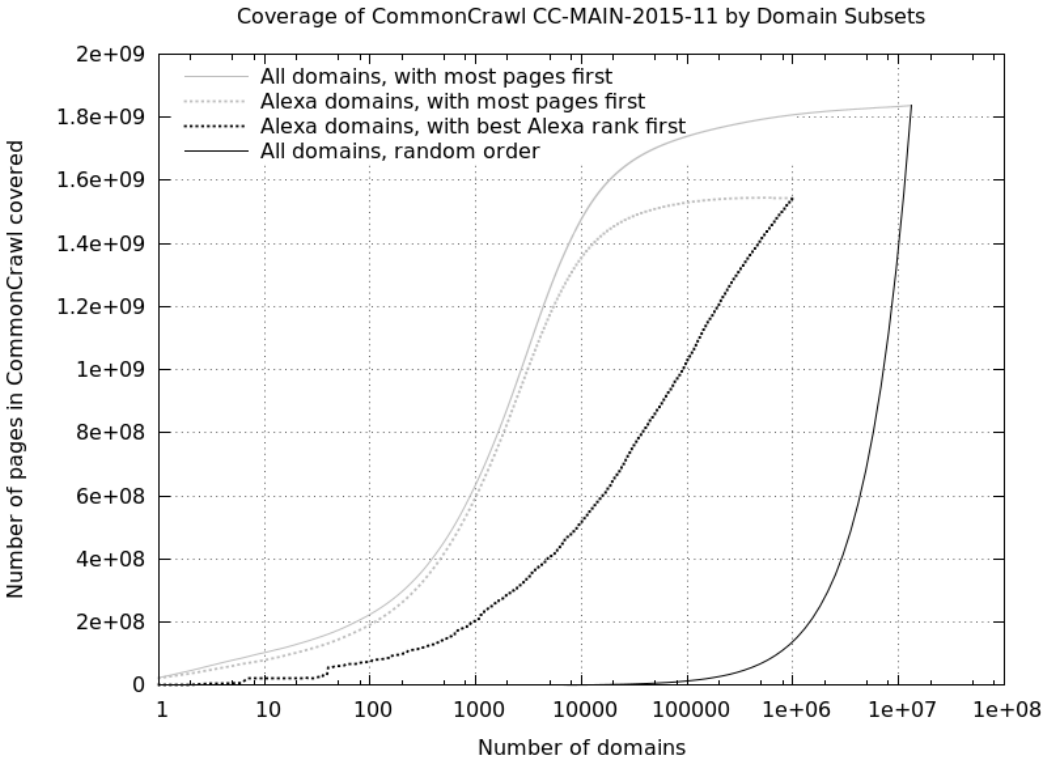

Abbildung 4.2: Vorkommen der Top Million Webseiten-URLs des Alexa Rankings im Index des Common Crawl von November 2015. Grafik erstellt von Johannes Kiesel, mit freundlicher Genehmigung verwendet.

der Common Crawl größtenteils URLs aus diesem Ranking enthält. Von allen URLs des Common Crawl entstammen 84% dem Alexa Ranking. Man sieht dies in Abbildung 4.2, bei der die Anzahl, der im Common Crawl enthaltenen Webseiten auf der y-Achse aufgetragen ist. Das Maximum der Alexa-Seiten liegt dort sehr nahe bei dem Maximum der Common Crawl Seiten und es entsteht die Vermutung, dass das Alexa Ranking als Seed, also Ausgangsliste für das Crawlen des Common Crawls verwendet wurde.

4.1.2 Struktur des Korpus

Für jede einzelne URL der Input-Liste werden folgende Dateien und Informationen generiert:

- Live Screenshot der Webseite.

- Rekonstruierbarer Screenshot der archivierten Seite.
- Die Differenz (falls vorhanden) der Screenshots farbig markiert.
- Alle Ressourcen der Webseite als WARC-Datei.
- Die Ratio der abweichenden Pixel beider Screenshots.

Die detaillierte Struktur mit Daten, die der Vereinfachung des Prozessablaufs dienten befindet sich im Anhang A.1.

PyWB befindet sich noch in der Entwicklungsphase und so konnten wir nicht alle Webseiten einwandfrei visuell reproduzierbar archivieren. Um eine durchschnittlich gute Qualität der Archiv-Screenshots sicherzustellen, haben wir sie automatisch mit Live-Screenshots verglichen. Mit Live-Screenshots meinen wir Screenshots die entstanden sind, indem wir per PhantomJS, während des Archivierens, auf die originale und online verfügbare Version der Webseite zugriffen. Wir verglichen basierend auf der visuellen Abweichung mit Hilfe von Resemble.js Cryer [2013] (siehe 5.1.4). Das separate Erstellen von Live Screenshot und Archiv Screenshot, sowie die Differenz beider ist nur nötig, da die Wiedergabe der WARC-Dateien nicht immer einwandfrei funktionierte. Wir stellten fest, dass sich vor allem Werbeblöcke, die sich in verschachtelten iFrames befanden nicht reproduzieren ließen und als Lücken in der gerenderten Archivansicht auftraten. Bei einer perfekten Wiedergabemöglichkeit würde der Live Screenshot ausreichen. Eingeschlossen in die Analyse wurden nur die Webseiten, bei denen beide Screenshots maximal eine Abweichung von 20% aufwiesen. Dieser Wert wurde von uns subjektiv, nach dem Betrachten einer Menge von 1000 archivierten Webseiten, die nach visueller Abweichung sortiert waren, festgelegt.

Segmentierungsvorschläge Die AMT-Arbeiter zeichnen durch das Worker-Interface (siehe Abschnitt 3.1) Rechtecke auf den archivierten Webseiten und produzieren somit Segmentierungsvorschläge. Es werden die Archiv Screenshots und nicht die Live Screenshots annotiert, da die erstgenannten reproduzierbar sind und somit später eine eindeutige Rückführung auf den konkreten Inhalt der einzelnen Segmente ermöglichen. Jeder Screenshot-Abschnitt wird von zehn Arbeitern annotiert und so werden je zehn Segmentierungsvorschläge produziert. Sie bestehen aus Rechtecks-Daten, die relativ zum Screenshot positioniert sind. Durch diese Position und die Verschiebung des Screenshot-Abschnitts kann später die genaue Position auf der ursprünglichen Webseite errechnet werden.

Logs und Metadaten der Archivierung Bei der Archivierung haben wir Log-Dateien der einzelnen Werkzeuge warcprox, PyWB und Resmble.js gespeichert. Für jeden Schritt wurde die benötigte Zeit gemessen. Darüber hinaus liegt jedem Archiveintrag der originale URL und den empfangenen HTTP-Antwort-Statuscode der Anfrage durch PhantomJS bei.

Segment Repräsentation im DOM Jede Segmentierng wird durch eine Textdatei vertreten, die einen eindeutigen xPath zu den einzelnen Fragmenten in der DOM-Struktur der gerenderten Seite enthält.

4.2 Annotationsprozess

Das Hauptprodukt der AMT-Arbeiter sind die Segmentierungen, die durch das Arbeiter-Interface erstellt werden. Diese Daten sind Rechtecke, die durch Position und Größe beschrieben sind. Über die vertikale Position des Screenshot-Abschnittes lässt sich die genaue Position des Segmentes auf der gerenderten Webseite feststellen.

Zudem wird auch der zeitliche Verlauf des Segmentierungsprozesses für jeden Screenshot-Abschnitt aufgezeichnet. Das bedeutet, dass jede Annotation nachträglich in Echtzeit beziehungsweise in variabler Geschwindigkeit betrachtet werden kann. In unserem Ansatz wurden diese Daten nicht in die automatische Analyse mit einbezogen, so könnten sie zukünftig jedoch verwendet werden, um den Korpus zu verbessern. Man könnte versuchen durch die Analyse des Entstehungsprozesses die Qualität einer Segmentierung festzustellen.

4.2.1 Segmentierung pro Webseite

Die Erstellung von Segmentierungen pro Webseite aus den Annotationen der AMT-Arbeiter erfolgt in drei Phasen. Erstens gibt es zehn verschiedene Segmentierungsvorschläge pro Screenshot-Abschnitt (da manuell von verschiedenen Personen erstellt) die vereinheitlicht werden müssen. Zweitens müssen die Screenshot-Abschnitte einer Webseite wieder zusammengesetzt werden und drittens sollte eine Rückführung von den vereinheitlichten Segmentierungsvorschlägen zu den konkreten Inhaltsfragmenten der Seite geschehen. Es handelt sich um folgende Fälle:

1. Gleicher Screenshot-Abschnitt, unterschiedliche Arbeiter.
2. Gleiche Webseite, unterschiedlicher Screenshot-Abschnitt.

3. Rückführung von Segmentierung auf DOM-Fragmente.

Diese werden in Abbildung 4.3 schematisch dargestellt.

Über die Boundingbox der Elemente der gerenderten Seite wird auf die DOM-Fragmente, die ein Segment beinhaltet geschlossen. Die Ausgabe der DOM-Fragmente pro Segment als xPath hat den Vorteil, dass Fragmente im DOM identifiziert werden können, auch wenn sich Segmente aus Inhalten von Geschwisterknoten zusammensetzen. So müssen Segmente nicht auf einzelnen DOM-Knoten beschränkt sein, wie bei dem Ansatz von Vineel [2009].

Bei der Frage, wie viele Arbeiter den gleichen HIT bearbeiten sollen, sind wir der empirisch festgestellten Empfehlung durch Carvalho et al. [2016] von zehn Arbeitern pro HIT gefolgt.

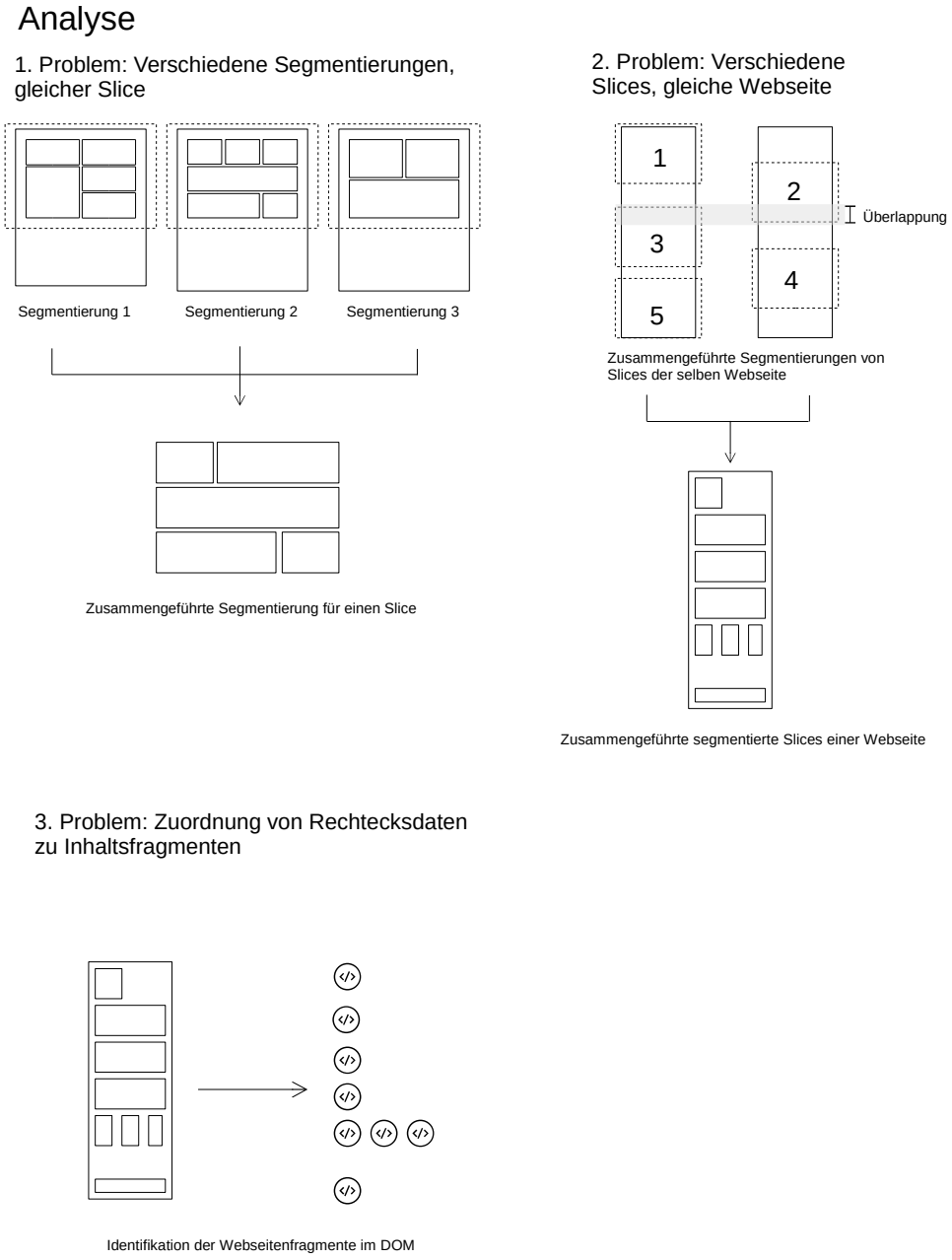

Abbildung 4.3: Drei Phasen zur Überführung der Annotationen zu Segmentierungen pro Webseite und Extrahierung der DOM-Fragmente.

Kapitel 5

Langzeitarchivierung und Rendering von Webseiten

Wie in Abschnitt 4.1 beschrieben verwendeten wir zur Archivierung eine Kombination aus Headless-Browser und Proxy. Dieser Ansatz steht im Kontrast zu klassischen Archivierungswerkzeugen wie dem Programm wget (Free Software Foundation [2015]), bei denen dynamisch nachgeladener Inhalt nicht beachtet wird. Im folgenden beschreiben wir die Werkzeuge, die wir für unsere Archivierungsmethode verwendeten.

5.1 Werkzeuge

Koordiniert wurden die Prozesse, die wir erstellten durch Grunt (Alman [2016]), einem Build-Werkzeug aus der Webentwicklung. Wir verwendeten es wegen seiner Erweiterbarkeit durch Plugins und dem erleichterten Umgang mit dem Dateien, zum Beispiel durch Globbing Patterns.

5.1.1 PhantomJS

Der Headless-Browser PhantomJS basiert auf WebKit und lässt sich über eine JavaScript API steuern. Gedacht ist das Werkzeug für Aufgaben des Testing, automatisierte Webseiten Benutzung, Screenshoterstellung und Netzwerk Monitoring (siehe Hidayat [2010]). Webseiten lassen sich aufrufen und man erhält programmatischen Zugriff auf den DOM, ebenso kann man Screenshots in frei wählbaren Auflösungen erstellen. Wir interagierten mit PhantomJS über den Wrapper CasperJS (Perriault [2011]).

5.1.2 warcprox

warcprox ist ein man-in-the-middle http(s)-Proxy. Man startet das Kommandozeilenprogramm mit gewünschtem Port und IP und trägt diese Informationen in den Browser seiner Wahl ein, in unserem Fall das Skript für PhantomJS. Jeder Anfrage und Antwort dieses Browsers wird nun in eine WARC-Datei geschrieben. WARC-Dateien sind Archive, die die Request-Header von Anfragen/Antworten als Klartext sowie deren mitgelieferten Daten (HTML, CSS, JavaSCript, Mediendaten) in Binärform enthalten. Der Webservice webrecorder.io RHIZOME [2015] verwendet warcprox als Bestandteil und bietet eine einfache Möglichkeit WARC-Dateien von Webseiten zu produzieren.

5.1.3 PythonWayback

PythonWayback, PyWB Kreymer [2013] bietet vereinfacht gesagt ein Internetarchiv wie WayBackMachine Archive [2016] für den eigenen PC. Als Input erhält es WARC-Dateien und stellt deren Inhalt als lokaler Server bereit. Dieser Server lässt sich über den Browser ansteuern und präsentiert eine Suchmaske. Hier kann nach archivierten URLs gesucht und die dazugehörigen Webseiten angezeigt werden.

Alle Links im ausgelieferten HTML werden als relative Pfade umgeschrieben, der Autor nennt es "Rewriting". Links die dynamisch in Javascript-Code generiert werden, werden dabei teilweise nicht geändert und verweisen auf Quellen aus dem Internet und nicht aus dem Archiv. Aus Gründen der Reproduzierbarkeit haben wir alle nicht-lokalen Anfragen blockiert; Quelldateien deren URL nicht als lokaler Link umgeschrieben wurde, konnten in unserer Methode nicht angezeigt und folglich nicht annotiert werden.

5.1.4 Resemble.js

Resemble.js vergleicht zwei Input-Bilder pixelgenau und errechnet die relative Abweichung der Bilder zueinander und gibt diese als Prozentwert zurück. Es wird per NodeJS gesteuert. Optional kann die Bibliothek auch die Unterschiede in einer Kopie des ersten Bildes einfärben, wie in Abbildung 5.3 veranschaulicht. Dies macht eine schnelle menschliche Interpretation möglich. Wir haben den Wert subjektiv nach einer Betrachtung von 1000 Differenzbildern festgelegt. Ein solches Differenzbild befindet sich für jede archivierte Seite im Korpus.

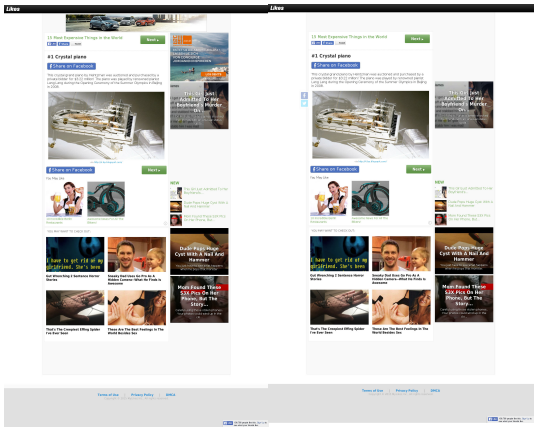

Abbildung 5.1: Online Screenshot.

Abbildung 5.2: Archiv Screenshot.

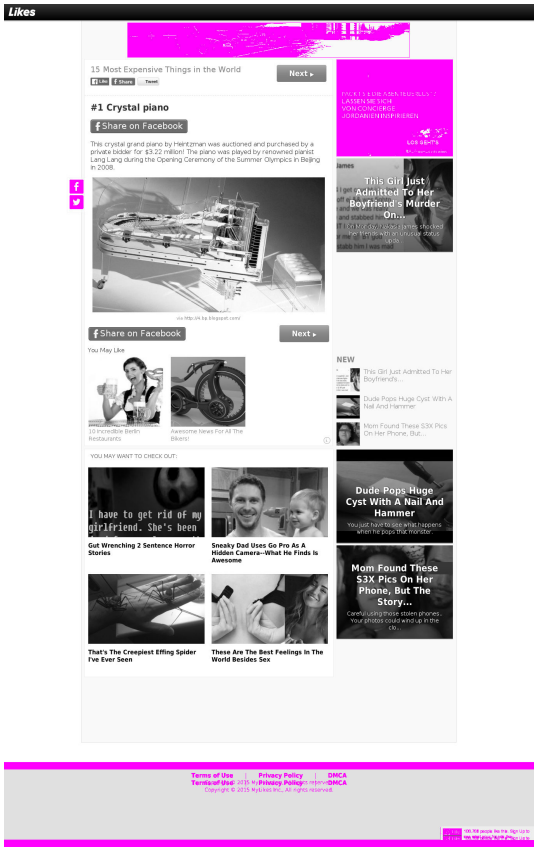

Abbildung 5.3: Screenshot Differenz durch Resemble.js erstellt. Bereiche in denen sich Online- und Archivscreenshot unterscheiden sind pink eingefärbt.

5.2 Limitierungen

Neben den Limitierungen bei der Wiedergabe von gewissen archivierten Inhalten durch PyWB (siehe Abschnitt 5.1.3) ist die größte Einschränkung unseres Ansatzes, die Verwendung von Screenshots anstatt einer gespiegelten Version der Webseite.

5.2.1 Limitierung durch Screenshots

In unserem Ansatz werden Webseiten durch die Abstraktionsebene Screenshot bearbeitet, wodurch sich Limitationen in Bezug auf die Annotation von gewissen Inhalten ergeben: animierter Inhalt und Inhalt der erst durch Nutzerinteraktion dargestellt wird, zum Beispiel durch das Betätigen eines Balkens in einer Scrollbox kann nicht Teil des Korpus werden; Abbildung 5.4 veranschaulicht die Problematik des verborgenen Inhalts. Animierter Inhalt kann nur durch ein Standbild dargestellt werden. Sind die Abgrenzungen aber erkennbar und werden richtig segmentiert, kann über die Boundingbox der Ressource dennoch unter Umständen auf die Originalressource geschlossen werden (wie Video, Animation und andere visuelle Inhalte). Es bestehen Ansätze auch animierten Inhalt visuell zu archivieren. Im Blog Mindthecode [2014] wird mit PhantomJS in einem Zeitintervall Screenshots einer Animation produziert und diese mit ffmpeg aneinandergefügt. Bridges [2015] erstellt Vorschaubilder von Flash-Inhalt. Um den Segmentierungsaufgabe nicht verwirrend zu gestalten haben wir auf das Aufnehmen von bewegtem Inhalt verzichtet.

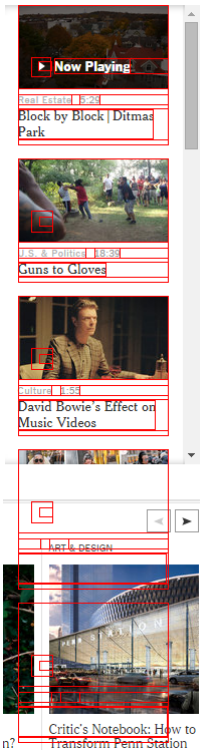

Abbildung 5.4: Beispiel einer Scrollbox und rote Markierung der teils verdeckten Content-Elemente

Kapitel 6

Zusammenfassung und Ausblick

Wir haben eine Pipeline geschaffen, an deren Anfang eine URL-Liste steht und an deren Ende zu jeder URL ein Segmentierungsvorschlag der zugehörigen Webseite existiert. Im Verlauf dieser Prozessfolge werden Webseiten archiviert, als Screenshots gespeichert. Diese Screenshot-Abschnitte werden von Menschen segmentiert, diese Segmentierungen von den Auftraggebern bewertet und die bewertete Segmentierung zusammengeführt. Der letzte Schritt wirft eine neues Arbeitsfeld auf, nämlich die Überführung der rechteckigen Segmentierungen zu den DOM-Elementen der Webseite.

Des Weiteren ergibt sich der Bedarf für ein genaues Regelwerk für den Segmentierungsvorgang, da schon unter den Autoren verschiedene Segmentierungen entstanden. Ein solches Regelwerk würde festlegen welche Inhalte ein Segment bilden und wäre von der Funktionsweise unserer Pipeline unabhängig.

Im weiteren Verlauf werden wir eine größere Anzahl von Webseiten segmentieren lassen um einen großen Goldstandard zu bilden, mit dem bestehende automatische Verfahren getestet und verglichen werden können.

Für zukünftige Forschung besteht die Option Computer Vision und Machine Learning in die Segmentierung von Webseiten mit einzubeziehen. Man könnte Verfahren aktualisieren, die auf veralteten Heuristiken beruhen, etwa dem Entwurf von Webseiten mit Hilfe von Tabellen. Ein ähnliches Verfahren wie das in dieser Arbeit beschriebene könnte auch leicht für die Kategorisierung von Inhalten implementiert werden. Weiterhin wäre es förderlich für Untersuchungen von Webseiten die Entwicklung von Werkzeugen im Bereich der Archivierung zu unterstützen, um möglichst akkurate Wiedergabe von Webseiten zu gewährleisten.

Anhang A

Anhang

A.1 Inhalt je Archiveintrag

- Archivierte Ressourcen der Webseite als WARC-Datei
- Live-Screenshot: Auf online Ressourcen per PhantomJS über warcprox zugegriffen
- Archiv-Screenshot: Archivierte Webseite per PyWB verfügbar gemacht und per PhantomJS zugegriffen
- Screenshot-Abschnitte des Archiv-Screenshots
- Screenshot-Differenz: Archiv-Screenshot und Live-Screenshot per Resemble.js verglichen
- Log des Werkzeugs PyWB
- Originale URL als Textdatei
- Response-Status der initialen Anfrage per PhantomJS
- Logs der Dauer aller drei Archivierungsschritte

Literaturverzeichnis

Alexa Internet, Inc. Alexa - actionable analytics for the web, 1996. URL `http://www.alexa.com/about`. 4.1.1

Ben Alman. Grunt: The javascript task runner, 2016. URL `http://gruntjs.com/`. 5.1

Apostolos Antonacopoulos, Basilios Gatos, and David Bridson. Page segmentation competition. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 1279–1283. IEEE, 2007. 3.3.2

Internet Archive. Internet archive: Wayback machine, 2016. URL `https://archive.org/web/`. 5.1.3

LLC Awio Web Services. W3counter: Global web stats - december 2015. `http://www.w3counter.com/globalstats.php?year=2015&month=12`, 2016. 4.1

Brian R. Bondy. Javascript adblock plus filter parser for lists like easylist, 2016. URL `https://github.com/bbondy/abp-filter-parser`. 3.3.4

Ryan Bridges. Putting the flash back in phantomjs. `http://www.ryanbridges.org/2013/05/21/putting-the-flash-back-in-phantomjs/`, 2015. 5.2.1

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In Xiaofang Zhou, Yanchun Zhang, and Maria E. Orlowska, editors, *APWeb*, volume 2642 of *Lecture Notes in Computer Science*, pages 406–417. Springer, 2003. ISBN 3-540-02354-2. 2

Arthur Carvalho, Stanko Dimitrov, and Kate Larson. How many crowdsourced workers should a requester hire? *Annals of Mathematics and Artificial Intelligence*, pages 1–28, 2016. 4.2.1

James Cryer. Resemble.js, 2013. URL `https://github.com/Huddle/Resemble.js`. 4.1.2

EasyList. Offizielle easylist-website, 2016. URL `https://easylist.adblockplus.org/de/`. 3.3.4

Gil Elbaz. Common crawl, 2011. URL `http://commoncrawl.org/`. 4.1, 4.1.1

Inc. Free Software Foundation. Gnu wget, 2015. URL `https://www.gnu.org/software/wget/`. 5

Google. Angularjs — superheroic javascript mvw framework, 2010. 4.1

Google. Editing styles and the dom - google chrome, 2016. URL `https://developer.chrome.com/devtools/docs/dom-and-styles#inspecting-elements`. 3.1.1

Thomas Gottron. *Content extraction-identifying the main content in HTML documents.* PhD thesis, Universität Mainz, 2008. 2

Ariya Hidayat. Phantomjs. `http://phantomjs.org/`, 2010. 4.1, 5.1.1

Christian Kohlschütter and Wolfgang Nejdl. A densitometric approach to web page segmentation. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1173–1182. ACM, 2008. 2

Tomaž Kovačič. Evaluating web content extraction algorithms. *Bachelor's Thesis, Faculty of Computer and Information Science, University of Ljubljana, Slovenia*, 2012. 2

Ilya Kreymer. Pywb. `https://github.com/ikreymer/pywb`, 2013. 5.1.3

Mathworks. Create draggable rectangle - matlab imrect - mathworks deutschland, 2016. URL `http://de.mathworks.com/help/images/ref/imrect.html`. 3.1.1

Mindthecode. Recording a website with phantomjs and ffmpeg. `http://mindthecode.com/recording-a-website-with-phantomjs-and-ffmpeg`, 2014. 5.2.1

Kazushi Nagayama. Deprecating our ajax crawling scheme, 2015. URL `https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html`. 4.1

Nicolas Perriault. Casperjs. `http://casperjs.org/`, 2011. 5.1.1

Ihsin T Phillips, Jisheng Liang, Atul K Chhabra, and Robert Haralick. A performance evaluation protocol for graphics recognition systems. In *Graphics Recognition Algorithms and Systems*, pages 372–389. Springer, 1997. 3.3.2

RHIZOME. webrecorder.io, 2015. URL `https://webrecorder.io/`. 5.1.2

Jack Spirou. Device information and digital fingerprinting written in pure javascript., 2016. URL `https://clientjs.org/`. 3.1.1

Gujjar Vineel. Web page dom node characterization and its application to page segmentation. In *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, pages 1–6. IEEE, 2009. 2, 3.1.1, 4.2.1

Max Wertheimer. Untersuchungen zur lehre von der gestalt. ii. *Psychologische Forschung*, 4(1):301–350, 1923. ISSN 1430-2772. doi: 10.1007/BF00410640. URL `http://dx.doi.org/10.1007/BF00410640`. 3.1.1