Die Nutzung von Webdaten in den Sozialwissenschaften

Simon Munzert und Dominic Nyhuis

Inhalt

1 Einführung in die Webdatensammlung 374
2 Was ist und warum nutzen wir Web Scraping? 375
3 Grundlagen des Web Scraping 380
4 Potenziale und Herausforderungen webbasierter Sozialwissenschaft: Ausgewählte Anwendungen 390
5 Gegenwärtige Problemfelder 392
6 Kommentiertes Literaturverzeichnis 394
Literatur 396

Zusammenfassung

Das Kapitel bietet einen Überblick der Webdatensammlung für die sozialwissenschaftliche Forschung. Zu diesem Zweck wird nach einem praktischen Beispiel eine Übersicht der grundlegenden Webtechnologien geboten, um in einem zweiten Schritt einen vertiefenden Blick auf das Web Scraping einerseits und Programmierschnittstellen andererseits zu werfen. Die praktische Umsetzung der Webdatensammlung wird mit Code-Beispielen in der Programmiersprache R illustriert. Nach der praktischen Einführung werden Potenziale und Herausforderungen der webbasierten Sozialwissenschaft am Beispiel ausgewählter Anwendungen aus der aktuellen Forschungsliteratur diskutiert. Abschließend werden verschiedene technische und konzeptionelle Problemstellungen der Webdatensammlung dargelegt und einige weiterführende Literaturhinweise für die vertiefte Auseinandersetzung mit den Themen des Kapitels geboten.

S. Munzert (✉)
Hertie School, Berlin, Deutschland
E-Mail: munzert@hertie-school.org

D. Nyhuis (✉)
Department of Political Science and the Center for European Studies, University of North Carolina, Chapel Hill, USA
E-Mail: nyhuis@unc.edu

© Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2020

C. Wagemann et al. (Hrsg.), *Handbuch Methoden der Politikwissenschaft*,
https://doi.org/10.1007/978-3-658-16936-7_22

Schlüsselwörter

Webdatensammlung · Web Scraping · Programmierschnittstellen · APIs · R

1 Einführung in die Webdatensammlung

Das rasante Wachstum des Internets in den vergangenen zwei Jahrzehnten hatte einen enormen Einfluss darauf, wie wir Daten sammeln, teilen und veröffentlichen. Unternehmen, öffentliche Institutionen und private Nutzer stellen jede erdenkliche Art von Information zur Verfügung. Soziale Medien, Smartphones, Online-Shopping-Plattformen und viele andere Dienste und Kommunikationswerkzeuge erzeugen riesige Datenmengen über menschliches Verhalten. Viele dieser Daten sind Produkte sozialer Interaktion und somit von unmittelbarem Interesse für die Sozialwissenschaften. Was einst ein Grundproblem der Disziplin war – ein Mangel an empirischen Beobachtungen –, wurde an vielen Stellen durch einen Überfluss an Daten ersetzt.

Diese Entwicklung bringt allerdings Probleme ganz eigener Art mit sich: Wie können Millionen von Textbotschaften analysiert, wie die Struktur riesiger Netzwerke erfasst, wie die „Signale" – substanziell bedeutsame Informationen – aus dem Datenrauschen herausgelesen werden? Für einige dieser Herausforderungen werden in diesem Band Lösungsansätze vorgestellt (siehe hierzu u. a. die Beiträge von Hutter, Leifeld, Proksch oder Weidmann und Gleditsch). Dieses Kapitel wendet sich der ersten Hürde in der Arbeit mit Webdaten zu: der Sammlung und Aufbereitung von Daten aus dem Netz.

Zum Einstieg und als Motivation stellen wir zunächst die Vorteile des Web Scraping im Rahmen einer Beispielanwendung dar. Hier, wie auch im weiteren Verlauf des Kapitels, nutzen wir die Programmiersprache R zur Darstellung der technischen Aspekte des Web Scraping. Gerade für sozialwissenschaftliche Anwender halten wir R für das Programm der Wahl, da es sich ebenso für die weiteren Schritte des Forschungsprozesses eignet, insbesondere für die Analyse und Visualisierung der gewonnenen Daten. Im Idealfall kann somit der gesamte Forschungsprozess – von der Datensammlung bis zur Veröffentlichung – innerhalb eines einheitlichen technischen Rahmens vollzogen werden. Zweitens mag es bei einer Leserschaft mit sozialwissenschaftlichem Hintergrund bereits gewisse Vorkenntnisse mit R geben. Durch seine Dominanz im Bereich neuerer statistischer Techniken hat R eine enorme Sogwirkung in den Sozialwissenschaften und darüber hinaus entfaltet. Somit können Anwender bei der Nutzung von R für die automatische Datensammlung Fähigkeiten erlernen, die im Forschungsalltag gewinnbringend eingesetzt werden können. Schließlich ist die Funktionalität von R im Bereich der webbasierten Datensammlung durch die Veröffentlichung von Zusatzpaketen mittlerweile so weit fortgeschritten, dass R seinen Wettbewerbern kaum noch nachsteht und auch komplexe Scraping-Probleme mit R bearbeitet werden können.

Im zweiten Abschnitt beschäftigen wir uns mit den Grundlagen des Web Scraping. Dabei bieten wir sowohl einen Überblick über die zentralen technischen

Bausteine (Webtechnologien), mit denen viele praktische Anwendungen selbstständig durchgeführt werden können. Im Detail beschäftigen wir uns mit den Grundlagen von HTML und dem Auslesen einfacher Webseiten mit Hilfe von R. Der Abschnitt schließt mit einer Diskussion von Programmierschnittstellen (APIs) und ihrer Bedeutung für die webbasierte Datensammlung. Im Anschluss präsentieren wir eine Reihe von Anwendungen aus der jüngeren politikwissenschaftlichen Forschung und identifizieren verschiedene Problemfelder für die automatische Datensammlung. Das Kapitel schließt mit Verweisen auf einige weiterführende Werke.

2 Was ist und warum nutzen wir Web Scraping?

Große Mengen unstrukturierter Daten lassen sich nur mit erheblichem Zeitaufwand und unter Inkaufnahme von Kodierungs- und Übertragungsfehlern manuell sammeln. Gerade in der Auseinandersetzung mit großen Datenbeständen ist es deshalb häufig lohnenswert, die Datensammlung zu automatisieren. Darum geht es im Kern, wenn von *Web Scraping*, *Web Harvesting* oder *Web Crawling* die Rede ist. Viele Ressourcen auf einem Server werden mittels eines Computerprogramms ausgelesen und die unstrukturierten Informationen in eine strukturierte Form gebracht – beispielsweise in eine Datenbank.[1]

Nicht immer mag der Vorteil des skript-basierten Web Scrapings, also der Automatisierung der Datensammlung, offensichtlich sein. Unter welchen Bedingungen also sollte man sich mit den in diesem Kapitel beschriebenen Techniken auseinandersetzen? Folgende Checkliste mag bei der Entscheidung helfen:

- Planen Sie, die Datenerhebung periodisch zu wiederholen, zum Beispiel, um Ihre Datenbank zu aktualisieren?
- Möchten Sie, dass Dritte Ihren Datenerhebungsprozess replizieren können?
- Arbeiten Sie häufig mit Online-Datenquellen?
- Ist die Aufgabe in Bezug auf Umfang und Komplexität nicht trivial, d. h. kann sie nicht durch einfache Copy-and-Paste-Operationen innerhalb kurzer Zeit gelöst werden?
- Falls die Aufgabe manuell erledigt werden kann – fehlen die Ressourcen, um die Arbeit erledigen zu lassen?

[1]Trotz ihrer engen Verwandtschaft ist das Interesse der Sozialwissenschaft häufig besser mit den Begriffen des Web Scraping oder des Web Harvesting beschrieben. Hier steht die Sammlung einer bestimmten Materialmenge für die weitere Analyse im Vordergrund, während beim Web Crawling, das auch unter dem Begriff des Web Spidering diskutiert wird, das Anliegen eher in der Indexierung von Webseiten und den Beziehungsnetzwerken zwischen Seiten besteht. So machen sich Web Crawler die internen und externen Verlinkungen auf einer Webseite zunutze, um das Beziehungsgeflecht zwischen verschiedenen Seiten zu erfassen. Diese Unterscheidung schließt freilich nicht aus, dass es durchaus sozialwissenschaftliche Anwendungen gibt, die sich Techniken des Web Crawling zunutze machen, etwa in der Analyse von Beziehungsnetzwerken in einem bestimmten Politikfeld (Ackland und O'Neil 2011; McNutt und Pal 2011).

- Sind Sie bereit, die Datenerhebung durch Programmierung zu automatisieren?
- Sind Sie interessiert an den technischen Grundlagen moderner, webdatenbasierter politikwissenschaftlicher Forschung?

Falls Sie zumindest einige dieser Punkte mit Ja beantworten konnten, sollten sich Techniken zur automatisierten Datensammlung als hilfreich erweisen. Bevor wir in die Details des Web Scraping mit einem Beispiel einstiegen, sei noch eine letzte Bemerkung vorangestellt: Das Potenzial des Internets für die sozialwissenschaftliche Forschung ist nicht auf die Sammlung von „Sekundärdaten" durch die zu beschreibenden Techniken beschränkt. Immer häufiger werden Online-Surveys genutzt, um schnell und kostengünstig Umfragen umzusetzen und Experimente zu realisieren. Crowdsourcing-Plattformen wie beispielsweise Amazon MTurk oder Crowdflower werden darüber hinaus seit Längerem verwendet, um etwa Klassifikationsaufgaben, die nicht gut automatisiert werden können, manuell ausführen zu lassen (Benoit et al. 2016). Auch wenn der Wert dieser Werkzeuge unbestritten ist, bleiben sie in diesem Kapitel unberücksichtigt.

Zur Illustration der konkreten Schritte bei der Sammlung von Webdaten sei eine kleine praktische Anwendung vorgestellt. Wir starten auf den Seiten der deutschen Wikipedia, die ein Paradebeispiel für die Fülle frei verfügbarer Daten im Netz darstellt. Unter https://de.wikipedia.org/wiki/Liste_der_Mitglieder_des_Deutschen_Bundestages_(18._Wahlperiode) findet sich eine Liste der Mitglieder des 18. Deutschen Bundestags (vgl. Abb. 1). Alle Abgeordneten sind zusätzlich zu ihrer Nennung auf diesem Index-Artikel mit jeweils einem eigenen Artikel auf der deutschen Wikipedia vertreten. Die Idee für die Anwendung ist nun folgende: Die Kohorte der Abgeordneten ist sowohl durch formelle (Parteimitgliedschaft, Mitgliedschaft in Ausschüssen, Wahlkreis, etc.) als auch durch informelle Kriterien (Bundesland, Arbeitsbereiche, etc.) miteinander vernetzt. Diese Vernetzung wird teilweise dadurch reflektiert, dass die individuellen Abgeordnetenseiten über Verlinkungen miteinander verknüpft sind. Durch Auslesen der individuellen Seiten können also die sichtbaren Netzwerkbeziehungen der Abgeordneten über diese Verlinkungen rekonstruiert werden, um beispielsweise besonders zentrale Akteure zu identifizieren oder Verbindungen zu identifizieren, die über offensichtliche Merkmale wie die Parteizugehörigkeit hinausgehen. Dazu werden alle Einträge der Abgeordneten heruntergeladen, die Verlinkungen zwischen den Einträgen identifiziert und das so gewonnene Netzwerk analysiert.

Vorbereitung

Für die praktische Anwendung verwenden wir die Software R und das Zusatzmodul `rvest`, welches Funktionen für das Scraping von Webseiten bereitstellt.[2]

[2]Den Code für diese Übung haben wir in folgendem GitHub-Archiv hinterlegt: https://github.com/simonmunzert/munzert-nyhuis-webdaten.

Abb. 1 Ausschnitt der Wikipedia-Seite https://de.wikipedia.org/wiki/Liste_der_Mitglieder_des_Deutschen_Bundestages_(18._Wahlperiode), Stand: 07. August 2017

Schritt 1: Artikelseiten der Abgeordneten identifizieren

Wir beginnen damit, die Übersichtsseite mit allen verlinkten Abgeordneten in R einzulesen. Dazu reicht der Befehl `read_html()` aus, mit dem wir den Quelltext der Seite im Objekt `html` speichern:

```
html <- read_html("https://de.wikipedia.org/wiki/
              Liste_der_Mitglieder_des_Deutschen_Bundestages_
(18._Wahlperiode)")
```

Jetzt werden alle Links im Quelltext identifiziert, die zu den Einträgen der Abgeordneten führen. Dazu verwenden wir einen sogenannten CSS-Selektor, der es uns erlaubt, nach spezifischen Elementen im Quelltext zu suchen.[3] Dabei machen wir uns zunutze, dass HTML-Seiten in der Regel eine klare Struktur aufweisen und alle interessierenden Elemente bestimmte Gemeinsamkeiten hinsichtlich ihrer Einbindung in die Struktur des Quelltextes aufweisen. Unsere Aufgabe besteht also darin, einen Selektor zu formulieren, der diese Gemeinsamkeiten beschreibt. Im vorliegenden Fall befinden sich alle Links in den Zellen der ersten Spalte der Tabelle. Dabei interessieren wir uns ausschließlich für die Adressen der Hyperlinks, die in einem `href`-Attribut in sogenannten Anker-Elementen (`<a>`) hinterlegt sind. Um

[3]Die Details der Schritte werden in Abschn. 3 erläutert.

alle Links aus dem Quelltext zusammenzustellen, können wir diese Überlegungen wie folgt in Code ausdrücken:

```
zellen <- html_nodes(html, css = ".wikitable td:nth-child(1)
a")
links <- html_attr(zellen, name = "href")
```

Nach kleineren Aufräumarbeiten, die wir hier aus Platzgründen nicht dokumentieren, im Skript aber nachvollzogen werden können (siehe Fußnote 2), landen wir bei einer Liste von 656 Links zu den Wikipedia-Einträgen der Abgeordneten.[4] Zur Illustration die ersten vier dieser Links:

```
> links <- links[5:length(links)] # Die ersten vier Links
sind irrelevant und werden ausgeschlossen
> links[1:4]
[1] "/wiki/Jan_van_Aken_(Politiker)" "/wiki/Stephan_Albani"
[3] "/wiki/Katrin_Albsteiger"        "/wiki/Agnes_Alpers"
```

Schritt 2: HTML-Dateien herunterladen
Im zweiten Schritt nutzen wir die gesammelten Link-Adressen, um die HTML-Dateien der entsprechenden Artikel herunterzuladen. Dazu greifen wir auf die `download.file()`-Funktion in R zurück und wenden sie auf alle 656 Links an. Wir springen erneut direkt zum Ergebnis und blicken in den eigens dafür angelegten Ordner auf unserer Festplatte:[5]

```
> list.files()[1:4]
[1] "Achim_Post.html"         "Agnes_Alpers.html"
[3] "Agnieszka_Brugger.html" "Albert_Rupprecht.html"
> length(list.files())
[1] 656
```

Innerhalb weniger Minuten wurden über 650 Dateien heruntergeladen und können nun weiterverarbeitet werden. Der Prozess des Web Scrapings im engeren Sinne ist damit bereits abgeschlossen.

[4]Die Abweichung von der Größe des 18. Deutschen Bundestags mit seinen 630 Mitgliedern ergibt sich durch Nachrücker für ausscheidende Abgeordnete.

[5]Die ersten vier Einträge unterscheiden sich von den ersten vier Einträgen zuvor, da wir im Zuge des Downloads neue Namen für die HTML-Seiten nach der Konvention Vorname_Nachname.html vergeben. Die heruntergeladenen Dokumente auf unserer Festplatte sind also anders sortiert als die Einträge auf der Index-Seite der Wikipedia.

Schritt 3: HTML-Dateien in R einlesen und Verlinkungen zwischen den Artikeln identifizieren
Nachdem wir alle benötigten Daten auf die Festplatte geladen haben, können wir uns nun der Datenverarbeitung widmen. Zuerst werden alle Dateien in einen „Container" in R importiert, eine sogenannte Liste (ebenfalls nicht dargestellt). Danach suchen wir in jedem der importierten Dokumente nach Verlinkungen zu weiteren Abgeordneten. Der Eintrag zu Katrin Göring-Eckardt (GRÜNE) beispielsweise enthält die folgende Passage:

Formatierter Text:
Seit Oktober 2013 ist sie neben Anton Hofreiter Vorsitzende der Bundestagsfraktion ihrer Partei,[2] ein Amt, das sie bereits von 2002 bis 2005 neben Krista Sager bekleidet hatte.

HTML-Code:
```
Seit Oktober 2013 ist sie neben <a href="/wiki/
Anton_Hofreiter" title="Anton Hofreiter">Anton Hofreiter</a>
Vorsitzende der <a href="/wiki/Bundestagsfraktion_B%C3%
BCndnis_90/Die_Gr%C3%BCnen" title="Bundestagsfraktion Bündnis
90/Die Grünen">Bundestagsfraktion ihrer Partei</a>,<sup
id="cite_ref-fraktionswahl_2-0" class="reference"><a
href="#cite_note-fraktionswahl-2">[2]</a></sup> ein Amt,
das sie bereits von 2002 bis 2005 neben <a href="/wiki/
Krista_Sager" title="Krista Sager">Krista Sager</a>
bekleidet hatte.
```

Links zu anderen Wikipedia-Einträgen können leicht identifiziert werden – sie beginnen mit `<a href="/wiki/`. Jetzt geht es lediglich darum, diese Links mit der Liste der Bundestagsabgeordneten abzugleichen. Dazu vergleichen wir die Liste, die wir im ersten Schritt von der Wikipedia gescrapt haben, mit allen Links auf den individuellen Abgeordnetenseiten. Im Beispiel ist Anton Hofreiter ebenfalls Mitglied des 18. Bundestages; wir speichern also eine Verknüpfung zwischen Göring-Eckardt und Hofreiter. Krista Sager war ebenfalls Abgeordnete, gehörte dem Bundestag nach der Wahl 2013 jedoch nicht mehr an. Entsprechend des Kriteriums, nur Verknüpfungen zwischen gegenwärtigen Mitgliedern zu nutzen, wird dieser Link also nicht gespeichert. Nach Auswertung aller Artikel nach diesem Schema erstellen wir eine Verknüpfungsmatrix, die alle Verlinkungen zwischen den Einträgen enthält (vgl. Tab. 1).

Schritt 4: Netzwerk visualisieren
In einem letzten Schritt sollen die aufgedeckten Verknüpfungen graphisch aufbereitet werden. Hierfür machen wir uns die Flexibilität der R-Software zunutze. Mittels des `igraph`-Pakets können wir den sogenannten *PageRank*-Score jeder Seite

Tab. 1 Ausschnitt aus der Verknüpfungsmatrix von Wikipedia-Abgeordnetenseiten

Von	Zu
Jan van Aken	Sevim Dağdelen
Jan van Aken	Inge Höger
Katrin Albsteiger	Barbara Lanzinger
Agnes Alpers	Birgit Menz
Peter Altmaier	Angela Merkel
Peter Altmaier	Ronald Pofalla
Peter Altmaier	Martina Renner
Peter Altmaier	Norbert Röttgen
...	...

berechnen, der, vereinfacht gesagt, ausdrückt, wie viele Artikel auf einen bestimmten Artikel verlinken. Besonders einflussreiche Politiker erzielen hier also einen höheren Wert.[6] Dies spiegelt sich auch in der – ebenfalls mit `igraph` generierten – Visualisierung des Netzwerks wider (vgl. Abb. 2): Den einflussreichsten Knotenpunkt bildet der Artikel von Angela Merkel; dahinter folgen Frank-Walter Steinmeier, Ursula von der Leyen und Peer Steinbrück.

3 Grundlagen des Web Scraping

3.1 Grundlegende Webtechnologien

Über eines sollte das Eingangsbeispiel nicht hinwegtäuschen: Das Handwerk des Datensammelns aus dem Web ist nicht immer ein derart geradliniger Prozess. Schwieriger wird es beispielsweise, wenn Webseiten dynamisch generiert werden oder die gewünschte Information in einem Textformat ohne Struktur- oder Meta-Information vorliegt. Das könnte beispielsweise dann der Fall sein, wenn wir uns für spezifische Informationen aus Online-Nachrichtenartikeln interessieren, die nicht im Quellcode einer Webseite hinterlegt sind. Auch für solche Problemstellungen gibt es freilich passende technische Lösungen. Deshalb gilt grundsätzlich: Automatisierte Datengewinnung aus dem Web ist kein Hexenwerk, erfordert aber mitunter grundlegende Kenntnisse über viele kleine Technologiebausteine.

Abb. 3 gibt einen Überblick über die wichtigsten dieser Bausteine. Die erste Säule versammelt Technologien, die bei der Veröffentlichung von Inhalten im Web zum Einsatz kommen, die zweite fasst Technologien zusammen, um diese Inhalte zu extrahieren und in nutzbare Formate (z. B. einen Datensatz) zu übertragen. *HTML* (Hypertext Markup Language) stellt dabei den wichtigsten Standard dar: Praktisch allen Webseiten im Netz liegt HTML-Code zugrunde. Da auch Tabellen, Listen,

[6] Der *PageRank*-Algorithmus wurde 1996 von den Google-Gründern Larry Page und Sergey Brin entwickelt, der später zur Grundlage der Suchmaschine wurde.

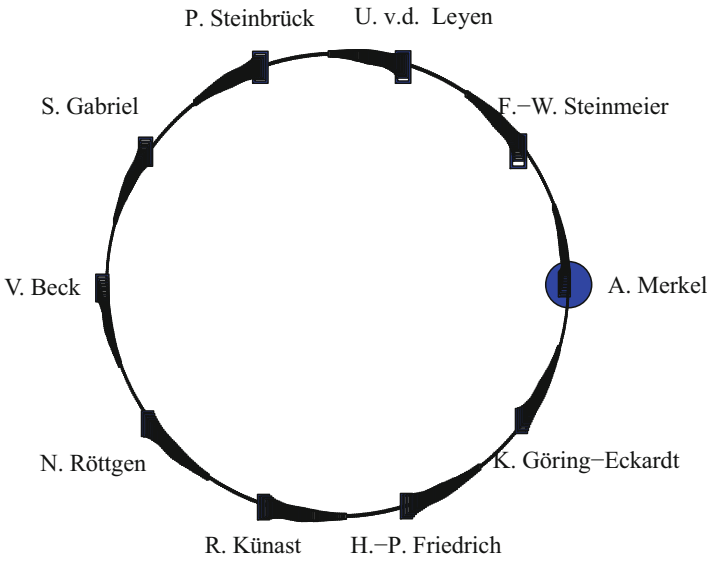

Abb. 2 Verknüpfungen zwischen den Wikipedia-Artikeln der Bundestagsabgeordneten

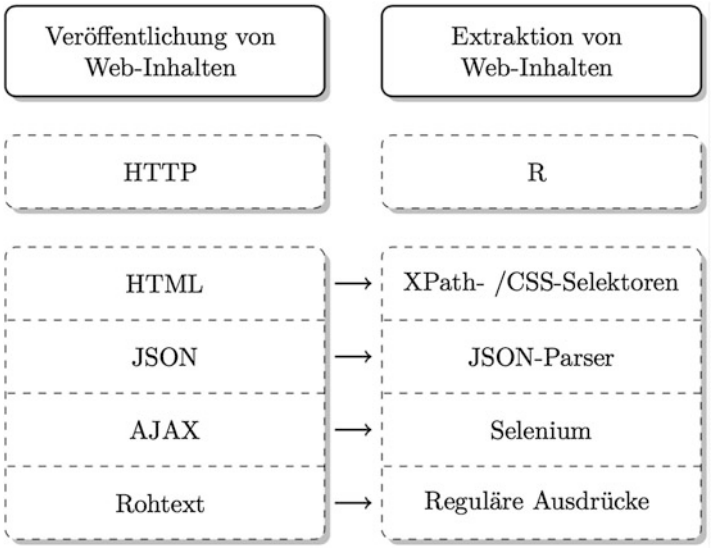

Abb. 3 Technologien zur Veröffentlichung und Extraktion von Webdaten

Überschriften und weitere Elemente von Webseiten in HTML-Code formatiert werden, ist es wichtig, die grundlegende Syntax von HTML zu verstehen, um diese Inhalte auslesen zu können. Dies ist mithilfe sogenannter *XPath-Ausdrücke* und *CSS-Selektoren* möglich, „Mini-Sprachen“, mit denen zu extrahierende Information

formalisiert beschrieben werden können (z. B. eine bestimmte Tabelle, Überschriften von Nachrichtenseiten, Absätze einer Pressemitteilung, etc.).

JSON (JavaScript Object Notation) ist zum de facto Standard unter den Datenaustauschformaten im Web geworden. Plattformen wie Twitter oder Facebook etwa bieten Entwicklern Zugang zu Teilen ihrer Daten, die für Forschungs- oder andere Zwecke genutzt werden können. JSON ist ein sehr flexibles Datenformat, das in allen gängigen Programmiersprachen verarbeitet werden kann. Dazu lesen sogenannte *JSON-Parser* JSON-Daten ein und wandeln sie in nutzbare Formate um – z. B. einen Datensatz.

AJAX (Asynchronous JavaScript and XML) steht für ein Set von Technologien, welches die dynamische Generierung von Webseiteninhalten ermöglicht. Webanwendungen wie Facebook, Google Maps oder viele moderne Nachrichtenwebseiten nutzen AJAX-Technologien, um ein dynamisches Nutzererlebnis im Browser zu ermöglichen. So werden etwa beim Scrollen auf einer Webseite neue Inhalte im Hintergrund nachgeladen. Denken Sie als Beispiel an die Google-Bildersuche. Wenn Sie die Bildersuche für einem bestimmten Suchbegriff verwenden, dann werden zunächst eine begrenzte Anzahl Bilder geladen in der Erwartung, dass sie nicht beliebig weit nach unten scrollen wollen und somit vermieden wird, dass unnötig viele Informationen vom Server geladen werden. Wenn Sie dann doch weiter scrollen, lädt das System automatisch weitere Bilder nach, um zu vermeiden, dass beim Nachladen Wartezeiten entstehen. Was das Betrachten solcher Inhalte im Browser komfortabler machen soll, ist aus Sicht des Webdatensammlers eher unerfreulich. Es ist technisch komplexer, Inhalte von solchen Seiten zu laden, aber durchaus möglich. Unser Mittel der Wahl hierfür ist das *Selenium*-Framework, mit dem man Interaktion im Web (z. B. das Ausfüllen von Formularen, Scrollen, Klicken, etc.) automatisieren kann.

Webinhalte von sozialwissenschaftlichem Interesse liegen meist in Textform vor – seien es Pressemitteilungen, politische Statements oder Meinungsäußerungen in Foren. Ein wichtiger Schritt bei der Datengewinnung ist deshalb häufig die Extraktion der relevanten Informationen aus reinen Textdaten. Hierbei sind reguläre Ausdrücke (*regular expressions*) hilfreich. Bei regulären Ausdrücken handelt es sich um eine formale Sprache, die, vergleichbar mit den im Eingangsbeispiel verwendeten CSS-Selektoren, genutzt werden kann, um regelmäßige Muster in Texten zu beschreiben. Auch hierfür sind spezielle Pakete in R implementiert worden. Beispielsweise könnten wir reguläre Ausdrücke verwenden, um alle genannten Webseiten aus einem Text zu extrahieren, indem wir Begriffe suchen, die mit „www“ beginnen.

Bislang haben wir verschwiegen, wie die genannten Technologien konkret zum Einsatz kommen. Um Inhalte von Servern abzurufen, wird meist das *HTTP* (Hypertext Transfer Protocol) verwendet. Beim „normalen“ Surfen wird die Kommunikation zwischen Server und Client (also dem Rechner des Benutzers) automatisch vom Browser abgewickelt. Automatisches Web Scraping versucht dagegen meist, den Browser für das Sammeln von Webinhalten zu umgehen. Glücklicherweise bieten viele Programmiersprachen Erweiterungen, die ebenfalls die Kommunikation zwischen Client und Server automatisieren, dem Nutzer aber, falls nötig, ermöglichen,

in die Interaktion zwischen Server und Client einzugreifen. Dies kann beispielsweise der Fall sein, wenn Zugangsdaten für Passwort-geschützte Seiten mitgeschickt werden müssen.

Wie bereits bemerkt, ist unsere Software der Wahl zum Sammeln von Webdaten die freie Programmiersprache R. Mit R können wir, wie im Eingangsbeispiel illustriert, Webseiten abfragen, Inhalte herunterladen, spezifische Teile einer Webseite mittels XPath-Ausdrücken oder CSS-Selektoren auslesen, JSON in R-Objekte überführen, manuelle Seitennutzung mittels Selenium simulieren und vieles mehr. Es dient uns somit als eine Art ‚Schweizer Taschenmesser' für nahezu alle Tätigkeiten rund um die automatisierte Webdatenerhebung und den weiteren Verlauf der Datenanalyse.

3.2 Datensammlung aus statischen Webseiten

In diesem Abschnitt nun wollen wir uns das konkrete Vorgehen bei der Webdatensammlung anschauen. Dabei gehen wir von statischen Webseiten aus. Wie bemerkt sind statische Seiteninhalte nicht mehr zwingend gängige Praxis; dynamische Elemente, wie beispielsweise das oben angesprochene Nachladen von Bildern bei der Google-Bildersuche, gewinnen für moderne Webarchitekturen zunehmend an Bedeutung (vgl. Abschn. 5). Ebenso blenden wir komplexe Problemstellungen aus, die sich etwa in der Ansprache von Formularen oder bei Fragen der Authentifizierung ergeben. Diese Elemente führen weit über einen einführenden Beitrag hinaus. Für Lösungsansätze bei entsprechenden Problemen verweisen wir auf die weiterführende Literatur, die in Abschn. 6 dargestellt ist. Dennoch reichen selbst grundlegende Kenntnisse der zentralen Webtechnologien und einfache Befehle aus, um bereits viele Aufgaben in der Webdatensammlung bearbeiten zu können.

Grundsätzlich ist zu bemerken, dass bei der Verwendung von HTML schlicht Textdateien interpretiert werden. Das bedeutet, dass sichtbarer Text durch Strukturinformationen eingerahmt wird, die für Nutzer üblicherweise nicht sichtbar sind. Dabei hat der Browser die Funktion, die Strukturinformationen zur Darstellung einer Seite zu interpretieren. So werden beispielsweise Verlinkungen zwischen Webseiten durch ein <a>-Element hergestellt:

```
<a href="https://de.wikipedia.org/wiki/
Liste_der_Mitglieder_des_Deutschen_Bundestages_(18._Wahlperiode)">18. Deutscher Bundestag</a>.
```

In diesem Beispiel wird ein Link zur Wikipedia-Seite des 18. Deutschen Bundestags hergestellt, die wir bereits im ersten Abschnitt dieses Kapitels betrachtet haben. Wenn dieser Ausdruck im Quellcode einer Seite vorkäme, dann wäre für Nutzer lediglich der Baustein zwischen den beiden Strukturelementen mit einem klickbaren Link sichtbar: `18. Deutscher Bundestag`.

Die Interpretation der Strukturinformationen durch den Browser gelingt, da die Zahl der möglichen Strukturelemente begrenzt ist und, wichtiger noch, die vorhandenen Elemente eine feste Bedeutung besitzen. So fasst beispielsweise ein <a>-Element immer einen Link ein, ein <p>-Element immer einen Absatz, das -Element wird stets verwendet, um den sichtbaren Ausdruck fett zu formatieren, und so weiter. Diese Einsicht ist für die automatische Datensammlung zentral. Die Webdatensammlung wird ganz erheblich dadurch erleichtert, dass nur wenige, wiederkehrende Strukturelemente im gesamten Netz verwendet werden und diese stets dieselbe Funktion erfüllen. Die Vielfalt von Webseiten ist also ausschließlich eine oberflächliche: Die sichtbare Varianz wird erzeugt durch die Trennung von Struktur und Darstellung. Für letzteres ist der CSS-Standard entscheidend, der die sichtbare Darstellung jenseits der grundlegenden Struktur der Seite beeinflusst. Wir beschäftigen uns in diesem Kapitel nicht weiter mit CSS, da dieser Standard für das Web Scraping weniger relevant ist. Er wird uns jedoch noch einmal begegnen, wenn wir uns damit beschäftigen, wie wir gezielt Elemente im HTML-Code ansprechen und auslesen können.

Aus dem oben dargestellten Link können wir lernen, dass Strukturelemente einen Ausdruck üblicherweise einrahmen, das heißt, sie werden geöffnet <a> und wieder geschlossen . Wir könnten den sichtbaren Ausdruck also beispielsweise kursiv darstellen, indem wir ihn außerdem durch ein -Element einfassen: 18. Deutscher Bundestag. Schließlich können wir lernen, dass Attribute verwendet werden, um weitere Informationen bereitzustellen, in diesem Fall das Ziel der Verlinkung. Dies geschieht mit dem Attribut href. Dem Element können durchaus weitere Attribute hinzugefügt werden. So könnten wir etwa das Verhalten des Browsers steuern, indem wir vorgeben, dass beim Klicken des Links ein neuer Tab im Browser geöffnet wird, statt die verlinkte Seite im aktiven Tab zu öffnen. Dazu würden wir das Attribut target mit dem Wert "_blank" verwenden: 18. Deutscher Bundestag.

Kommen wir nun zu der Frage zurück, wie wir uns bei der Webdatensammlung zunutze machen, dass das gesamte Netz auf denselben Strukturelementen basiert. Durch diese Uniformität können wir mit ganz wenigen Befehlen ausgesprochen aufwendige Daten gewinnen. Als Beispiel sei noch einmal auf Analysen von Hyperlink-Netzwerken in abgegrenzten Politikfeldern hingewiesen (Ackland und O'Neil 2011; McNutt und Pal 2011). Solchen Arbeiten liegt häufig die Annahme zugrunde, dass die Verlinkungspraxis zwischen den Webpräsenzen von Interessengruppen Informationen über die Beziehungen zwischen den Gruppen in der Offline-Welt vermitteln. Wir wissen bereits, dass Links in HTML-Code durch <a>-Elemente gesetzt werden. Um also einen Datensatz aller Verlinkungen auf allen Seiten einer bestimmten Domäne zusammenzustellen und in einem weiteren Schritt einen Datensatz aller Verlinkungen auf den Seiten der gefundenen Domänen, so genügt es, auf allen Seiten eines Servers nach <a>-Elementen zu suchen und die verlinkten Elemente zu speichern, sie aufzurufen und diesen Schritt auf allen Seiten zu wiederholen. So kann mit nur wenigen Zeilen Code ein außerordentlich umfangreicher Netzwerkdatensatz zusammengestellt werden. Dabei ist der eigentliche Datensamm-

lungsprozess durchaus aufwendig, aber letztlich handelt es sich nur um eine Fleißaufgabe, die automatisiert werden kann. Im Übrigen unterscheidet sich auch die Funktionalität von Web Spidern nicht wesentlich von der eben beschriebenen Grundstruktur, und selbst Google operiert im Kern nach diesem Prinzip.

Um den Nutzen der immer gleichen Strukturelemente an einem zweiten Beispiel zu verdeutlichen, sei auf tabellarische Informationen im Quellcode einer Webseite verweisen. Wenden wir uns ein weiteres Mal der Überblicksseite der Mitglieder des 18. Deutschen Bundestages zu. Nehmen wir an, dass wir an der gesamten Information interessiert sind, die in der zweiten Tabelle der Seite dargestellt ist. Wir können die entsprechende Tabelle leicht in R einlesen, wenn wir uns vor Augen führen, dass Tabellen in HTML durch ganz wenige Strukturelemente eingebunden werden. Schauen wir uns hierzu den Quellcode der Seite im Browser an, d. h. ohne, dass die Strukturinformationen vom Browser interpretiert werden. Dies geschieht in den gängigen Browsern, indem Sie auf den Seiteninhalt rechtsklicken und dann die Option für die Anzeige des Quellcodes auswählen. In Google Chrome beispielsweise können Sie auf „View Page Source“ klicken. Ein Ausschnitt aus dem Quellcode ist in Abb. 4 dargestellt. Wir stellen fest, dass die Tabelle durch die Elemente `<table>` und `<tbody>` gerahmt wird, während drei Elemente für die Strukturierung der Tabelleninhalte ausreichen: `<th>` für die Kopfzeile, `<tr>` für die Zeilen und `<td>` für die Abgrenzung der Zellen innerhalb der Zeilen.

Durch die Verwendung weniger Strukturelemente, die im gesamten Netz eine identische Bedeutung haben, kann leicht ein Programm geschrieben werden, welches den HTML-Code von Tabellen interpretiert und die entsprechenden Daten in

```
<table class="wikitable zebra sortable">

<tbody><tr class="hintergrundfarbe5">
<th> Mitglied des Bundestages</th>
<th> geb.</th>
<th> Partei</th>
<th> Bundesland<small><sup>1</sup></small></th>
<th> Wahlkreis<small><sup>1</sup></small></th>
<th> Erststimmen<br /> in %<small><sup>1</sup></small></th>
<th> Bemerkungen
</th></tr>
<tr>
<td data-sort-value="Aken, Jan van"><span id="A"></span><a
href="/wiki/Jan_van_Aken_(Politiker)" title="Jan van Aken (Polit:
Aken</a> </td>
<td> 1961 </td>
<td> DIE LINKE </td>
<td> Hamburg </td>
<td> </td>
<td> </td>
<td>
</td></tr>
<tr>
```

Abb. 4 Ausschnitt des Quellcodes der Seite https://de.wikipedia.org/wiki/Liste_der_Mitglieder_des_Deutschen_Bundestages_(18._Wahlperiode), Stand: 07. August 2017

ein nutzbares Format übersetzt. In R gelingt dies beispielsweise durch das Paket `htmltab`.

```
library(htmltab)
html <- "https://de.wikipedia.org/wiki/Liste_
                der_Mitglieder_des_Deutschen_Bundestages_(18.
_Wahlperiode)"
tab <- htmltab(html, which = 2, rm_nodata_cols = F)
> tab[1:3, 1:4]
  Mitglied des Bundestages geb.    Partei    Bundesland
2             Jan van Aken 1961 DIE LINKE       Hamburg
3           Stephan Albani 1968       CDU Niedersachsen
4        Katrin Albsteiger 1983       CSU        Bayern
```

Falls die gesuchte Information nicht tabellarisch vorliegt, müssen wir den Weg gehen, der bereits im ersten Abschnitt dieses Kapitels angedeutet wurde. Dabei laden wir üblicherweise zunächst den Quellcode einer oder mehrerer Seiten herunter und versuchen dann, die gesuchte Information gezielt herauszufiltern. Für den ersten Schritt ist die Kommunikation zwischen Server und Client zentral, die im Regelfall über das HTTP(S)-Protokoll erfolgt. Wir gehen an dieser Stelle nicht weiter auf diesen Aspekt der webbasierten Datensammlung ein. Glücklicherweise stehen in R verschiedene Pakete zur Verfügung, welche die Interaktion in vielen Fällen übernehmen, ohne dass wir in den Prozess eingreifen müssen, etwa durch Anpassung der Default-Optionen. In diesen Fällen spiegelt der Scraping-Prozess die übliche Ansteuerung von Webseiten durch Nutzer mit Hilfe des Browsers, da auch dort nicht in die Kommunikation zwischen Server und Client eingegriffen wird. Die relevante Funktionalität für diesen und auch den weiteren Schritt stellt das Paket `rvest` mit der Funktion `read_html()` bereit, die bereits in Abschn. 2 verwendet wurde.

Für den zweiten Schritt der Datensammlung machen wir uns die Baumstruktur von HTML-Code zunutze. Das bedeutet, dass die verschiedenen Strukturelemente von HTML-Code hierarchisch organisiert sind – sie sind ineinander verschachtelt. Dabei ist ein Dokument von einem `<html>`-Element umklammert, danach folgt ein Abschnitt mit Meta-Informationen und Definitionen, der durch `<head>` umrahmt wird und schließlich der für Nutzer sichtbare Teil in `<body>`. Darunter folgen jeweils weitere Elemente bis zu einer beliebigen Tiefe des Baums. Die hierarchische Struktur können wir bei der Webdatensammlung ausnutzen, indem wir gezielt Informationen in einem bestimmten Element adressieren. Dazu verwenden wir XPath, eine Syntax, mit der die Struktur von HTML-Code abgebildet werden kann. Die Alternative, CSS-Selektoren, erfüllen bei Verwendung einer etwas anderen Syntax dieselbe Funktion. Wenn wir das Beispiel der Bundestagsabgeordneten wieder aufgreifen und das Ziel verfolgen, alle Links aus der Seite auszulesen, dann genügt der XPath "`//a`". Mit diesem Ausdruck teilen wir mit, dass wir alle

<a>-Elemente suchen, die sich irgendwo im Dokument (//) befinden. In R sieht das Ganze wie folgt aus:

```
library(htmltab)
html <- "https://de.wikipedia.org/wiki/Liste_
                der_Mitglieder_des_Deutschen_Bundestages_(18.
_Wahlperiode)"
doc <- read_html(html)
nodes <- html_nodes(doc, xpath = "//a")
```

Dieser Code produziert allerdings eine Menge unerwünschter Information, da auch andere Links als die der uns interessierenden Bundestagsabgeordneten in R importiert werden. Um zum Beispiel nur jene Links auszulesen, die sich innerhalb einer Tabelle befinden, können wir den Ausdruck wie folgt erweitern: „//table//a". Nun werden nur jene Links extrahiert, die sich irgendwo im Dokument unterhalb eines <table>-Elements befinden. Wir können die Baumstruktur aus Abb. 4 freilich auch explizit verwenden. In dem Fall würden wir <a>-Elemente suchen für die das folgende gilt: „//table/tbody/tr/td/a". Hier verwenden wir einen statt zwei Schrägstriche nach dem <table>-Element, um anzuzeigen, dass nur Elemente extrahiert werden sollen, die in der Baumstruktur unmittelbar aufeinander folgen. Eine solche explizite Ansprache eines bestimmten Pfades macht den Ausdruck im Zweifel weniger anfällig für fehlerhafte Extraktionen, allerdings erfordert er erstens eine tiefe Kenntnis der Seitenstruktur und zweitens beraubt er Nutzer womöglich der notwendigen Flexibilität bei der Formulierung des XPaths.

XPath und CSS-Selektoren bieten eine ausgesprochen flexible Syntax zur Datenextraktion. Sie stellt viele Möglichkeiten bereit, die für Einsteiger zunächst etwas unhandlich und auch schwer begreiflich wirken mögen. Für einen ersten Zugriff zur Formulierung eines XPath kann es deshalb gelegentlich hilfreich sein, sich die Funktionalität von Browsern zunutze zu machen. So kann beispielsweise bei Betrachtung des Quellcodes in Google Chrome mit einem Rechtsklick auf ein bestimmtes Element „Copy/Copy XPath" ausgewählt werden. Der so formulierte XPath adressiert zwar nur das spezifische Element, aber es ist vergleichsweise leicht, den entsprechenden Ausdruck so weit zu verallgemeinern, dass er alle gesuchten Informationen herausfiltert, so lange sie mit der gleichen Struktur im Quellcode hinterlegt sind.

Mit Hilfe der genannten Bausteine können bereits grundlegende Web Scraping-Anwendungen selbst geschrieben werden. Für einen tieferen Einblick in die Bestandteile der automatisierten Datensammlung verweisen wir auf die Literatur, die in Abschn. 6 beschrieben ist. Für einen gezielten und vertieften Eindruck der spezifischen Webtechnologien, die in diesem Abschnitt vorgestellt wurden, sei schließlich noch explizit auf die hervorragenden Einführungen und Referenz-Seiten der W3-Schools hingewiesen, die unter https://www.w3schools.com zu erreichen sind.

3.3 Programmierschnittstellen (APIs)

Das Internet ist ein inhärent soziales Medium. Die Interaktion zwischen Individuen ist ein grundlegendes Merkmal einiger der populärsten Anwendungen wie etwa Twitter oder Facebook. Diese Unternehmen begründen ihre Geschäftsmodelle auf den aus Interaktionen resultierenden Daten über individuelles Verhalten. Dabei überlagern sich wirtschaftliche und sozialwissenschaftliche Interessen, da die entsprechenden Daten zugleich die größte je dagewesene Ansammlung menschlicher Verhaltensdaten darstellen. Nicht zuletzt im Wissen um den Wert dieser Daten für die Erforschung gesellschaftlicher Dynamiken gewährt eine Reihe von Plattformen den kostenlosen Zugang zu ausgewählten Daten. Dazu verwenden Twitter, Facebook und viele weitere Anbieter sogenannte Programmierschnittstellen (*Application Programming Interfaces*, APIs).

Der Grundgedanke von APIs ist denkbar simpel: Ein Datenanbieter stellt eine Infrastruktur zur Verfügung, mithilfe derer Daten abgefragt werden können. Gerade im Bereich der automatischen Webdatensammlung liegt der Nutzen von APIs in der Möglichkeit, Programme an den Schnittstellen anzudocken, um Datenabfragen zu erleichtern. So wurden für R zahlreiche Zusatzpakete geschrieben, die auf spezifischen Services aufbauen. Die Twitter-API beispielsweise kann mit dem Paket `rtweet` angesteuert werden, welches unter anderem die Funktion `search_tweets()` bereitstellt, mit dem die Suche von Tweets zu bestimmten Themen unmittelbar aus R heraus vorgenommen werden kann.[7]

Schauen wir uns als Beispiel ein letztes Mal die Wikipedia an: Unter https://wikimedia.org/api/rest_v1/ dokumentiert das Wikimedia-Team, welche Daten aus Wikipedia-Projekten abgerufen werden können, ohne – wie im Eingangsbeispiel illustriert – zuerst HTML-Seiten herunterladen zu müssen, um dann den Inhalt von Interesse aus dem HTML-Code zu filtern. Dazu gehören etwa Meta-Daten, die bei der Nutzung der Plattform anfallen. So wird z. B. für alle Artikel die Anzahl der täglichen Aufrufe zur Verfügung gestellt. Um an diese Daten zu kommen, braucht es mit dem nutzergeschriebenen Paket `pageviews` nicht mehr als eine Zeile Code. Hier fragen wir die Abrufstatistiken für die Einträge zu Angela Merkel und Martin Schulz zwischen dem 01.09.2016 und dem 01.07.2017 ab:

```
> library(pageviews)
> merkel_views <- article_pageviews(project = "de.wikipedia",
article = "Angela Merkel", user_type = "user", start =
"2016090100", end = "2017070100")
```

(Fortsetzung)

[7] Die Bedeutung von Programmierschnittstellen für moderne Web-Infrastrukturen geht weit über die Bedürfnisse der automatischen Datensammlung hinaus. Auch Anwendungen von Drittanbietern, die auf Plattformen wie Twitter oder Facebook aufsetzen, machen sich die Zugangs- und Abfragemöglichkeiten von APIs zunutze, beispielsweise der bekannte Twitter-Client Tweetbot.

```
> schulz_views <- article_pageviews(project = "de.wikipedia",
article = "Martin Schulz", user_type = "user", start =
"2016090100", end = "2017070100")
```

Die innerhalb weniger Sekunden heruntergeladenen Daten lassen sich sofort mit minimalem Aufwand analysieren:

```
> plot(ymd(merkel_views$date), merkel_views$views, col =
"black", type = "l")
> lines(ymd(schulz_views$date), schulz_views$views, col =
"darkgrey")
```

Das leicht polierte Ergebnis ist in Abb. 5 dargestellt. Auf den ersten Blick wird der Zeitpunkt der Nominierung von Martin Schulz als SPD-Kanzlerkandidat Ende Januar deutlich, als über 150.000 Besucher – augenscheinlich in Reaktion auf die Bekanntgabe – dessen Seite abrufen.

In den letzten Jahren hat die Menge der über APIs verfügbaren, sozialwissenschaftlich relevanten Webdaten rasant zugenommen.[8] Bemerkenswert ist darüber hinaus, dass im Zuge der Open-Government-Bewegung und entsprechender Gesetzgebung (z. B. der *Freedom of Information Act* oder das *Informationsfreiheitsgesetz*) immer mehr Verwaltungen dazu übergehen, amtliche Daten auf diese Weise zur Verfügung zu stellen. Viele dieser und anderer Datenquellen sind bereits über

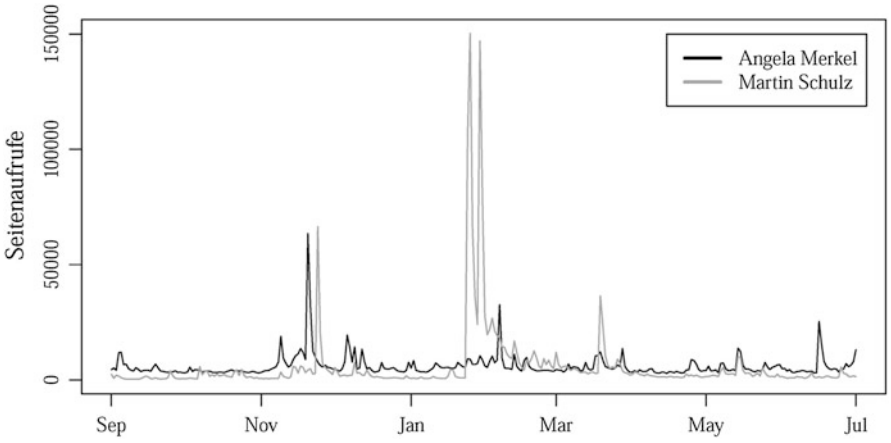

Abb. 5 Seitenabrufe der Einträge zu Angela Merkel und Martin Schulz in der deutschsprachigen Wikipedia, September 2016 bis Juli 2017

[8]Einen Überblick bietet das Projekt *rOpenSci* (https://ropensci.org/).

R-Pakete erschlossen, sodass sich der Datenerhebungsprozess häufig nicht viel komplexer gestaltet als eben beschrieben. Wenn eine solche Anbindung noch nicht existiert, kann sie meist ohne allzu großen Aufwand selbst geschrieben werden.

Grundsätzlich ist das Datensammeln über APIs der Goldstandard der Webdatengewinnung, da im Vergleich zum klassischen Screen Scraping, dem Herunterladen von Webseiten und der Extraktion von Information aus HTML-Code, einige Probleme entfallen: Erstens kommen die Daten ohne störende Layout-Information beim Endnutzer an (häufig im XML- oder JSON-Format, das leicht in gängige Datenstrukturen umgewandelt werden kann). Zweitens ist der Prozess der Datensammlung über APIs üblicherweise robuster: Im Gegensatz zu Webseiten sind Änderungen der Datenstrukturen meist leichter nachvollziehbar und treten außerdem seltener auf. Schließlich kann der Nutzer davon ausgehen, dass der Anbieter mit der Verwendung seiner Daten einverstanden ist. Ethische oder rechtliche Fragen wie „Darf ich die Server einer Webseite ungefragt durch viele tausend Anfragen belasten?“ oder „Darf ich die heruntergeladenen Daten für eine wissenschaftliche Auswertung nutzen?“ sind meist von vornherein geklärt. Es ist deshalb lohnenswert, sich mit der Funktionsweise von APIs vertraut zu machen und im konkreten Fall frühzeitig zu klären, ob die benötigten Daten über eine entsprechende Schnittstelle verfügbar sind.

4 Potenziale und Herausforderungen webbasierter Sozialwissenschaft: Ausgewählte Anwendungen

Um die Potenziale und Problemstellungen von Webdaten für die Sozialwissenschaft zu illustrieren, werden die jüngeren Entwicklungen im Folgenden anhand ausgewählter Forschungsbeiträge diskutiert. Monroe (2013) fasst die Chancen und Herausforderungen von *Big Data* für die Politikwissenschaft in fünf „Vs“ zusammen: Volumen, Verbreitungsgeschwindigkeit, Vielfalt, Verknüpfung und Validität.[9] Wir strukturieren die folgenden Darstellungen entlang dieser fünf Vs, da mit ihrer Hilfe gut illustriert werden kann, dass die Entwicklungen der letzten Jahre über die Verfügbarkeit von Datenmengen ungekannten Ausmaßes hinausgehen, und auf bislang zu wenig beachtete Probleme hinweist.

Die Massennutzung sozialer Medien zur Kommunikation, Netzwerkpflege und Meinungsäußerung und das damit verbundene *Datenvolumen* birgt aus politikwissenschaftlicher Sicht gewaltige Potenziale – sie bietet eine nie dagewesene Grundlage für die empirische Mikrofundierung existierender Theorien politischen Handelns, aber auch für die Entwicklung neuer Methoden zur Messung politischer Einstellungen. Darüber hinaus erlauben stetig wachsende Rechenkapazitäten die Zusammenführung und Auswertung von Datenquellen. Neuere, umfragebasierte Wahlvorhersageansätze, wie sie zuletzt bei der US-Präsidentschaftswahl angewandt wurden, sind nicht zuletzt deshalb so erfolgreich, weil sie auf Massen an veröffentlichten Meinungsumfragen zurückgreifen und diese effizient bündeln (Linzer 2013; Silver 2012).

[9]Im Original „volume, velocity, variety, vinculation, and validity“ (Monroe 2013, S. 1).

Daten aus sozialen Netzwerken und vielen weiteren Diensten werden heute in Echtzeit erfasst und veröffentlicht – in einer zuvor ungekannten *Verbreitungsgeschwindigkeit*. Solche Datenströme erlauben die Analyse von Ereignissen, die mit klassischen Erhebungsmethoden kaum zu erfassen sind, etwa spontan ausbrechende Aufstände oder Reaktionen auf Katastrophen. Ebenso können Dynamiken sozialer Phänomene besser erfasst werden als durch Befragungen, die öffentliche Meinung vergleichsweise statisch abbilden. So nutzt etwa Mellon (2013) den Service Google Trends zur Messung von Issue-Salienz, die sonst nur in grobem zeitlichem Raster und auf wenige thematische Felder beschränkt mit Most-Important-Problem-Batterien zu erfassen ist. Shaw und Hill (2014) analysieren Paradaten aus über 600 Online-Wikis, um Organisationsdynamiken zu beobachten, wie sie bereits in Michels „ehernem Gesetz der Oligarchie" postuliert wurden (Michels 1911). King et al. (2013) schließlich erheben Inhalte aus Millionen von Mitteilungen auf chinesischen Social-Media-Plattformen und spüren damit die Logik der behördlichen Zensur auf, nämlich die Unterdrückung von Nachrichten mit Mobilisierungspotenzial (King et al. 2017).

Die *Verknüpfung* unterschiedlicher Datenquellen birgt großes Potenzial für die sozialwissenschaftliche Forschung. Menschliches Verhalten findet immer in einem sozialen Umfeld statt: Politische Einstellungen bilden sich in der Familie und Netzwerken, Wahlentscheidungen werden nach einem Austausch mit Freunden und Bekannten getroffen, Konflikte zwischen Gruppen oder Staaten ausgetragen. Soziale Netzwerke bilden Bekanntschaften unmittelbar ab und liefern dabei ein vollständigeres Bild als etwa berichtete Freundschaftsnetzwerke in standardisierten Befragungen. Derartige Informationen sind vielfältig nutzbar. Barberà (2015) entwickelt ein Verfahren, um aus Twitter-Follower-Netzwerken, die viele Hunderttausend Nutzer umfassen, die ideologischen Präferenzen politischer Akteure zu schätzen. Darüber hinaus haben kommerzielle und nichtkommerzielle Organisationen das Potenzial der Verknüpfung von Daten erkannt, um beispielsweise Käuferprofile anzulegen oder zielgenau Spenden zu akquirieren. Auch solche Daten sind bereits politikwissenschaftlich genutzt worden: So greift Bonica (2013) auf einen Datensatz zurück, der über 100 Millionen Spendeneingänge bei US-amerikanischen Wahlen umfasst, um politische Positionierungen von Kongressmitgliedern, der Exekutive, Interessengruppen und Wählern zu schätzen.

Die *Vielfalt* neuer Datenquellen deutet sich in den genannten Arbeiten bereits an. Ein häufiges Merkmal neu erschlossener Datenquellen ist die Zweckentfremdung von Daten zur Verhaltensmessung. Eine weitere Anwendung, die existierende Daten kreativ nutzt, präsentieren Kuhn und Weidmann (2015): Sie verwenden hochauflösende Satellitendaten zur Lichtstärkemessung (Chen und Nordhaus 2011; Henderson et al. 2011), um – in Verbindung mit geokodierten Daten über Siedlungsstrukturen – ein Proxy-Maß für Ungleichheit innerhalb ethnischer Gruppen zu entwickeln (Cederman et al. 2015; Weidmann und Schutte 2017). So lassen sich Untersuchungen auf Gebiete ausweiten, für die keine reliablen Datenquellen vorliegen.

Die *Validität* schließlich ist im Gegensatz zu den vorgenannten Merkmalen eher eine Herausforderung als ein Potenzial der Webdatensammlung. Entgegen der bisweilen kolportierten Vorstellung, dass Big Data sozialwissenschaftliche Theorie obsolet mache, sprechen natürlich auch Datenmassen nicht für sich selbst. Mehr

noch: Wenn prozessgenerierte Daten, wie es Webdaten häufig sind, zur Messung sozialer Phänomene genutzt werden, stehen Anwender in der Bringschuld für den Validitätsbeweis. Sozialwissenschaftler haben sich über die letzten Jahrzehnte Fähigkeiten zur Beurteilung von Datenquellen angeeignet. Selektions- und Messfehlerproblematiken in Umfragedaten, aber auch in bekannten, ländervergleichenden Datensätzen sind bekannt und können unter Umständen mit geeigneten statistischen Verfahren korrigiert werden. Im Umgang mit neuen Datenquellen wird diese Erfahrung erst seit einigen Jahren aufgebaut. Wen oder was repräsentieren aus Twitter-Daten extrahierte und aggregierte Meinungen (Nyhuis und Faas 2018)? Wie vollständig und sauber aufbereitet sind von Dritten angelegte Datensätze? Verzerren unbeobachtete konfundierende Faktoren die Validität von Beobachtungen? Es bedarf deshalb sowohl starker Theorien als auch rigoroser Validierungsanstrengungen, um die Nutzung von häufig nicht-reaktiven Messungen menschlichen Verhaltens als Proxy für politikwissenschaftliche Konzepte zu rechtfertigen. Geschieht dies nicht, fällt die sozialwissenschaftliche Forschung hinter schmerzliche erworbene Lernfortschritte zurück. Als das US-Magazin *Literary Digest* 1936 versuchte, mithilfe einer großangelegten Umfrage den Gewinner der anstehenden Präsidentschaftswahl vorherzusagen, beteiligten sich etwa 2,3 Millionen Leser – im damaligen Maßstab zweifellos Big Data. Trotzdem lag die Prognose – ein Sieg des Republikaners Landon – fatal daneben, was in erster Linie auf die Auswahl eines Teils der Teilnehmer auf Basis von Telefonregistern zurückgeführt wurde, denn: Telefonanschlüsse fanden sich damals vor allem in reichen, republikanischen Haushalten (Squire 1988). Deutlicher kann kaum veranschaulicht werden, dass der Umfang allein noch nicht die Nutzung einer Datenquelle rechtfertigt.

5 Gegenwärtige Problemfelder

Ist für die Sozialwissenschaft angesichts der Datenflut im Netz ein goldenes Zeitalter angebrochen, in dem Fragestellungen über menschliches Verhalten, über soziale, politische und wirtschaftliche Interaktion genauer als je zuvor beantwortet werden können? Es lohnt sich, die skizzierten Entwicklungen mit einem gesunden Maß an Skepsis zu verfolgen. Im Folgenden seien einige Problemfelder angedeutet, die sich aus unserer Sicht gegenwärtig abzeichnen.

Technische Hürden
Klassische Datenmanagement- und Analysestrategien stoßen schnell an ihre Grenzen, wenn die Masse der Daten die Kapazität des Arbeitsspeichers überschreitet. Darüber hinaus erfordern unstrukturierte Datenquellen (z. B. HTML-Code) oder in den Sozialwissenschaften noch wenig verbreitete Datenaustauschstandards (z. B. XML und JSON) spezielles Wissen zur Datenextraktion; textbasierte Ressourcen setzen außerdem Erfahrung im Umgang mit Techniken des Text Mining voraus. Während die Verfügbarkeit von APIs tendenziell zunimmt, werden konventionelle Webseiten zunehmend „verskriptet“, also für die Einbindung dynamischer Elemente durch (zumeist JavaScript-)Code angereichert. Diese Entwicklung

erschwert das klassische Screen-Scraping. Es ist also durchaus denkbar, dass sich die aktuelle Goldgräberstimmung – jeder kann beliebige Daten aus dem Netz gewinnen – bald in Frustration darüber umschlägt, dass viele Informationen nur noch schwer automatisiert gesammelt werden können. Eine weitere Hürde des Web Scraping entsteht durch die zunehmende Bedeutung von mobilen Plattformen. Bei der klassischen, webbasierten Datensammlung werden Webseiten ausgelesen. Es gibt dagegen immer mehr sozialwissenschaftlich relevante Datenbestände, die ausschließlich über Apps zugänglich sind. Zwar können auch solche Daten durchaus ausgelesen werden, entsprechende Vorhaben sind jedoch komplexer.

Proprietäre Dienste und Zentralisierung des Internets
Wenn nur wenige Monopolisten das Internet dominieren, ist das für Web Scraping-Anwender ein Problem, da der Zugriff auf solchen Plattformen häufig eingeschränkt ist, und die Verbreitung von Daten teils explizit ausgeschlossen wird. Wenn das Unternehmenskapital im Wissen über die Nutzer besteht, dann gibt es verständlicherweise Vorbehalte gegen das Anlegen von Kopien. Dies ist beispielsweise der Fall für Google und Facebook und gilt ebenso für die meisten Medienseiten, die für die sozialwissenschaftliche Forschung häufig besonders interessant sind. Twitter ist in der Sozialwissenschaft nur deshalb ein derart stark erforschtes Phänomen, da es einen vergleichsweise einfachen Zugriff auf seine Daten gewährt.

Sozialwissenschaftliche Forschungsagenda
Die Arbeit mit Webdaten bedeutet die Beschränkung auf einen bestimmten Teil der Population – und hier wiederum nur auf bestimmte Merkmale. Webbasierte Daten sollten aber nicht deshalb zum Standard werden, weil sie am bequemsten zu erheben sind. Auch wenn sich immer mehr soziales Verhalten in digitalen Räumen abspielt, gilt stets sorgfältig abzuwägen: Über welche Population können und sollen Aussagen getroffen werden? Inwiefern lassen sich die für eine bestimmte Population gewonnenen Erkenntnisse generalisieren? Diese Überlegungen werden dadurch erschwert, dass klassische Korrekturverfahren wie z. B. die Gewichtung auf Basis relevanter Merkmale häufig nicht angewandt werden können, da individuelle Strukturinformationen fehlen. In einem weiteren Sinne sollte das Erkenntnisinteresse handlungsleitend für den sozialwissenschaftlichen Forschungsprozess sein. So stellen Daten, egal ob groß oder klein, lediglich ein Mittel zum Zweck dar. Es muss jedoch in jedem Fall deutlich werden, welche sozialwissenschaftlich und gesellschaftlich relevante Fragestellung bearbeitet wird, um nicht mit empirischen Kanonen auf theoretische Spatzen zu schießen.

Vermittlung von Fähigkeiten im Rahmen des sozialwissenschaftlichen Studiums
Um das Potenzial neuer Datenquellen für die sozialwissenschaftliche Forschung nutzen zu können, bedarf es technischer Kompetenzen. Dabei ist das Programmieren in skriptbasierter Statistiksoftware eine Schlüsselkompetenz. Die bisherige Ausbildung bereitet noch zu wenig auf neue Entwicklungen vor. Während in den meisten Fällen lediglich Einführungen in die Methoden der empirischen Sozialforschung und

Statistik zum Standardrepertoire der Methodenausbildung gehören, müssen sich Studierende die ersten Schritte der Datensammlung und -aufbereitung oftmals mühsam selbst aneignen. Die Herausforderung für die Lehrverantwortlichen besteht deshalb darin, Einheiten zur Vermittlung dieser Kompetenzen in den ohnehin schon stark gefüllten Curricula unterzubringen.[10]

6 Kommentiertes Literaturverzeichnis

Die Themen dieses Kapitels sind in deutlich größerem Umfang in Munzert et al. (2014) dargestellt. Neben einem Überblick über die Grundlagen der gängigen Webtechnologien (HTML, XML, HTTP, etc.) bietet das Buch eine anwendungsorientierte Einführung in die Praxis des Web Scraping mit R, ergänzt durch praktische Fallstudien. Da Webdaten häufig aus wenig strukturiertem Text bestehen, wird zudem eine Einführung in die automatische Textverarbeitung gegeben. Hier werden Möglichkeiten zur Generierung sozialwissenschaftlich nutzbarer Datenpunkte aus großen Textbeständen aufgezeigt. Die rasante Entwicklung der verfügbaren R-Pakete im Bereich der automatisierten Datensammlung, ebenso wie die permanente Überarbeitung von Webseiten, stellen dabei eine Herausforderung für das Buch dar – und vermutlich für jede Arbeit in diesem Bereich. Sobald die Ausführungen über Grundlagen hinausgehen, besteht die Gefahr, dass getroffene Aussagen durch Entwicklungen überholt werden. Um diesem Problem zu begegnen, stellt eine begleitende Webseite Materialien und, falls nötig, überarbeitete Skripte bereit.[11]

Gandrud (2015) bietet einen ebenfalls stark anwendungsorientierten Einblick in Verfahren zur Sicherstellung der Reproduzierbarkeit des Forschungsprozesses. Dabei geht es allerdings nicht um die automatische Sammlung großer Datenbestände; vielmehr werden Werkzeuge vorgestellt, die es erlauben, den eigenen Arbeitsprozess konsistenter zu gestalten. Neben dem grundsätzlich wichtigen Bemühen, die Reproduzierbarkeit der eigenen Forschung zu gewährleisten, besteht die Verbindung zu den Themen dieses Kapitels in der Automatisierung der Ansprache des eigenen Rechners. Tatsächlich existiert hier eine enge Verwandtschaft mit der Automatisierung der Datensammlung aus dem Web. Selbst die Pfadstruktur von Webseiten, die wir uns beim Web Scraping häufig zunutze machen und die Pfadstruktur auf dem eigenen Rechner sind vergleichbar. Die Fähigkeit zur automatisierten Datensammlung aus dem Web versetzt somit in die Lage, Prozesse auf dem eigenen System zu automatisieren – und umgekehrt. Technisch verwendet Gandrud R und vornehmlich das Paket knitr, welches die Einbindung und Auswertung von Code in Textdokumenten ermöglicht.

Wickham und Grolemund (2017) beschäftigen sich ebenfalls mit Elementen des datengetriebenen Forschungsprozesses, die erst nach der eigentlichen Datensammlung ansetzen. Dabei steht bei den beiden Autoren die Frage im Vordergrund, wie

[10] Vergleiche hierzu ausführlicher Munzert (2018).

[11] Die begleitende Webseite ist frei zugänglich unter http://www.r-datacollection.com.

sich Daten in R einlesen, verarbeiten und visualisieren, bzw. analysieren lassen. Auch hier lässt sich das Bemühen erkennen, den Forschungsprozess stringenter zu gestalten. Zu diesem Zweck wird eine Reihe von R-Paketen vorgestellt, die zu großen Teilen von Hadley Wickham selbst geschrieben wurden. Viele der eingeführten Pakete sind für den Umgang mit webbasierten Daten besonders deshalb nützlich, da bei der Verarbeitung von Daten aus dem Netz die größte Hürde häufig erst im zweiten Schritt zu überwinden ist – dann nämlich, wenn es um die Säuberung von Datenbeständen geht.

Deutlich näher an den Inhalten des vorliegenden Kapitels bewegt sich das Buch von Nolan und Temple Lang (2014), das wie Munzert et al. (2014) in spezifische Aspekte des Web Scraping mit Hilfe von R einführt. Dabei ist die Beschäftigung mit Webtechnologien Klammer des Buches, sodass im Rahmen der Einführung in die verschiedenen Formate und Protokolle Exkurse in die Ableitungen des XML-Standards vorgenommen werden, beispielsweise Excel-Dokumente oder SVG-Graphiken. Wie das zuvor genannte Werk basiert auch diese Einführung häufig auf Paketen, die vom Autor maßgeblich gestaltet wurden. Insgesamt ist das Buch eher anspruchsvoll und weniger anwendungsorientiert als die zuvor genannten Arbeiten und mag deshalb für Einsteiger weniger geeignet sein. Zugleich bietet das Buch jedoch Hilfestellung bei Problemen, die mit Standardlösungen nicht zu bearbeiten sind. Allerdings muss bemerkt werden, dass die ausgesprochen aktive R-Entwicklergemeinde auch diesem Buch zu schaffen macht, da eine Reihe von Paketen nicht mehr dem Stand der Technik entsprechen, und seit dem Erscheinen des Buches neuere, bessere Lösungen veröffentlicht wurden.

Wie dargestellt, ist hinsichtlich der Verwendung von Software unsere Präferenz und Empfehlung für Einsteiger in das Web Scraping die Nutzung von R, da Sozialwissenschaftler häufig bereits mit der Sprache in Kontakt gekommen sind, bevor sie sich der automatischen Datensammlung zuwenden. Darüber hinaus bietet R den klaren Vorteil, dass sich der gesamte Forschungsprozess innerhalb eines einheitlichen technischen Rahmens bewerkstelligen lässt – von der Datensammlung über die Analyse und Visualisierung bis zur Anfertigung von druckfertigen Berichten, etwa mit den Werkzeugen, die in den Arbeiten von Gandrud (2015) sowie Wickham und Grolemund (2017) vorgestellt werden. Selbstverständlich ist R aber weder die einzige und für viele Beobachter noch nicht einmal die naheliegende Sprache für die automatische Datensammlung. R ist in diesem Bereich eher ein Nachzügler und viele zentrale Pakete sind erst in jüngster Zeit veröffentlicht worden. So wurde beispielsweise das weit verbreitete Paket rvest erst Ende 2014 auf der Plattform CRAN veröffentlicht. Klassischere Sprachen für das Web Scraping sind beispielsweise Perl, PHP oder Python. Auch hier gibt es gute Überblickswerke, etwa das Buch von Mitchell (2015), das ebenfalls stark anwendungsorientiert in übliche Szenarien des Web Scraping einführt und auch komplexere Aspekte wie beispielsweise dynamische Webseiten diskutiert, ohne sich zu sehr in technischen Details zu verlieren.

Auf eine weitere, ausgesprochen hilfreiche Ressource sei schließlich noch explizit hingewiesen. Um bei der Vielzahl an Erweiterungen für R nicht den Überblick zu verlieren, bieten CRAN Task Views eine Übersicht der relevanten Pakete und

Funktionen in einem bestimmten Feld. Der CRAN Task View *Web Technologies* (https://cran.r-project.org/web/views/WebTechnologies.html) stellt alle relevanten Erweiterungen im Bereich der automatisierten Datensammlung zusammen. Hier werden in einer sehr umfassenden Sichtung der verfügbaren Ressourcen sowohl grundlegende Werkzeuge zum Web Scraping als auch hoch spezialisierte Pakete aufgeführt. Dabei mag besonders die Übersicht der API-Anbindungen hilfreich sein, wenn ganz spezifische Funktionen zur Bearbeitung einer Forschungsfrage benötigt werden.

Literatur

Ackland, Robert, und Mathieu O'Neil. 2011. Online collective identity: The case of the environmental movement. *Social Networks* 33(3): 177–190.

Barberà, Pablo. 2015. Birds of the same feather tweet together: Bayesian ideal point estimation using Twitter data. *Political Analysis* 23(1): 76–91.

Benoit, Kenneth, Drew Conway, Benjamin E. Lauderdale, Michael Laver, et al. 2016. Crowd-sourced text analysis: Reproducible and agile production of political data. *American Political Science Review* 110(2): 278–295.

Bonica, Adam. 2013. Ideology and interests in the political marketplace. *American Journal of Political Science* 57(2): 294–311.

Cederman, Lars-Erik, Nils B. Weidmann, und Nils-Christian Bormann. 2015. Triangulating horizontal inequality: Toward improved conflict analysis. *Journal of Peace Research* 52(6): 806–821.

Chen, Xi, und William D. Nordhaus. 2011. Using luminosity data as a proxy for economic statistics. *Proceedings of the National Academy of Sciences* 108(21): 8589–8594.

Gandrud, Christopher. 2015. *Reproducible research with R and RStudio*. Boca Raton: CRC.

Henderson, Vernon, Adam Storeygard, und David N. Weil. 2011. A bright idea for measuring economic growth. *American Economic Review* 101(3): 194–199.

King, Gary, Jennifer Pan, und Margaret E. Roberts. 2013. How censorship in China allows government criticism but silences collective expression. *American Political Science Review* 107(2): 326–334.

King, Gary, Jennifer Pan, und Margaret E. Roberts. 2017. How the Chinese government fabricates social media posts for strategic distraction, not engaged argument. *American Political Science Review* 111(3): 484–501.

Kuhn, Patrick M., und Nils B. Weidmann. 2015. Unequal we fight: Between- and within-group inequality and ethnic civil war. *Political Science Research and Methods* 3(3): 534–568.

Linzer, Drew A. 2013. Dynamic Bayesian forecasting of presidential elections in the states. *Journal of the American Statistical Association* 108(501): 124–134.

McNutt, Kathleen, und Leslie A. Pal. 2011. ‚Modernizing government': Mapping global public policy networks. *Governance* 24(3): 439–467.

Mellon, Jonathan. 2013. Where and when can we use Google Trends to measure issue salience? *PS: Political Science and Politics* 46(2): 280–290.

Michels, Robert. 1911. *Zur Soziologie des Parteiwesens in der modernen Demokratie: Untersuchungen über die oligarchischen Tendenzen des Gruppenlebens*. Leipzig: Klinkhardt.

Mitchell, Ryan. 2015. *Web scraping with Python: Collecting data from the modern web*. Beijing: O'Reilly.

Monroe, Burt L. 2013. The five Vs of big data political science: Introduction to the virtual issue on big data in political science. *Political Analysis* 21(V5): 1–9.

Munzert, Simon. 2018. Auf dem Weg zu einer fundierten Softwareausbildung in der Sozialwissenschaft. In *Computational Social Science: Die Analyse von Big Data*, Hrsg. Joachim Behnke et al., 379–402. Baden-Baden: Nomos.

Munzert, Simon, Christian Rubba, Peter Meißner, und Dominic Nyhuis. 2014. *Automated web data collection with R: A practical guide to web scraping and text mining*. Hoboken: Wiley.

Nolan, Deborah, und Duncan Temple Lang. 2014. *XML and web technologies for data sciences with R*. New York: Springer.

Nyhuis, Dominic, und Thorsten Faas. 2018. Twitter als Spiegel öffentlicher Meinung? Die Schätzung politischer Bewertungen auf Twitter mittels halbautomatischer Textklassifizierung. In *Computational Social Science: Die Analyse von Big Data*, Hrsg. Joachim Behnke et al., 235–253. Baden-Baden: Nomos.

Shaw, Aaron, und Benjamin M. Hill. 2014. Laboratories of oligarchy? How the iron law extends to peer production. *Journal of Communication* 64(2): 215–238.

Silver, Nate. 2012. *The signal and the noise: Why so many predictions fail – But some don't*. New York: Penguin.

Squire, Peverill. 1988. Why the 1936 Literary Digest poll failed. *Public Opinion Quarterly* 52(1): 125–133.

Weidmann, Nils B., und Sebastian Schutte. 2017. Using night light emissions for the prediction of local wealth. *Journal of Peace Research* 54(2): 125–140.

Wickham, Hadley, und Garrett Grolemund. 2017. *R for data science: Import, tidy, transform, visualize, and model data*. Beijing: O'Reilly.