Was ist los?

Erhebung und Visualisierung von empirischen Daten verschiedener Internet-Zeitungen

Halbjahresarbeit 2016/2107 von **Jaro Habiger**

Inhaltsverzeichnis

1 Idee

In meiner Freizeit programmiere ich gerne. Ich mag es, Probleme mit Computern oder anderen technischen Geräten zu lösen. Auch gewinne ich hierbei oft neue Erkenntnisse, da man sich beim Programmieren unkonventionell mit den Dingen beschäftigen muss: Man muss mehr in Strukturen denken, muss versuchen einem Computer, der kein Verständnis von Sinn oder Inhalt hat, das Problem nur in logischen Zusammenhängen zu beschreiben. Somit kann mithilfe eines Computerprogramms fast nie eine wirkliche Lösung für ein Problem oder eine Antwort auf eine Frage gefunden werden, allerdings können Computer einem Menschen helfen, eine bestimmte Aufgabe besser, schneller oder einfacher zu erledigen.

Besonders interessant finde ich es immer dann, diese Macht, die wir uns mit der Programmierung von Computern geben können, zu verwenden, wenn wir sie nutzen um neue Einblicke in unsere Umwelt zu erlangen. Auch hier gilt zwar wieder, dass der Computer eigentlich viel dümmer als wir ist, und keine Zusammenhänge begreifen kann, allerdings stumpfe Aufgaben viel schneller und ausdauernder zu erledigen vermag. Diese Kombination aus Eigenschaften ermöglicht es, große Datensätze mit mathematischen Methoden zu analysieren und die Ergebnisse für Menschen ansprechend aufzubereiten. Bei dieser Aufbereitung bietet sich eine optische Darstellung an, da die Augen des Menschen das Sinnesorgan sind, das am schnellsten Daten erfassen kann [1]. Es bietet sich also an, empirische Daten grafisch aufzubereiten, sprich Datenvisualisierung zu betreiben.

Relativ schnell war mir also klar, das ich mich mit Datenvisualisierung beschäftigen will. Anfangs war allerdings nicht ganz klar, welchen Datensatz ich auf welche Aspekte hin untersuchen werde. Eines Tages nahm ich dann eine Ausgabe des Spiegels zur Hand und sah auf der ersten Seite eine sogenannte Wortwolke, in der oft verwendete Worte größer gedruckt waren, als weniger oft verwandte. Diese Form der Darstellung fand ich sofort sehr ansprechend. Ich entschied mich also dazu, verschiedene Texte über das aktuelle Weltgeschehen, die ich von verschiedenen Online-Zeitungen herunterladen kann, zu analysieren. Diese Analyse wollte ich von Anfang an nicht unbedingt nur auf die Generierung von Wortwolken beschränken, sondern mit dem Datensatz "herumexperimentieren", also gucken, welche interessanten Visualisierungen möglich sind.

Also fing ich an und suchte Zeitungen aus.

2 Die Auswahl der Zeitungen

Als das Thema der Halbjahresarbeit feststand, musste ich Zeitungen aussuchen, die ich analysieren wollte. Hierbei orientierte ich mich an einer Studie der Arbeitsgemeinschaft Online Forschung aus dem November 2014. Hierbei ging ich nach der Anzahl der "Unique Users", also quasi der Leser der jeweiligen Nachrichtenseite. Nach dieser Studie [2] sind die 10 Nachrichtenseiten mit den meisten Lesern:

Rang	Nachrichtenseite	Leser (in Milionen)
1	Bild.de	16,91
2	Focus Online	13,65
3	Spiegel Online	11,43
4	Die Welt	9,56
5	Süddeutsche.de	7,41
6	stern.de	6,45
7	Zeit Online	5,78
8	n-tv.de	4,69
9	FAZ.net	4,54
10	N24.de	4,07

Von diesen Online-Zeitungen habe ich nun Bild.de, Focus Online sowie Spiegel Online ausgewählt. Dies sind die 3 Zeitungen mit den meisten Lesern. Zusätzlich habe ich Süddeutsche.de und Zeit Online ausgewählt, da ich diese Zeitung manchmal selber lese. An dieser Auswahl ist unter anderem sehr interessant, dass sie Nachrichtenquellen verschiedener politischer Ausrichtungen und intellektuellen Anspruchsniveaus enthält. Dies ermöglicht es, am Ende auch Vergleiche zwischen den Zeitungen durchzuführen und eigene Hypothesen leicht zu überprüfen.

3 Konzeptioneller Aufbau

Als nächstes galt es, ein System zu konzipieren, welches die Daten der ausgewählten Zeitungen periodisch herunterlädt, vorverarbeitet und die gesammelten Ergebnisse für spätere Analyse abspeichert. Dieser Schritt ist ein sehr wichtiger, der gut bedacht sein muss, da die am Ende durchführbaren Analysen davon abhängen, welche Daten in diesem Schritt erhoben werden. Ich habe mich dazu entschlossen, die Daten, die ich erhebe, in einer Datenbank zu speichern. Auf diese sollten nun kleine Auswertungsprogramme, die verschiedene Visualisierungen generieren, zugreifen. Diese sollten im Webbrowser laufen, da dieser die Programmierung der Visualisierungen vereinfacht und die Visualisierungen sehr portabel macht. Der gesamte Auswertungsprozess sollte jeweils durch diese Visualisierung vorgenommen werden, um einfacheres Experimentieren mit dem Datensatz zu ermöglichen. Um dies möglich zu machen, ist zusätzlich noch ein "Adapter" zwischen Visualisierung und Datenbank notwendig.

Dieses System sieht am Ende aus wie in Abb. 1 gezeigt.

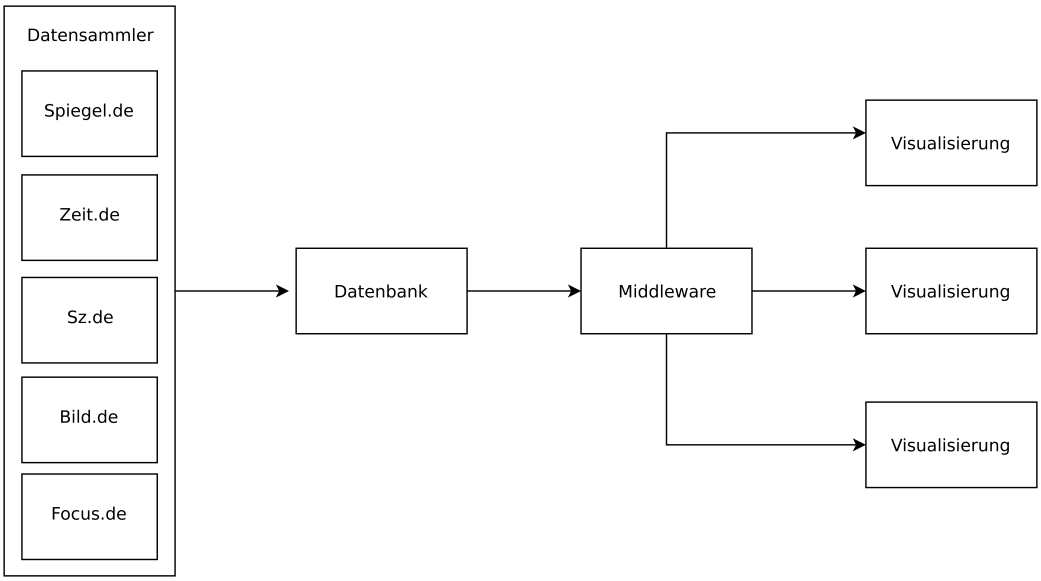

Abb. 1: Der Aufbau des Gesamtsystems

Wie ich dieses System nun umgesetzt habe, ist im nächsten Absatz beschrieben.

4 Umsetzung

Der nun konzipierte Aufbau lässt sich sehr gut in einzelne Teilprobleme unterteilen, was ich bei meiner Umsetzung auch sehr strikt beachtet habe. Hierbei ist es am einfachsten, den Datenfluss von den verschiedenen Nachrichtenquellen bis zur fertigen Visualisierung zu betrachten. In den nachfolgenden Abschnitten wird dieser Verarbeitungsprozess beschrieben. Die einzelnen Schritte sind teilweise eher schwierig zu verstehen, da ihr Sinn wenig greifbar ist. Dies liegt daran, das viele verschiedene Repäsentationen und Strukturen von Daten ineinander umgewandelt werden müssen.

4.1 Daten sammeln

Als erstes müssen Daten zur weiteren Verwertung von den verschiedenen Nachrichtenquellen gesammelt werden. Dies geschieht über die sogenannten "RSS-Feeds". Bei diesen handelt es sich um ein standardisiertes Format, über das Nachrichtenanbieter ihre Artikel, inklusive Metadaten wie z.B. den Zeitpunkt der Veröffentlichung, in maschinenlesbarer Form bereitstellen. Im Prinzip sind also die gleichen Daten wie auf der normalen Internetseite abrufbar, mit dem Unterschied, dass diese einfacher mit Hilfe von Programmen verarbeitet werden können (Siehe Abb. 2 und 3).

Diese Darstellung als RSS-Feed [3] ermöglicht es, die Artikel verschiedener Online-Zeitungen zu betrachten, ohne für jede Zeitung einen komplett neuen Datensammler programmieren zu müssen.

Das Datensammelprogramm ist eine Software, welche ich in Python geschrieben habe. Python ist eine Programmiersprache, die auf Einfachheit und Flexibilität optimiert ist [4]. Das Datensammelprogramm lädt sich den RSS-Feed einer einzelnen Nachrichtenquelle periodisch herunter und verarbeitet ihn weiter. Ich habe mich dazu entschieden, jeden RSS-Feed einmal pro Minute neu zu analysieren. Im ersten Schritt der Verarbeitung lädt das Programm den Feed als Datei von den Servern der jeweiligen Online-Zeitung herunter. Diese Datei ist eine sogenannte XML-Datei. In ihr werden Daten als Baumstruktur abgebildet. Hierbei muss man sich die Datei als "Stamm" des Baums vorstellen. Die ersten "Äste" der Baumstruktur beinhalten die Metadaten, wie z.B. den Erstellungszeitpunkt und den Herausgeber des Feeds. Die

Abb. 2: Ein Artikel einmal in der normalen Darstellung...

```
▼<rss xmlns:content="http://purl.org/rss/1.0/modules/content/" version="2.0">
  ▼<channel>
     <title>SPIEGEL ONLINE - Schlagzeilen</title>
     <link>http://www.spiegel.de</link>
    ▼<description>
       Deutschlands führende Nachrichtenseite. Alles Wichtige aus Politik,
       Wirtschaft, Sport, Kultur, Wissenschaft, Technik und mehr.
     </description>
     <language>de</language>
     <pubDate>Sun, 11 Dec 2016 20:15:15 +0100</pubDate>
     <lastBuildDate>Sun, 11 Dec 2016 20:15:15 +0100</lastBuildDate>
    ▼<image>
       <title>SPIEGEL ONLINE</title>
       <link>http://www.spiegel.de</link>
       <url>http://www.spiegel.de/static/sys/logo_120x61.gif</url>
     </image>
    ▼<item>
      ▼<title>
         Gut zwei Promille: Ostfriese sucht Reeperbahn in Braunschweig
       </title>
      ▼<link>
         http://www.spiegel.de/panorama/gesellschaft/ostfriese-sucht-reeperbahn-in-
         braunschweig-a-1125427.html#ref=rss
       </link>
```

Abb. 3: ... und einmal als Teil eines RSS-Feeds

nachfolgenden “Äste” enthalten jeweils einen Artikel. Diese Artikel wiederum beinhalten verschiedene Unterelemente, d.h. die “Äste” verzweigen sich in weitere, kleinere “Ästchen”. In diesen stehen nun z.B. Autor, Titel, oder eine Kurzfassung des Textes. Außerdem ist ein Link zur Vollversion des Artikels gegeben. Diese textuelle Repräsentation einer Baumstruktur gilt es nun in eine einfacher verwendbare Repräsentation im Speicher des Python Programms umzuwandeln, also zu “parsen”. Das Wort “parsen” ist sehr schwierig ins Deutsche zu übersetzen. Es beschreibt den Sachverhalt, die textuelle Repräsentation eines strukturierten Datensatzes in die “native” Repräsentation einer Programmiersprache für diesen Datensatz zu bringen. Die “native” Repräsentation eines Datensatzes in einer Programmiersprache ist die Standardform, diesen Datensatz abzubilden. Diese hat oft bestimmte Funktionen, die das Programmieren vereinfachen, wie z.B. die Zugriffsmöglichkeit auf einzelne Elemente bei Listen. Hierfür wird eine Programmbiliothek verwendet, die einen sogenannten “XML-Parser” beinhaltet, der genau dies erreicht. Nach diesem Schritt können die einzelnen Artikel betrachtet werden. Hierbei wird zuallererst überprüft, welche Artikel schon einmal verarbeitet wurden. Diese werden verworfen. Die übriggebliebenen, also neuen Artikel werden in eine allgemeinere Repräsentation für Neuigkeiten und ihre Metadaten gebracht, die ich mir überlegt habe. Diese ist auch wieder eine Baumstruktur und sieht aus wie in Abb. 4.

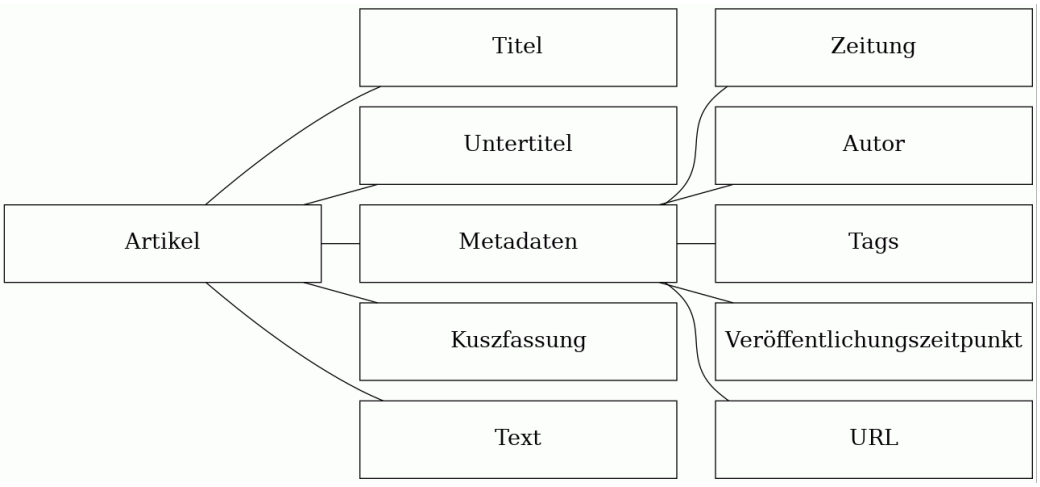

Abb. 4: Die Datenstruktur der z Zwischenrepräsentation der Artikel

Diese Umwandlung ist nötig, da der RSS-Standard zwar die grobe Struk-

tur und ihre Repräsentation als XML-Datei spezifiziert, letzten Endes jeder Nachichtenanbieter allerdings doch nicht ganz kompatible Feeds ausliefert. Diese kleinen Unterschiede werden hier also angeglichen, damit die weitere Verarbeitung leichter von statten gehen kann, und in der weiteren Verarbeitung keine Unterschiede mehr beachtet werden müssen.

Ein weiteres Problem der RSS-Feeds ist, dass sie nur Kurzfassungen der Artikel enthalten. In der Umwandlung müssen also noch die vollständigen Artikel von der Internetseite des Nachrichtenanbieters heruntergeladen und der Volltext aus dieser extrahiert werden. Hierzu wird zuerst das HTML-Dokument der Seite "geparsed". Aus dem "geparsten" HTML-Dokument werden nun die relevanten Teile, zu denen z.B. nicht die Kopf- oder Fußzeile gehören, mithilfe sogenannter "CSS-Selektoren" identifiziert. Hiernach werden alle HTML-Elemente dieser Bereiche in reinen Text umgewandelt.

Architektonisch ist der Datensammler ein Programm, in das weitere kleinere Module "hereingesteckt" werden, die die einzelnen Nachrichtenseiten ansprechen (Siehe Abb. 5).

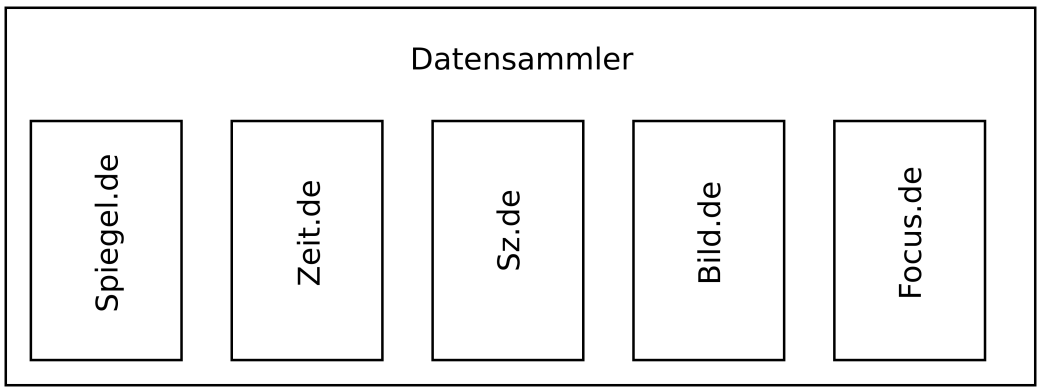

Abb. 5: Der Datensammler und seine Module

Am Ende der Datenaquirierung werden die gesammelten Daten an die nächste Stufe weitergegeben, in der die Daten gespeichert werden.

4.2 Speicherung

Die nun gesammelten Daten müssen vor der weiteren Verwendung ersteinmal strukturiert zwischenggespeichert werden. Dies geschieht in einer Datenbank.

Ich habe mich dafür entschieden, eine sogenannte “NoSQL-Datenbank” zu verwenden, da diese flexiblere Abfragemethoden ermöglichen. “NoSQL” steht hierbei dafür, dass die Datenbank nicht über die Datenbanksprache SQL angesprochen wird, wie es normalerweise der Fall ist, sondern eine andere Abfragemöglichkeit bietet [5]. Relativ früh habe ich mich dafür entschieden, dass ich meine Datenbankabfragen als “MapReduce”-Funktionen formulieren möchte.

4.3 MapReduce

Das MapReduce-Verfahren hat einige große Stärken, wie z.B. die hohe Parallelisierbarkeit und die damit verbundene hohe Geschwindigkeit, bei gleichzeitig hoher Flexibilität. MapReduce ist ein von der Firma Google eingeführtes Programmiermodell [6], welches wie folgt funktioniert (Vgl. Abb. 6):

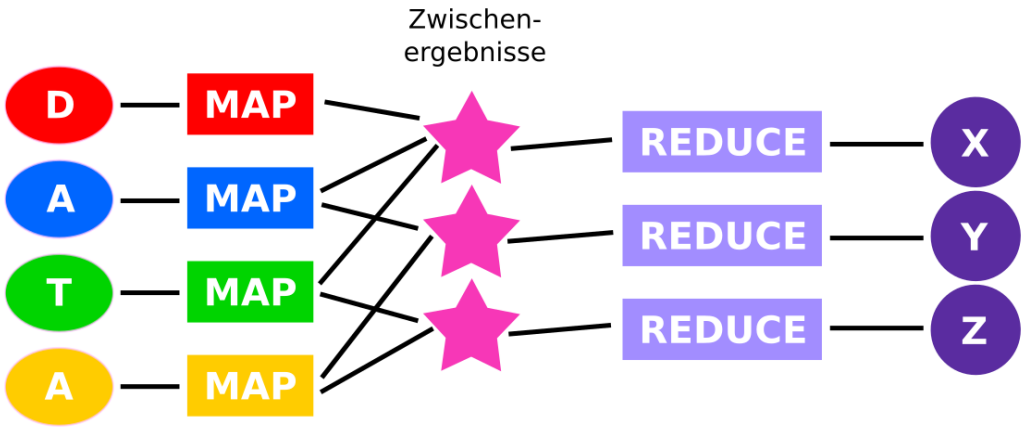

Abb. 6: Der MapReduce Prozess als Grafik [7]

1. Am Anfang des Prozesses steht eine Menge aus n Eingangsdatensätzen. In meinem Fall sind das die Zeitungsartikel, die als Objekte mit der oben beschriebenen Datenstruktur vorliegen. Für jeden dieser Artikel wird jetzt die Map-Funktion ausgeführt. Diese ordnet jedem Artikel n Schlüssel-Wert-Paare zu. In dem Fall, dass wir zu jedem vorkommenden Wort die Häufigkeit bestimmen wollen, ordnet die Map-Funktion also jedem Artikel die Menge der darin enthaltenen Wörter zu. Da diese Funktion auf jeden Artikel angewandt wird, kann man sich in diesem

Fall den gesamten Map-Prozess als eine Funktion vorstellen, die die Menge aller Artikel der Menge der darin enthaltenen Worte zuordnet. Der Code, den ich für dieses Beispiel als Map-Funktion entwickelt habe ist:

```
1  function map() {
2    var text = this.text.replace(/[^A-Za-zÄäÖöÜüß ]/g, " ");   //
        entferne alle überfüssigen Satzzeichen, wie .,?!-
3    var words = text.split(" ");   // Teile den Text in Wörter
4    for (var i = 0; i < words.length; i++) {   // für jedes Wort
5          var word = words[i];
6          if(word) {
7             emit(word, 1);   // füge der Ausgabemenge das Wort
                 hinzu
8          }
9    }
10 }
```

Die etwas seltsam anmutende Zeichenkette `/[^A-Za-zÄäÖöÜüß ]/g` in Zeile 2 ist ein regulärer Ausdruck: Reguläre Ausdrücke sind eine formale Beschreibungssprache für abstrakten Text [8], die man zum Beispiel benötigt, um, wie in diesem Beispiel, alle Zeichen, die nicht A-Z, a-z, Ö, ö, Ä, ä, Ü, ü, oder ß sind durch ein Leerzeichen zu ersetzen. Reguläre Ausdrücke werden uns auch im weiteren Verlauf noch oft begegnen, da sie essenziell für digitale Textverarbeitung sind.

2. Im nächsten Schritt werden Elemente mit gleichen Schlüsseln gruppiert. Nach diesem Schritt liegt also eine Menge aus Schlüssel-Wertmengenpaaren vor (Auch wenn ich immer wieder das Wort Menge verwende, meine ich eigentlich Multimengen, da es relevent ist, wie oft ein bestimmtes Element in der Menge vorkommt [9]). Dieser Schritt ist, anders als die anderen beiden Schritte, bei allen anderen Abfragen gleich. Da in unserem Beispiel das Wort als Schlüssel verwendet wurde, sähe eine beispielhafte Menge nach diesem Schritt wie folgt aus, was stark an eine Strichliste erinnert:

```
1 [
2    "der": {1; 1; 1; 1; 1; 1; 1; 1; 1},
3    "die": {1; 1; 1; 1},
4    ...
```

```
]
```

3. Im dritten und letzten Schritt wird für jeden Schlüssel die sogenannte Reduce-Funktion angewandt. Diese reduziert die Mengen, die den Schlüsseln zugeordnet sind, auf einen Wert. In unserem Beispiel fällt die Reduce-Funktion relativ einfach aus, da sie nur die Elemente der Menge aufsummieren muss:

```
reduce(key, values) {
  return values.reduce((a, b) => a + b);  // "=>" bedeutet so viel wie "wird zu"
}
```

4. Der vierte und letzte Schritt gehört eigentlich nicht mehr zum MapReduce-Verfahren. Dieser wird nach dem MapReduce-Verfahren ausgeführt und dient dazu, die Ergebnisse zu sortieren und eventuell Feinheiten zu verbessern. So ist es zum Beispiel möglich, selten genutzte Worte oder sogenannte Stoppworte in diesem Schritt auszusortieren. Stoppworte sind häufig auftetende Worte, die keine Relevanz für die Erfassung des Dokumenteninhaltes haben. Hierfür verwende ich verschiedene Stoppwortlisten, die von Sprachforschern erstellt worden sind, zusammen mit eigenen Ergänzungen. Der hierzu gehörige Code, der die Wortliste filtert, sieht wie folgt aus:

```
function filter(data) {
  return data.filter(word =>
      stopwords.indexOf(word["_id"].toLowerCase()) < 0)  // word["_id"] ist das eigentliche Wort
}
```

   Des weiteren wird in diesem Schritt versucht, Wörter mit dem gleichen Stamm, und somit mit der gleichen Bedeutung zusammenzuführen, auch wenn diese unterschiedliche Endungen haben. Ein Beispiel hierfür ist, dass zum Beispiel die Wörter “Trump” und “Trumps” zusammengezählt werden. Hierbei wird immer das kürzeste Wort behalten, da dies meist die Grundform ist. Der Algorithmus, den ich hierfür entwickelt habe, ist nicht ganz so einfach:

```
function word_merge(list) {
```

```
2 modList = list.slice();  // kopiere die liste in eine neue
    Unabhängige
3 modList.forEach((nowItem, nowCount, nowObject) => {  // für
    jedes Wort
4     var regex = new RegExp("^" + nowItem["_id"].toLowerCase() +
        '.{0,2}','g');  // baue einen regulären Ausdruck, der
        die Ähnlichkeit abbildet
5     nowObject.forEach((item, index, object) => {  // für jedes
        Wort
6         if(nowItem["_id"].toLowerCase() ==
            item["_id"].toLowerCase()) return;  // vergleiche
            nicht jedes wort mit sich selbst
7         checkWord = item["_id"].toLowerCase();
8         if(checkWord.match(regex)) {  // wende den regulären
            Ausdruck an - sind sich die Wörter ähnlich?
9             nowObject[nowCount]["value"] += item["value"];  //
                führe ähnliche Wörter zusammen
10            object.splice(index, 1);
11        }
12    });
13 });
14 return modList
15 }
```

4.4 Datenbank

Nachdem klar war, wie die Abfragen formuliert werden sollten, habe ich verschiedene Datenbanken in Betracht gezogen. Zuerst habe ich angefangen, mit der Datenbank “MongoDB” [10] zu arbeiten, störte mich aber sehr stark an deren Komplexität. Außerdem bietet MongoDB keine Möglichkeit, diese über das HTTP-Protokoll anzusprechen, was für die Visualisierung allerdings sehr wichtig ist. Also sah ich mich nach anderen Alternativen um und fand “CouchDB” [11], welche vom Apache-Projekt entwickelt wird. Diese Datenbank erfüllte die meisten meiner Anforderungen relativ gut, weshalb ich meinen gesamten, bis dahin existierenden Code an CouchDB anpasste. Nach einiger Zeit des Testens stellte sich allerdings heraus, das CouchDB wahrscheinlich

zu langsam sein würde und ein unpassendes Rechteverwaltungssystem mitbringt, was die Arbeitsersparnis im Gegensatz zu MongoDB zunichte machen würde. In Folge dieser Erkentnis entschied ich mich dazu, meine Software zurück auf MongoDB zu portieren und für diese eine HTTP-Schnittstelle zu entwickeln.

4.5 Middleware

Diese HTTP-Schnittstelle ist gewissermaßen das Bindeglied zwischen der Datenbank und der Visualisierung, weshalb es im nachfolgenden "Middleware" genannt wird. Sie nimmt HTTP-Anfragen von der Visualisierung ("Frontend") entgegen, leitet diese an die Datenbank weiter und schickt die Antwort der Datenbank an das Frontend zurück. Insofern könnte man sie auch als Übersetzer zwischen verschiedenen Protokollen verstehen. Eine Übersetzung ist notwendig, da ich die Visualisierung im Webbrowser implementiert habe, und im Browser nur sehr wenige Schnittstellen verfügbar sind. HTTP ist eine dieser wenigen Möglichkeiten und bietet sich an, da es verhältnismäßig einfach zu implementieren ist [12]. Zusätzlich zu der Übersetzung der Mapreduce-Anfragen ermöglicht es die Middleware, verschiedene Filter zu setzen. Mit Hilfe von diesen Filtern können z.B. der Zeitraum oder die Auswahl der Zeitungen eingegrenzt werden. Diese Filter werden über einen HTTP-Cookie übertragen, der von einer speziellen Einstellungsseite gesetzt wird. Durch diese Funktion ist es möglich, den Ausgangsdatensatz einzugrenzen und so Vergleiche zwischen verschiedenen Zeitspannen oder Zeitungsgruppen anzustellen.

4.6 Visualisierung

Ich habe mich entschieden, die Visualisierung im Browser umzusetzen. Dies lag hauptsächlich daran, dass ich bereits einige der verwendeten Technologien kannte, und somit die Einstiegshürde niedriger war. Außerdem gibt es für den Browser und die damit verbundenen Technologien, wie zum Biespiel der Programmiersprache JavaScript, bereits sehr gute Bibliotheken zur Datenvisualisierung, auf die ich für meine Arbeit zurückgreifen konnte. Ein weiterer, nicht unwichtiger Punkt, ist die einfache Portabilität von Programmen, welche im Browser laufen: Sie können auf fast jedem Computer, unabhängig von

Betriebssystem oder Prozessorarchitektur, innerhalb von Sekunden aufgerufen und ausgeführt werden.

Eines der Hauptziele bei der Konzeption und Entwicklung der gesamten Analysesoftware war die Flexibilität: Es sollte möglichst schnell und einfach möglich sein, verschiedene Analysen über den Datensatz durchzuführen. Dies bezog sich nicht nur auf vorgefertigte, bei der Entwicklung bedachte Analysen, sondern auch auf zukünftige Ansätze. So sollte es einfach möglich sein, eigene neue Analysen über den Datensatz durchzuführen. Dies führte zu der konzeptionellen Endscheidung, dass die Analyse durch den Quellcode des Visualisierungsmoduls vorgegeben sein sollte und alle anderen Zwischenschritte, wie die Middleware oder die Datenbank nur auf die Anfragen des Analysemodul hören sollten. Dies hat zur Folge, das ein großer Teil der Komplexität des Codes im Frontend liegt.

Jede Visualisierung fragt am Anfang die benötigten Daten bei der Middleware über MapReduce-Anfragen an. Die einzelnen Funktionen, die hierzu benötigt werden, werden also in dem Code der Visualisierung definiert und hängen davon ab, welche Daten aus dem Datensatz benötigt werden. Dieser Vorgang kann mehrere Male wiederholt werden, um verschiedene Informationen über verschiedene Aspekte des Datensatzes zu bekommen. Hiernach werden die nun vorliegenden Daten weiterverarbeitet. Sie können also gefiltert, sortiert oder miteinander verrechnet werden. Liegen die Daten nun in der gewünschten Form vor, kann die eigentliche Visualisierung beginnen.

Um die nun aufbereiteten Daten zu visualisieren, gibt es viele verschiedene Möglichkeiten. Ich habe mich dazu entschieden, die Softwarebibliothek "d3.js" [13] zu verwenden, welche es ermöglicht, Verbindungen zwischen Datenmengen und Elementen im Browser herzustellen. Das Aussehen dieser Elemente kann nun mithilfe von gängigen Techniken, wie "CSS" angepasst werden. Für komplexere Visualisierungen werden auch andere Bibliotheken verwendet (siehe [14] und [15]), um z.B. die einzelnen Positionen der Worte zu berechnen. Mehr zu den einzelnen Visualisierungen ist im Abschnitt "Die verschiedenen Visualisierungen" zu finden.

4.7 Zusammenführung

Da die gesamte Software modular aufgebaut ist, besteht sie aus vielen kleinen Komponenten, die spezialisiert und alleine nicht sehr nützlich sind. Diese kleinen Komponenten nennt man in der Softwareentwicklung "Microservices" [16]. Um aus diesen kleinen Microservices eine funktionierende Software zu erhalten, muss man diese jeweils in einer passenden Umgebung starten und richtig miteinander verbinden. Um dies zu bewerkstelligen, habe ich das Software-Paket "Docker" verwendet. Dieses erlaubt es zu allererst, die einzelnen Umgebungen für jeden Microservice zu beschreiben. Diese enthalten zum Beispiel die verschiedenen Programmbibliotheken, die ich in den einzelnen Teilen verwandt habe, oder einen Webserver, um den Visualisierungscode an den Webbrowser auszuliefern. Eine solche Umgebung mit allen darin enthaltenen Programmen nennt man "Container". Diese Container erstellt man immer auf der Basis eines anderen "Basiscontainers". Dies sorgt dafür, dass man sich nicht immer um alles innerhalb eines Containers kümmern muss, also zum Beispiel nicht immer selber das Betriebssystem installieren muss, sondern nur noch die eigenen Komponenten hinzufügen muss. Dies geschieht automatisiert über eine Datei, die jedem Microservice beigefügt ist, dem sogenannten "Dockerfile". Auch ist es möglich, komplett vorgefertigte Container zu verwenden, was ich z.B. für die Datenbank nutze.

Die einzelnen Container, die nun die komplette Umgebung enthalten, die die jeweiligen Microservices zum laufen benötigen, müssen nun nur noch alle richtig verbunden werden. Dies geschieht mit einem Programm namens "Docker-compose". Diesem gibt man eine Datei, die den Zustand des gesamten gewünschten Systems beschreibt. Automatisiert wird aus diesem und allen Dockerfiles der Microservices dann genau das beschriebene System erstellt und gestartet. Die Datei, die dieses System abbildet, sieht wie folgt aus:

```
1 version: '2'
2
3 services:
4   data-collectors:
5      build: data-collectors/
6      links:
7         - mongodb
8   backend:
```

```
9       build: backend/
10      links:
11         - mongodb
12
13   frontend:
14      build: frontend/
15      links:
16         - backend
17      ports:
18         - "80:80"
19
20   mongodb:
21      image: "mongo:latest"
```

Auch hier sieht man wieder sehr schön die Struktur und die Modularität der Software und ihre einzelnen Dienste.

Dieses System, das alle ständig laufenden Komponenten enthält, wird nun auf einem Server gestartet und ermöglicht es, sowohl den Code der Visualisierung als auch Auskünfte über den Datensatz zu bekommen.

5 Die verschiedenen Visualisierungen

Letzten Endes habe ich mich entschlossen, fünf verschiedene Analysen zu implementieren. Diese veranschaulichen verschiedene Informationen über den Datensatz und sind alle sehr unterschiedlich komplex. All diese sind jedoch nicht trivial, da immer versucht werden muss, aus einem Fließtext, also einem Format, mit dem Computer eigentlich nichts anfangen können, Informationen zu gewinnen und zu veranschaulichen. Hierbei ist eine besondere Herausforderung, dass dies alles passieren muss, obwohl der Computer eigentlich über kein Verständnis von “Sinn” verfügt. Aus diesem Grund müssen statistische Verfahren als Hilfsmittel zur Hand genommen werden, die es dem Menschen, der die Daten letzten Endes interpretiert, ermöglichen, Informationen aus der Datenmenge zu ziehen.

Im nachfolgenden sind die verschiedenen Visualisierungen erklärt und es werden beispielhafte Analysen gezeigt.

5.1 Häufigkeitstabelle

Die erste und einfachste Form der Visualisierung, die ich implementiert habe, ist die Häufigkeitstabelle (siehe Abb. 7). Sie gibt an, wie häufig ein bestimmtes Wort im Datensatz verwandt wurde. Sie ist eigentlich noch keine Visualisierung im engeren Sinne, da sie nur aus einer Tabelle besteht, und die Daten nicht ansprechend aufbereitet. Sie ist dennoch sehr praktisch, um sich schnell einen groben Überblick über den Datensatz zu verschaffen. Dies habe ich auch sehr exzessiv bei der Entwicklung von Algorithmen, wie z.B. dem zum Zusammenführen ähnlicher Worte, genutzt. Der Code, der der Wortzählung zu Grunde liegt, ist bereits im Abschnitt “MapReduce” gezeigt und erläutert worden. Auch ist es sehr wichtig, dass Stoppworte herausgefiltert werden, um ein relevantes Ergebnis zu produzieren.

word	count
Jahr	257
La	197
Uhr	151
EU	122
Polizei	110

Abb. 7: Die Darstellung der Worthäufigkeiten als Tabelle

5.2 Wortwolke

Die Grundidee meiner Halbjahresarbeit kam mir, als ich im Spiegel eine Wortwolke entdeckte. Diese Visualisierung war also auch sehr naheliegend und konnte auf den gleichen Datensatz wie die Häufigkeitstabelle zurückgreifen. Dieser wird nun nicht mehr als Tabelle ausgegeben, sondern als Wortwolke, in der öfter vorkommende Wörter größer dargestellt werden. Diese Form der Visualisierung gibt auf intuitive Art und Weise sehr schnell einen groben Überblick über die verwandten Worte und so über die wahrscheinlich gerade relevanten Themen (siehe z.B. Abb. 8).

Abb. 8: Eine Wortwolke vom 20. Januar 2017, dem Tag, an dem Donald Trump ins Amt des US-Präsidenten eingeführt wurde

5.3 Landkarte

Meine nächste Visualisierung sollte nun nicht nur die Häufigkeit verwandter Wörter, sondern auch ihren Zusammenhang darstellen. Ich wollte also, dass Wörter, die öfter zusammen vorkommen, in der Visualisierung näher beieinander stehen. Dies habe ich erreicht, indem ich zuerst, wie bei der Häufigkeitstabelle und der Wortwolke die häufigsten Wörter abgefragt habe. Danach habe ich eine zweite Abfrage geschrieben, die die "Distanz" der häufigsten n Wörter zueinander in den Artikeln feststellt. Diese sieht wie folgt aus:

```
1 function map() {
2     var givenWords = __liste_der_häufigsten_n_wörter__;
3     var words = this.text.replace(/[^A-Za-zÄäÖöÜüß ]/g, "
          ").split(" ").filter(w => w);
4
5     // erstellt eine Liste, die jedes Wort zu den Positionen des
          Vorkommens zuordnet
6     var wordOccurrences = {};
```

```
7     givenWords.forEach(word => {
8         var occurrences = [];
9         for(var i = 0; i < words.length; i++) {
10            if(word == words[i]) {
11                occurrences.push(i);
12            }
13        }
14        wordOccurrences[word] = occurrences;
15    });
16
17    // Iteriere über jede Wortkombination
18    givenWords.forEach(w => {
19        givenWords.forEach(v => {
20            if(w != v && wordOccurrences[w] &&
                  wordOccurrences[v]) {
21                list = [];
22
23                // Iteriere über jede Kombination der Vorkommen
                      jeder Wortkombination
24                wordOccurrences[w].forEach(n => {
25                    wordOccurrences[v].forEach(m => {
26                        // berechne die "Distanz"
27                        list.push(1.0 / Math.pow(Math.abs(m -
                              n), 0.001));
28                    });
29                });
30
31                // Gebe die errechnete "Distanz" zurück
32                strength = list.reduce((a, b) => a + b, 0);
33                if(strength != 0) {
34                    arr = [w, v].sort();
35                    emit(arr[0] + "." + arr[1],
                          JSON.stringify([strength, 1]));
36                }
37            }
38        });
39    });
40 };
```

Hierbei ist zu beachten, dass diese, anders als die anderen Abfragen, nicht statisch ist, sondern von der Liste der häufigsten Wörter als Eingabeparameter abhängt. Eingabeparameter beginnen und enden mit zwei Unterstrichen und werden vor dem Versenden der Anfrage durch die entsprechenden Werte ausgetauscht.

Abb. 9: Die Nachrichtenlandkarte vom 20. Januar 2017, dem Tag, an dem Donald Trump ins Amt des US-Präsidenten eingeführt wurde

Ein einfaches Beispiel für die Anwendung dieser Visualisierung ist die Analyse des gesamten Datensatzes, wie sie in Abb. 9 zu sehen ist. Auf dieser Abbil-

dung sind schon sehr schön einfache Strukturen, wie z.B. die Nähe zwischen den Worten “Trump” und “US” zu erkennen. Interessanterweise scheinen die Worte “Donald” und “Trump” nicht sehr häufig zusammen vorzukommen. Dies könnte z.B. daran liegen, das sie austauschbar verwendet werden können und so nicht immer zusammenstehen.

5.4 Zeitreihe

Natürlich kann man sich in dem gesammelten Datensatz auch ganz andere Parameter angucken und muss nicht immer nur den Zustand zu einem Zeitpunkt betrachten. So ist es zum Beispiel möglich, Aussagen über die Veränderung verschiedener Parameter über die Zeit treffen. Genau dies habe ich nun also in meiner nächsten Visualisierung gemacht.

Zuerst habe ich nur die Anzahl der pro Stunde veröffentlichten Wörter über die Zeit betrachtet (siehe Abb. 10). Hierbei sind sehr deutlich die Arbeitszeiten der Redakteure und die Wochenenden zu erkennen.

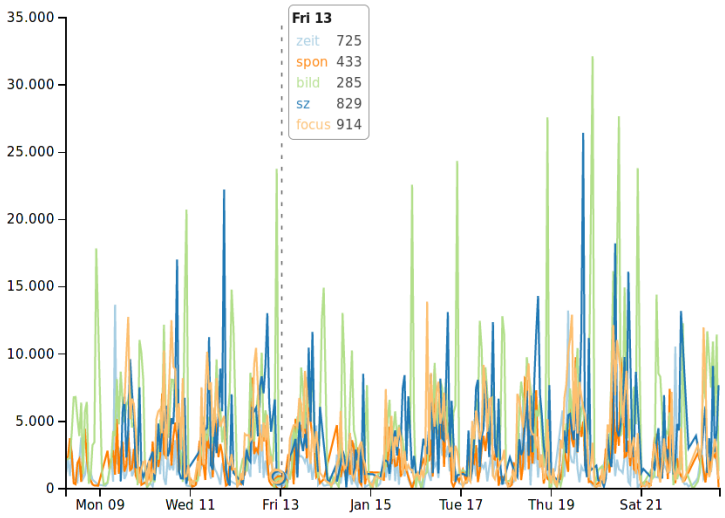

Abb. 10: Auch Redakteure haben Wochenende. Gezeigt ist die Anzahl der geschriebenen Wörter pro Stunde über die Zeit

Als nächstes habe ich mir verschiedene spezifische Begriffe in ihrer Populatität über die Zeit angeschaut. Hierzu habe ich zunächst die Basiszeiteinheit

von einer Stunde auf einen Tag heraufgesetzt, um das Rauschen der Daten zu verringern. D.h. nun werden also alle Artikel von einem Tag zusammengeworfen um einen Datenpunkt zu bilden.

Mithilfe der neuen Analysemöglichkeit habe ich nun die Amtseinführung von Donald Trump betrachtet (Abb. 11). Hierbei ist sehr schön zu sehen, wie die Nennung des Wortes “Trump” am Tage seiner Amtseinführung ein Allzeithoch hat - zumindest bei der Bild-Zeitung. Hieraus können wir schließen, dass die Bild-Online Redaktion sehr flexibel aufgebaut ist und ihre Redakteure weniger stark an ihr Ressort gebunden sind, als die der anderen Zeitungen. Dies sorgt dafür, dass schneller und intensiver auf aktuelle Themen reagiert werden kann, worunter vielleicht aber auch die journalistische Qualität durch weniger fachliche Kompetenz des einzelnen Journalisten leidet.

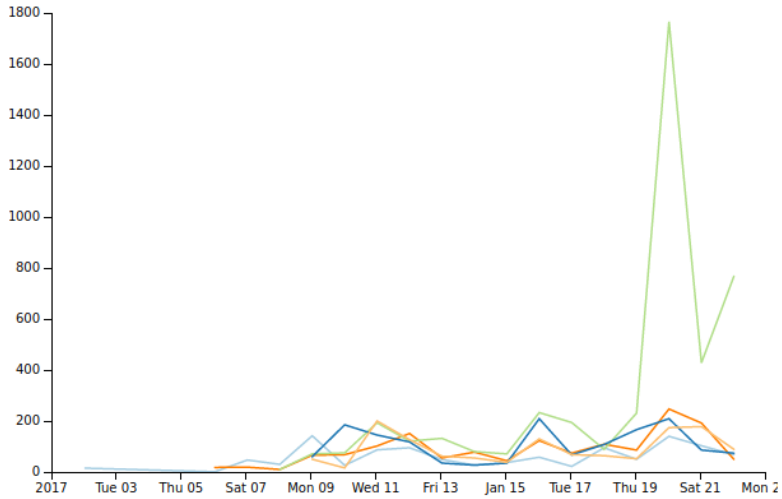

Abb. 11: Der “Peak-Trump”: Anzahl der Nennungen des Wortes “Trump” pro Tag

5.5 Vergleich verschiedener Zeitungen

Als letztes habe ich noch eine relativ einfache Analysemöglichkeit gebaut, die es ermöglicht, verschiedene Zeitungen zu vergleichen. Aus dieser bekommt man die Information, wie oft in einer bestimmten Zeitung ein bestimmtes Wort vorkommt. Hierbei muss man, um eine Vergleichbarkeit gewährleisten

zu können, eine relative Häufigkeit betrachten, da die Veröffentlichungsmengen der verschiedenen Zeitungen sehr unterschiedlich groß sind (Siehe Abb. 10). Um diese relative Häufigkeit zu errechnen, kann man zwei Ansätze verfolgen: 1. Wie viel Prozent der Wörter entsprechen dem gesuchten Wort? 2. Wie viel Prozent der Artikel enthalten das gesuchte Wort?

In Abb. 12 ist zu sehen, welche Zeitung wie viel über das "Dschungelcamp" berichtet. Hierbei ist die zweite Darstellungsweise gewählt worden, da diese sehr viel intuitiver greifbar ist. Das Ergebnis dieser Analyse war für mich sehr unerwartet, besonders, weil die Bild-Zeitung die Rangfolge nicht anführt. Stattdessen ist zu beobachten, dass der Focus mehr Interesse an dem Thema "Dschungelcamp" zeigt, obwohl man ihn in einem intellektuell höheren Millieu verorten würde.

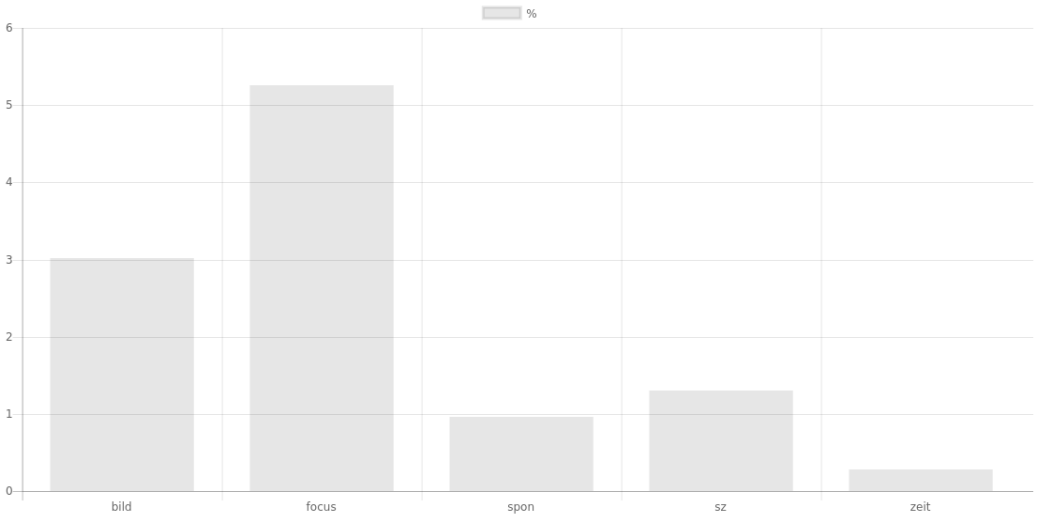

Abb. 12: Welche Zeitung berichtet über das "Dschungelcamp"? Die Grafik zeigt die Anzahl der Artikel verschiedener Zeitungen, die das Wort "Dschungelcamp" enthalten

6 Bedienungsanleitung

Um die Software selbst zu benutzen, gibt es zwei Möglichkeiten:

1. Die erste und einfachere Möglichkeit ist es, die Version, die auf meinem Server läuft, zu verwenden. Aus dieser stammen auch meine Grafiken. Sie kann allerdings teilweise sehr langsam sein (mehrere 10 Minuten Wartezeit), da der verwandte Server sehr leistungsschwach ist. Hierfür muss einfach in einem aktuellen Webbrowser (ich empfehle Firefox) die Adresse `wasistlos.tk` aufgerufen werden. Nun sollte eine Übersicht erscheinen, die alle möglichen Visualisierungen und eine Einstellungsseite mit jeweils einem Icon aufrufbar macht. Behält man die Maus unbewegt mehrere Sekunden über einem Icon, erscheint ein kleiner Erklärungstext zu jeder Visualisierung.
2. Die zweite Möglichleit ist, alle Komponenten auf einem lokalen Computer laufen lassen. Hierfür wird ein Computer mit unixoidem Betriebssystem (z.B. Linux oder macOS), den Softwarepaketen Docker und GNU-Make und einem Internetanschluss benötigt. Um die Software zu starten, muss zunächst der Inhalt des USB-Sticks, welcher sich im Anhang befindet, auf den betreffenden Computer kopiert werden und ein Terminal in dem Ordner, in dem sich nun der Inhalt des Sticks befindet, geöffnet werden. In diesem muss nun der Befehl `make` ausgeführt werden. Danach muss noch `make restore` ausgeführt werden, um den bereits von mir gesammelten, beigefügten Datensatz in die lokale Datenbank zu kopieren. Nun kann wie in Möglichkeit 1 fortgefahren werden, nur dass als Adresse statt `wasistlos.tk` `localhost` verwendet werden muss.

7 Fazit

Insgesamt fand ich meine Halbjahresarbeit sehr interessant. Anfangs ging es mir nur darum, Wortwolken, wie sie im Spiegel zu finden waren, für verschiedene andere Zeitungen zu erstellen. Dies weitete sich dann aber relativ schnell aus und ich begann, mit anderen Möglichkeiten der Auswertung und Visualisierung zu experimentieren. Einen großen Teil der Arbeit, die ich in die Halbjahresarbeit steckte, steckte ich in die Entwicklung dieses Systems, das die verschiedenen Visualisierungen ermöglicht. Für mich war es persönlich eine gute Herausforderung, die Möglichkeit zur Analyse zu entwickeln. Diese ermöglicht es dann jedem selbst, die eigentliche Analyse nach eigenen Parametern durchzuführen und dementsprechend seine eigenen Schlüsse ziehen.

Beispielhaft habe ich auch einige Analysen durcheführt und sie interpretiert, um einen Denkanstoß zu geben und die sehr vielfältigen Möglichkeiten zu demonstrieren.

Mit der Auswertungssoftware bin ich alles in allem sehr zufrieden. Besonders die sehr flexible und modulare Architektur ermöglicht auch Wiederverwendbarkeit der Codes in weiteren Datenanalyseprojekten. Das softwaretechnisch größte Problem, die Geschwindigkeit, ließe sich wahrscheinlich relativ einfach durch ein Austauschen der Datenbank, durch etwas Passenderes, das Zwischenergebnisse speichert, lösen. Insgesamt sind im Rahmen des Projekts ungefähr 1000 Zeilen Code entstanden und ich habe auch auf der technischen Seite vieles Neues gelernt. So habe ich zum Beispiel zum ersten Mal Docker verwendet und NoSQL-Datenbanken verstanden.

Mit meinem eigentlichen Ergebnis, der Analysesoftware, kann nun der interessierte Leser seine eigenen Schlüsse ziehen und die aktuelle Nachrichtenlage aus einer statistischen Metaperspektive betrachten, womit ich mein Ziel erreicht habe.

8 Anhang

8.1 Quellcode

8.1.1 dev.yml

```
version: '2'

services:
  backend:
    build: backend/
    links:
      - mongodb

  frontend:
    build: frontend/
    links:
      - backend
    ports:
      - "80:80"

  mongodb:
    image: "mongo:latest"
    ports:
        - "27027:27017"
```

8.1.2 deploy.yml

```
version: '2'

services:
  data-collectors:
    build: data-collectors/
    links:
      - mongodb
    restart: always

```

```
10   backend:
11     build: backend/
12     links:
13       - mongodb
14
15   frontend:
16     build: frontend/
17     links:
18       - backend
19     ports:
20       - "80:80"
21
22   mongodb:
23     image: "mongo:latest"
24     #ports:
25     #    - "27027:27017"
```

8.1.3 Makefile

```
1  default: dev
2
3  dev:
4      docker-compose -f dev.yml up --build -d
5
6  deploy:
7      DOCKER_HOST=tcp://212.47.231.85:2376 \
8      DOCKER_MACHINE_NAME=deploy \
9      docker-compose -f deploy.yml up --build -d
10
11 clean:
12     docker-compose -f dev.yml down --remove-orphans
13
14 restore: dev
15     mongorestore --port 27027 backup
16
17 count:
18     cloc --exclude-dir=lib --exclude-lang=XML .
```

8.1.4 README.md

```
# Technical setup
All the things are set-up using docker compose. To start the app
    type `docker-compose up` in this directory. For more
    technical information about the architecture, i suggest
    looking at the `docker-compose.yml` file.
```

8.1.5 data-collectors/main.py

```
import os
import time

from pymongo import MongoClient
from pymongo.errors import DuplicateKeyError


def load_modules():
    modules = list(map(lambda path: __import__('modules.' +
        path[:-3], fromlist=['modules']),
                       filter(lambda path: path.endswith(".py"),
                           os.listdir("modules"))))
    for module in modules:
        try:
            module.init()
        except AttributeError:
            pass
    return modules


def push_article(collection, article):
    try:
        collection.insert_one(article)
        print("add %s" % article["_id"])
    except DuplicateKeyError:
        pass  # article is already in db

if __name__ == "__main__":
```

```
27      # initialize the database connection:
28      client = MongoClient(host="mongodb")
29      collection = client.whatsup.news
30
31      # load the modules
32      modules = load_modules()
33      last_updated = {}
34
35      # crawl newspapers
36      while True:
37          for module in modules:
38              print(str(module) + ":")
39              articles, feed_time =
                    module.get_articles(last_updated.get(module, 0))
40              last_updated[module] = feed_time
41
42              # push the articles in the DB
43              for article in articles:
44                  if not article["text"]:
45                      print("fuck: " + article["_id"])
46                      continue
47                  push_article(collection, article)
48          print("crawling finished")
49          time.sleep(60)
```

8.1.6 data-collectors/requirements.txt

```
1 mongo
2 feedparser
3 pyquery
```

8.1.7 data-collectors/Dockerfile

```
1 FROM python:3
2
3 COPY requirements.txt /code/
4 WORKDIR /code
5 RUN pip install -r requirements.txt
```

```
6
7 COPY . /code
8
9 CMD ["python", "-u", "main.py"]
```

8.1.8 data-collectors/README.md

```
1 # Data collectors
2 This directory contains the data collectors. They collect data
    from different news sites and add it to the DB.
```

8.1.9 data-collectors/article_structure.json

```
1 {
2   "title": "required",
3   "subtitle": "optional",
4   "summary": "optional",
5   "text": "optional",
6
7   "meta": {
8     "source": "required",
9     "author": "optional",
10    "tags": "optional",
11    "timestamp": "optional",
12    "url": "optional"
13  }
14 }
```

8.1.10 data-collectors/modules/bild.py

```
1 import feedparser
2 import time
3 from pyquery import PyQuery as pq
4 import re
5
6
7 def init():
8     print("bild module loaded")
```

```
9
10
11 def get_articles(last_updated):
12     articles = []
13     feed =
           feedparser.parse("http://www.bild.de/rssfeeds/vw-alles/vw-alles-26970192,sort
14     feed_time = time.mktime(feed["feed"]["updated_parsed"])
15     if feed_time > last_updated:
16         raw_articles = feed["entries"]
17         for raw_article in raw_articles:
18             if "BILDplus Inhalt" in raw_article["summary"]:  #
                   fuck bild plus
19                 continue
20             page = pq(url=raw_article["link"])
21             page("em").remove()
22             page("style").remove()
23             page("script").remove()
24
25             article = {
26                 "title": raw_article["title"],
27                 "summary": re.sub("<[\s\S]*>", "",
                       raw_article["summary"]),
28                 "text": page(".txt").text(),
29                 "raw": page.html(),
30
31                 "meta": {
32                     "source": "bild",
33                     "tags": list(map(lambda tag: tag["term"],
                           raw_article["tags"])),
34                     "timestamp":
                           time.mktime(raw_article["published_parsed"]),
35                     "url": raw_article["link"]
36                 },
37                 "_id": raw_article["link"]
38             }
39             if time.time() > article["meta"]["timestamp"] +
                   60*60*48:
40                 continue
```

```
41              articles.append(article)
42
43      return articles, feed_time
```

8.1.11 data-collectors/modules/spon.py

```
1  import feedparser
2  import time
3  from pyquery import PyQuery as pq
4  import re
5
6
7  def init():
8      print("spon module loaded")
9
10
11 def get_articles(last_updated):
12     articles = []
13     feed =
           feedparser.parse("http://www.spiegel.de/schlagzeilen/index.rss")
14     feed_time = time.mktime(feed["feed"]["updated_parsed"])
15     if feed_time > last_updated:
16         raw_articles = feed["entries"]
17         for raw_article in raw_articles:
18             page = pq(url=raw_article["link"])
19             if page(".obfuscated").text():  # fuck spiegel plus
20                 continue
21             article = {
22                 "title": raw_article["title"],
23                 "summary": re.sub("<[\s\S]*>", "",
                       raw_article["summary"]),
24                 "text": page(".article-section")("p").text(),
25                 "raw": page.html(),
26
27                 "meta": {
28                     "source": "spon",
29                     "tags": list(map(lambda tag: tag["term"],
                           raw_article["tags"])),
```

```
30                      "timestamp":
                           time.mktime(raw_article["published_parsed"]),
31                      "url": raw_article["link"]
32                  },
33                  "_id": raw_article["link"]
34              }
35              articles.append(article)
36
37      return articles, feed_time
```

8.1.12 data-collectors/modules/zeit.py

```
1  import feedparser
2  import time
3  from pyquery import PyQuery as pq
4  import re
5
6
7  def init():
8      print("zeit module loaded")
9
10
11 def get_articles(last_updated):
12     articles = []
13     feed = feedparser.parse("http://newsfeed.zeit.de/index")
14     feed_time = time.mktime(feed["updated_parsed"])
15     if feed_time > last_updated:
16         raw_articles = feed["entries"]
17         for raw_article in raw_articles:
18             text = ""
19             cnt = 0
20             while not text and cnt < 100:  # zeit.de sucks hard
21                 page = pq(url=raw_article["link"])
22                 text = page(".article-page")("p").text()
23                 cnt += 1
24             if not text:
25                 print("fuck: " + raw_article["link"])
26                 continue
```

```
27              article = {
28                  "title": raw_article["title"],
29                  "summary": re.sub("<[\s\S]*>", "",
                        raw_article["summary"]),
30                  "text": text,
31                  "raw": page.html(),
32
33                  "meta": {
34                      "source": "zeit",
35                      "author": re.sub("ZEIT ONLINE: \w* - ", "",
                            raw_article["author"]).split(", "),
36                      "tags": list(map(lambda tag: tag["term"],
                            raw_article["tags"])),
37                      "timestamp":
                            time.mktime(raw_article["published_parsed"]),
38                      "url": raw_article["link"]
39                  },
40                  "_id": raw_article["link"]
41              }
42              articles.append(article)
43      return articles, feed_time
```

8.1.13 data-collectors/modules/focus.py

```
1  import feedparser
2  import time
3  from pyquery import PyQuery as pq
4  import re
5
6
7  def init():
8      print("focus module loaded")
9
10
11 def get_articles(last_updated):
12     articles = []
13     feed =
           feedparser.parse("http://rss.focus.de/fol/XML/rss_folnews.xml")
```

```
14      feed_time = time.mktime(feed["feed"]["updated_parsed"])
15      if feed_time > last_updated:
16          raw_articles = feed["entries"]
17          for raw_article in raw_articles:
18              page = pq(url=raw_article["link"])
19              page("style").remove()
20              page("script").remove()
21
22              article = {
23                  "title": raw_article["title"],
24                  "summary": re.sub("<[\s\S]*>", "",
                        raw_article["summary"]),
25                  "text": page(".textBlock").text(),
26                  "raw": page.html(),
27
28                  "meta": {
29                      "source": "focus",
30                      "tags": list(map(lambda tag: tag["term"],
                            raw_article["tags"])),
31                      "timestamp":
                            time.mktime(raw_article["published_parsed"]),
32                      "url": raw_article["link"]
33                  },
34                  "_id": raw_article["link"]
35              }
36              articles.append(article)
37
38      return articles, feed_time
```

8.1.14 data-collectors/modules/sz.py

```
1 import feedparser
2 import time
3 from pyquery import PyQuery as pq
4 import re
5
6
7 def init():
```

```
8     print("sz module loaded")
9
10
11 def get_articles(last_updated):
12     articles = []
13     feed =
          feedparser.parse("http://rss.sueddeutsche.de/app/service/rss/alles/index.rss?
14     feed_time = time.mktime(feed["feed"]["updated_parsed"])
15     if feed_time > last_updated:
16         raw_articles = feed["entries"]
17         for raw_article in raw_articles:
18             page = pq(url=raw_article["link"])
19             page("style").remove()
20             page("script").remove()
21             page(".authorProfileContainer").remove()
22             page(".ad").remove()
23             page(".feedbackClick").remove()
24
25             article = {
26                 "title": raw_article["title"],
27                 "summary": re.sub("<[\s\S]*>", "",
                      raw_article["summary"]),
28                 "text": page("#article-body").text(),
29                 "raw": page.html(),
30
31                 "meta": {
32                     "source": "sz",
33                     "tags": list(map(lambda tag: tag["term"],
                          raw_article["tags"])),
34                     "timestamp":
                          time.mktime(raw_article["published_parsed"]),
35                     "url": raw_article["link"]
36                 },
37                 "_id": raw_article["link"]
38             }
39             articles.append(article)
40
41     return articles, feed_time
```

8.1.15 backend/requirements.txt

```
flask
mongo
```

8.1.16 backend/Dockerfile

```
FROM python:3

COPY requirements.txt /code/
WORKDIR /code
RUN pip install -r requirements.txt

COPY . /code

CMD ["python", "-u", "mongo_http.py"]
```

8.1.17 backend/mongo_http.py

```
from pymongo import MongoClient
from bson.code import Code
from flask import Flask, request, jsonify, abort

import json
import base64

# initialize the database connection
client = MongoClient(host="mongodb")
collection = client.whatsup.news

# init the flask server
app = Flask(__name__)


@app.route('/api')
def wrap():
    map_function = request.args.get('map')
    reduce_function = request.args.get('reduce')
```

```
20      if "query" in request.args:
21          query = request.args.get('query')
22      else:
23          query = request.cookies.get('query')
24
25      if reduce_function is None or map_function is None:
26          return abort(400)
27
28      map_function = str(map_function)
29      reduce_function = str(reduce_function)
30
31      if query and str(query) != "":
32          query =
               json.loads(base64.b64decode(str(query)).decode("utf-8"))
33      else:
34          query = {}
35
36      result = collection.map_reduce(Code(map_function),
           Code(reduce_function), out={"inline": 1}, query=query)
37
38      return jsonify(result["results"])
39
40
41  # finally run the flask server
42  if __name__ == "__main__":
43      app.run("0.0.0.0", 3141, debug=True)
```

8.1.18 backend/README.md

```
1  # MongoHttp
2  The mongoDB HTTP wrapper is writen in python. It allows us to
       make unauthenticated
3  map/reduce requests to a Mongodb via HTTP, which is not possible
       by default. . It exposes a single simple
4  HTTP GET endpoint:
```

GET /?map=[mapfunction]&reduce=[reducefunction]

```
This endpoint returns the result of the map/reduce request as
    given by the DB.

## Example
```

map: function(doc) {

}

reduce: function() {

}

8.1.19 frontend/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Was ist los?</title>

    <style>
        body, html {
            width: 100%;
            height: 100%;

            box-sizing: border-box;

            display: flex;
            flex-direction: row;
            flex-wrap: wrap;
        }
        a {
            box-sizing: border-box;
            flex-grow: 1;
            background-size: cover;
            display: inline-block;
            position: relative;

```

```
25              min-width: 500px;
26              min-height: 500px;
27          }
28
29          a:hover {
30              box-shadow: 0 0 30px #0087AF;
31              z-index: 10;
32          }
33      </style>
34  </head>
35  <body>
36      <a href="visuals/table.html" style="background-image:
            url(img/table.png)"></a>
37      <a href="visuals/cloud.html" style="background-image:
            url(img/cloud.png)"></a>
38      <a href="visuals/force.html" style="background-image:
            url(img/force.png)"></a>
39
40      <a href="visuals/time.html?q=wort" style="background-image:
            url(img/time.png)"></a>
41      <a href="visuals/time_d.html?q=wort"
            style="background-image: url(img/time.png)"></a>
42
43      <a href="visuals/contains.html?q=wort"
            style="background-image: url(img/contains.png)"></a>
44      <a href="visuals/match.html?q=wort" style="background-image:
            url(img/contains.png)"></a>
45
46      <a href="settings.html" style="background-image:
            url(img/settings.svg)"></a>
47  </body>
48  </html>
```

8.1.20 frontend/settings.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
```

```
4       <meta charset="UTF-8">
5       <title>Was ist los?</title>
6
7       <script src="js/mongo_query.js"></script>
8       <script>
9
10          // fill the form
11          function map() {
12              emit(this.meta.source, 0)
13          }
14
15          function render(res) {
16              var newspapers = res.map(e => e["_id"]);
17
18              var checkboxHtml = newspapers.map(e => '<label
                    for="' + e + '">' + e + '</label><input
                    type="checkbox" id="' + e + '"><br>');
19              document.getElementById("newspapers").innerHTML =
                    checkboxHtml.reduce((a, b) => a + b);
20
21              Array.from(document.getElementsByTagName("input")).forEach(el
                    => el.onchange = () => {
22                  var from = document.getElementById("from").value;
23                  var to = document.getElementById("to").value;
24
25                  from = +new Date(from) / 1000;
26                  to = +new Date(to) / 1000;
27
28                  var sources = newspapers.filter(elem =>
                        document.getElementById(elem).checked);
29
30                  var query = {
31                      "meta.timestamp": {"$gte": from},
32                      "meta.timestamp": {"$lte": to},
33
34                      "meta.source": {"$in": sources}
35                  };
36
```

```
37                 if (!(sources && from && to)) return;
38
39                 var base64 = btoa(JSON.stringify(query));
40                 console.log(base64);
41                 console.log(query);
42
43                 document.cookie = "query=" + base64 + "; path=/";
44
45
46                 alert("saved!");
47             })
48         }
49
50         mongo_query(map, count, render, false);
51     </script>
52 </head>
53 <body>
54     <form>
55         <p>
56             <label for="from">from:</label><input
                   type="datetime-local" id="from"></br>
57             <label for="to">to:</label><input
                   type="datetime-local" id="to">
58         </p>
59         <p id="newspapers">
60             <!-- this is autogenerated -->
61         </p>
62     </form>
63 </body>
64 </html>
```

8.1.21 frontend/nginx.conf

```
1 ##################################################################
2 # this config file will be placed inside the docker container #
3 ##################################################################
4
5 server {
```

```
    listen       80;
    server_name  localhost;

    # serve static files
    location / {
        root   /usr/share/nginx/html;
        index  index.html;

        # do nothing it the extension is already present
        if ($request_filename ~* ^.+.html$) {
          break;
        }

        # add .html if it was not present
        if (-e $request_filename.html) {
          rewrite ^/(.*)$ /$1.html permanent;
          break;
        }
    }

    # pass requests for dynamic content to the mongo_http API
    location /api {
      add_header Access-Control-Allow-Origin *;
      proxy_pass      http://backend:3141;
      proxy_read_timeout 60m;
    }

    # error pages
    error_page  404 /404.html;
    error_page   500 502 503 504  /50x.html;

    # i do wanted GET for everything...
    large_client_header_buffers 4 8M;
}
```

8.1.22 frontend/Dockerfile

```
FROM nginx:latest
```

```
COPY nginx.conf etc/nginx/conf.d/default.conf
COPY . /usr/share/nginx/html
```

8.1.23 frontend/js/word_merge.js

```
/**
 * Created by jaro on 22.12.16.
 */

function word_merge(list) {
    modList = list.slice();
    modList.forEach((nowItem, nowCount, nowObject) => {
        var regex = new RegExp("^" + nowItem["_id"] +
            '.{0,2}','g');
        nowObject.forEach((item, index, object) => {
            if(nowItem["_id"] == item["_id"]) return;
            if(nowItem["_id"].length < 4) return;
            checkWord = item["_id"];
            if(checkWord.match(regex)) {
                nowObject[nowCount]["value"] += item["value"];
                object.splice(index, 1);
            }
        });
    });
    return modList
}
```

8.1.24 frontend/js/mongo_query.js

```
/**
 * Created by jaro on 13.11.16.
 */
var host = document.location.host.indexOf(":") != -1 ?
   "http://localhost" : "";

function mongo_query(map, reduce, callback, query) {
    var request = new XMLHttpRequest();
    request.onreadystatechange = function () {
```

```
9           if (this.readyState == 4 && this.status == 200) {
10              callback(JSON.parse(this.responseText));
11          }
12      };
13
14      var requestUrl = host + "/api";
15      requestUrl += "?map=" + encodeURIComponent(map.toString());
16      requestUrl += "&reduce=" +
            encodeURIComponent(reduce.toString());
17      if(query && (typeof query === 'string' || query instanceof
            String)) {
18          requestUrl += "&query=" +
                encodeURIComponent(query.toString());
19      } else if(typeof(query) === "boolean"){
20          requestUrl += "&query";
21      }
22
23      request.open("GET", requestUrl, true);
24      request.send();
25  }
26
27  function example_map() {
28      emit("count", 1);
29  }
30
31  function example_reduce(key, values) {
32      return values.reduce((previousValue, currentValue) =>
            currentValue + previousValue);
33  }
34  var count = example_reduce;
```

8.1.25 frontend/visuals/cloud.html

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <script src="../lib/d3.v3.js"></script>
```

```
6     <script src="../lib/d3.layout.cloud.js"></script>
7
8     <script src="../js/mongo_query.js"></script>
9     <script src="../lib/stopwords.js"></script>
10    <script src="../js/word_merge.js"></script>
11
12    <script src="../params/cloud/first.js"></script>
13    <script>
14        window.onload = () => {
15            function map() {
16                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                      ]/g, " ").split(" ");
17                for (var i = 0; i < words.length; i++) {
18                    var word = words[i];
19                    if(word) {
20                        emit(words[i], 1);
21                    }
22                }
23            }
24
25            mongo_query(map, count, render);
26        };
27
28        var render = (wordList) => {
29            wordList = word_merge(wordList
30                    .filter(word =>
                          stopwords.indexOf(word["_id"].toLowerCase())
                          < 0)
31                    .filter(word =>
                          german_stopwords_case.indexOf(word["_id"])
                          < 0)
32                    .sort((a,b) => b["value"] - a["value"])
33                    .slice(0, num_words)
34            ).map(elem => {
35                return {text: elem["_id"], size: elem["value"] /
                      wordList[0]["value"] * text_multiply};
36            }).sort((a,b) => b["value"] - a["value"]);
37            console.log(wordList);
```

```
38              var layout = d3.layout.cloud()
39                      .size([window.innerWidth,
                           window.innerHeight])
40                      .words(wordList)
41                      .padding(padding)
42                      .rotate(() => 0)
43                      .font("Serif")
44                      .fontSize(d => d.size)
45                      .on("end", draw);
46
47              layout.start();
48
49              function draw(words) {
50                  d3.select("body").append("svg")
51                          .attr("width", layout.size()[0])
52                          .attr("height", layout.size()[1])
53                          .append("g")
54                          .attr("transform", "translate(" +
                               layout.size()[0] / 2 + "," +
                               layout.size()[1] / 2 + ")")
55                          .selectAll("text")
56                          .data(words)
57                          .enter().append("text")
58                          .style("font-size", d => d.size + "px")
59                          .style("font-family", "Serif")
60                          .attr("text-anchor", "middle")
61                          .attr("transform", d => "translate(" +
                               [d.x, d.y] + ")rotate(" + d.rotate +
                               ")")
62                          .text(d =>  d.text);
63              }
64
65          };
66      </script>
67  </head>
68  <body>
69
70  </body>
```

```
</html>
```

8.1.26 frontend/visuals/force.html

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">

    <title>Visualizer | Word Table</title>

    <style type="text/css" media="screen">
        html, body, svg {
            width: 100%;
            height: 100%;
            margin: 0;
            padding: 0;
        }

        text {
            text-anchor: middle;
            font-family: Serif;
        }
    </style>
    <script src="../js/mongo_query.js"></script>
    <script src="../lib/stopwords.js"></script>
    <script src="../js/word_merge.js"></script>
    <script type="text/javascript"
       src="../lib/vivagraph.min.js"></script>

    <script src="../params/force/first.js"></script>
    <script>
        words = [];
        links = [];

        function zeroStage() {
            var map = function () {
```

```
34                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                      ]/g, " ").split(" ");
35                for (var i = 0; i < words.length; i++) {
36                    var word = words[i];
37                    if(word) {
38                        emit(words[i], 1);
39                    }
40                }
41            };
42            mongo_query(map, count, firstStage);
43        }
44        function firstStage(inWords) {
45            words = word_merge(
46                inWords
47                    .filter(word =>
                          stopwords.indexOf(word["_id"].toLowerCase())
                          < 0)
48                    .filter(word =>
                          german_stopwords_case.indexOf(word["_id"])
                          < 0)
49                    .filter(word => word["_id"][0].toUpperCase()
                          == word["_id"][0])
50                    .sort((a,b) => b["value"] - a["value"])
51                    .slice(0, num_nodes)
52            ).filter(word =>
                  stopwords.indexOf(word["_id"].toLowerCase()) < 0)
53            .filter(word =>
                  german_stopwords_case.indexOf(word["_id"]) < 0)
54            .sort((a,b) => b["value"] - a["value"]);
55
56            var map = function() {
57                var givenWords = __marker__;
58                var distance_function = __marker2__;
59                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                      ]/g, " ").split(" ").filter(w => w);
60
61                // create dict with {"word": ['o', 'c', 'c',
                      'u', 'r', 'e', 'n', 'c', 'e']}
```

```
62                  var wordOccurrences = {};
63                  givenWords.forEach(word => {
64                      var occurrences = [];
65                      for(var i = 0; i < words.length; i++) {
66                          if(word == words[i]) {
67                              occurrences.push(i);
68                          }
69                      }
70                      wordOccurrences[word] = occurrences;
71                  });
72
73                  givenWords.forEach(w => {
74                      givenWords.forEach(v => { // every word
                            combination is covered as w and v
75                          if(w != v && wordOccurrences[w] &&
                                wordOccurrences[v]) {
76                              list = [];
77                              wordOccurrences[w].forEach(n => {
78                                  wordOccurrences[v].forEach(m => {
79                                      list.push(distance_function(n,
                                            m));
80                                  });
81                              });
82
83                              strength = list.reduce((a, b) => a +
                                    b, 0);
84                              if(strength != 0) {
85                                  arr = [w, v].sort();
86                                  emit(arr[0] + "." + arr[1],
                                        JSON.stringify([strength,
                                        1]));
87                              }
88                          }
89                      });
90                  });
91              };
92
93              var reduce = function (key, values) {
```

```
94                  var vals = values.map(val => JSON.parse(val));
95                  var newVals = vals[0].map((e, i) => e +
                        vals[1][i]);
96                  return JSON.stringify(newVals);
97              };
98
99              map = map.toString().replace("__marker__",
                    JSON.stringify(words.map(obj => obj["_id"])));
100             map = map.replace("__marker2__",
                    distance_function.toString());
101             mongo_query(map, reduce, secondStage);
102         }
103         function secondStage(links_param) {
104             links = links_param.map(link => {
105                 link["value"] = JSON.parse(link["value"]);
106                 return link;
107             }).map(link => {
108                 link["value"] = link["value"][0];
109                 return link;
110             });
111             render();
112         }
113         function render() {
114             var graph = Viva.Graph.graph();
115
116             words.forEach(obj => {
117                 graph.addNode(obj["_id"], max_node_size *
                        obj["value"] / Math.max.apply(Math,
                        Object.keys(words).map(a =>
                        words[a]["value"])));
118             });
119
120             // one link per node
121             lnks = {};
122             links.forEach(link => {
123                 var words = link["_id"].split(".");
124                 if(!lnks[words[0]]) {
125                     lnks[words[0]] = Object.keys(Object.keys)
```

```
126                 }
127                 lnks[words[0]].push([words[1], link["value"]]);
128             });
129             Object.keys(lnks).forEach(key =>
130                 lnks[key] = lnks[key].sort((a, b) => a[1] -
                        b[1]).splice(0, num_min_links)
131             );
132             Object.keys(lnks).forEach(key => {
133                 start = key;
134                 lnks[key].forEach(end => {
135                     graph.addLink(start, end[0], end[1])
136                 });
137             });
138
139             // top x nodes
140             links.sort((l, m) => {
141                 return l["value"] - m["value"];
142             }).splice(0, num_top_links).forEach(link => {
143                 var words = link["_id"].split(".");
144                 graph.addLink(words[0], words[1], link["value"])
145             });
146
147             var graphics = Viva.Graph.View.svgGraphics();
148             graphics.node(function(node) {
149                 return Viva.Graph.svg('text')
150                         .attr("font-size", node.data)
151                         .attr("dy", ".25em")
152                         .text(node.id);
153             });
154
155             graphics.link(function(link){
156                 return Viva.Graph.svg('path')
157                         .attr('stroke', '#eee')
158                         .attr('stroke-dasharray', '5, 5');
159             }).placeLink(function(linkUI, fromPos, toPos) {
160                 // linkUI - is the object returend from link()
                        callback above.
161                 var data = 'M' + fromPos.x + ',' + fromPos.y +
```

```
162                         'L' + toPos.x + ',' + toPos.y;
163                 // 'Path data'
                        (http://www.w3.org/TR/SVG/paths.html#DAttribute
                        )
164                 // is a common way of rendering paths in SVG:
165                 linkUI.attr("d", data);
166             });
167
168
169
170             var layout = Viva.Graph.Layout.forceDirected(graph,
                    layout_params);
171
172             var renderer = Viva.Graph.View.renderer(graph, {
173                 graphics : graphics,
174                 layout : layout
175             });
176             renderer.run();
177         }
178
179         window.onload = zeroStage;
180     </script>
181 </head>
182
183 <body>
184 </body>
185
186 </html>
```

8.1.27 frontend/visuals/match.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5
6     <script src="../js/mongo_query.js"></script>
7
```

```
8     <script src="../lib/Chart.bundle.min.js"
          integrity="sha256-RASNMNlmRtIreeznffYMDUxBXcMRjijEaaGF/gxT6vw="
          crossorigin="anonymous"></script>
9     <script>
10        newspaperWords = [];
11        actualWords = [];
12
13        var stage1 = () => {
14            function map() {
15                this.text
16                        .replace(/[^A-Za-zÄäÖöÜüß ]/g, " ")
17                        .split(" ")
18                        .filter(w => w)
19                        .forEach(word => emit(this.meta.source,
                              1));
20            }
21
22            mongo_query(map, count, res => newspaperWords = res);
23            stage2();
24        };
25
26        var stage2 = () => {
27            function map() {
28                this.text
29                        .replace(/[^A-Za-zÄäÖöÜüß ]/g, " ")
30                        .split(" ")
31                        .filter(w => w)
32                        .filter(w => /__marker__/i.test(w))
33                        .forEach(word => emit(this.meta.source,
                              1));
34            }
35
36            map = map.toString().replace("__marker__",
                  decodeURIComponent(location.href.split("?q=")[1]));
37
38            mongo_query(map, count, render);
39        };
40
```

```
41
42
43          var render = (res) => {
44              actualWords = res;
45
46              var data = {
47                  labels: newspaperWords.map(e => e["_id"]),
48                  datasets: [
49                      {
50                          label: "%",
51                          data: newspaperWords.map((e, i) =>
                                actualWords[i].value / e.value *
                                100),
52                      }
53                  ]
54              };
55
56              var ctx = "myChart";
57
58              var myBarChart = new Chart(ctx, {
59                  type: 'bar',
60                  data: data,
61                  options: {
62                      scales: {
63                          yAxes: [{
64                              ticks: {
65                                  beginAtZero:true
66                              }
67                          }]
68                      }
69                  }
70              });
71          };
72          window.onload = stage1;
73      </script>
74  </head>
75  <body>
76      <canvas id="myChart"></canvas>
```

```
</body>
</html>
```

8.1.28 frontend/visuals/contains.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">

    <script src="../js/mongo_query.js"></script>

    <script src="../lib/Chart.bundle.min.js"
       integrity="sha256-RASNMNlmRtIreeznffYMDUxBXcMRjijEaaGF/gxT6vw="
       crossorigin="anonymous"></script>
    <script>
        newspaperWords = [];
        actualWords = [];

        var stage1 = () => {
            function map() {
                        emit(this.meta.source, 1);
            }

            mongo_query(map, count, res => newspaperWords = res);
            stage2();
        };

        var stage2 = () => {
            function map() {
                if(/__marker__/i.test(this.text)) {
                    emit(this.meta.source, 1);
                }
            }

            map = map.toString().replace("__marker__",
               decodeURIComponent(location.href.split("?q=")[1]));

```

```
31              mongo_query(map, count, render);
32          };
33
34
35
36          var render = (res) => {
37              actualWords = res;
38
39              var data = {
40                  labels: newspaperWords.map(e => e["_id"]),
41                  datasets: [
42                      {
43                          label: "%",
44                          data: newspaperWords.map((e, i) => {
45                              try {
46                                  return actualWords[i].value /
                                        e.value * 100;
47                              } catch(e) {
48                                  return 0;
49                              }
50                          }),
51                      }
52                  ]
53              };
54
55              var ctx = "myChart";
56
57              var myBarChart = new Chart(ctx, {
58                  type: 'bar',
59                  data: data,
60                  options: {
61                      scales: {
62                          yAxes: [{
63                              ticks: {
64                                  beginAtZero:true
65                              }
66                          }]
67                      }
```

```
                }
            });
        };
        window.onload = stage1;
    </script>
</head>
<body>
    <canvas id="myChart"></canvas>
</body>
</html>
```

8.1.29 frontend/visuals/time.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <script src="../lib/d3.v3.js"></script>
    <script src="../lib/d3_timeseries.js"></script>
    <link rel="stylesheet" href="../lib/d3_timeseries.css">

    <script src="../js/mongo_query.js"></script>
    <script>
        data = {};

        window.onload = () => {
            function map() {
                if(this.meta.timestamp < 1483228800) {
                    return;
                }
                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                    ]/g, " ").split(" ");
                words.forEach(word => {
                    if(/__marker__/i.test(word)) {
                        emit(new Date(this.meta.timestamp *
                            1000).toISOString().slice(0, 13) +
                            "." + this.meta.source, 1);
                    }
```

```
23                  });
24              }
25              map = map.toString().replace("__marker__",
                    decodeURIComponent(location.href.split("?q=")[1]));
26              mongo_query(map, count, render);
27          };
28
29          window.addEventListener("resize", render);
30          var render = (wordList) => {
31              data = {};
32              wordList.map(elem => {
33                  var source = data[elem["_id"].split(".")[1]];
34                  if(!source) {
35                      source = [];
36                  }
37                  source.push({date: new
                        Date(elem["_id"].split(".")[0] + ":00"), n:
                        elem["value"]});
38                  data[elem["_id"].split(".")[1]] = source;
39              });
40              console.log(data);
41
42              var chart = d3.timeseries()
43                      .margin.left(90)
44                      .yscale.domain([0])
45
46              Object.keys(data).forEach(key => {
47                  chart.addSerie(data[key], {x:'date',y:'n'},
                        {interpolate:'linear', label: key})
48              });
49              chart('#chart')
50
51          };
52      </script>
53  </head>
54  <body>
55      <div id="chart"></div>
56  </body>
```

```
</html>
```

8.1.30 frontend/visuals/table.html

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">

    <title>Visualizer | Word Table</title>

    <script src="../js/mongo_query.js"></script>
    <script src="../lib/stopwords.js"></script>
    <script src="../js/word_merge.js"></script>

</head>

<body>
    <main id="out">
        <!--the actual content goes here -->
        <script>
            function map() {
                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                   ]/g, " ").split(" ");
                for (var i = 0; i < words.length; i++) {
                    var word = words[i];
                    if(word) {
                      emit(words[i], 1);
                    }
                }
            }

            function render(obj) {
                obj = word_merge(obj
                                 .filter(word =>
                                    stopwords.indexOf(word["_id"].toLowerCase())
                                    < 0)
```

```
32                                  .filter(word =>
                                       german_stopwords_case.indexOf(word["_id"])
                                       < 0)
33                                  .sort((a,b) => b["value"] -
                                       a["value"])
34                                  .slice(0, 500)
35              ).sort((a,b) => b["value"] - a["value"]);
36
37              var table = "<table style='margin: auto;'>";
38              table += "<tr><th>word</th><th>count</th></tr>";
39              obj.forEach(elm => {
40                  table += "<tr><td>" + elm["_id"] +
                        "</td><td>" + elm["value"] +
                        "</td></tr>";
41              });
42              table += "</table>";
43
44              document.getElementById("out").innerHTML = table;
45
46              data = obj;
47          }
48
49          var data = [];
50          mongo_query(map, count, render);
51      </script>
52
53      <!-- first place a loading animation here -->
54      <img src="../img/loading.svg" alt="loading..."
           class="loading"/>
55  </main>
56 </body>
57
58 </html>
```

8.1.31 frontend/visuals/time_d.html

```
1 <!DOCTYPE html>
2 <html>
```

```
<head>
    <meta charset="utf-8">
    <script src="../lib/d3.v3.js"></script>
    <script src="../lib/d3_timeseries.js"></script>
    <link rel="stylesheet" href="../lib/d3_timeseries.css">

    <script src="../js/mongo_query.js"></script>
    <script>
        data = {};

        window.onload = () => {
            function map() {
                if(this.meta.timestamp < 1483228800) {
                    return;
                }
                var words = this.text.replace(/[^A-Za-zÄäÖöÜüß
                   ]/g, " ").split(" ");
                words.forEach(word => {
                    if(/__marker__/i.test(word)) {
                        emit(new Date(this.meta.timestamp *
                           1000).toISOString().slice(0, 10) +
                           "." + this.meta.source, 1);
                    }
                });
            }
            map = map.toString().replace("__marker__",
               decodeURIComponent(location.href.split("?q=")[1]));
            mongo_query(map, count, render);
        };

        window.addEventListener("resize", render);
        var render = (wordList) => {
            data = {};
            wordList.map(elem => {
                var source = data[elem["_id"].split(".")[1]];
                if(!source) {
                    source = [];
                }
```

```
37                  source.push({date: new
                        Date(elem["_id"].split(".")[0]), n:
                        elem["value"]});
38                  data[elem["_id"].split(".")[1]] = source;
39              });
40              console.log(data);
41
42              var chart = d3.timeseries()
43                      .margin.left(90)
44                      .yscale.domain([0])
45
46              Object.keys(data).forEach(key => {
47                  chart.addSerie(data[key], {x:'date',y:'n'},
                        {interpolate:'linear', label: key})
48              });
49              chart('#chart')
50
51          };
52      </script>
53  </head>
54  <body>
55      <div id="chart"></div>
56  </body>
57  </html>
```

8.1.32 frontend/params/cloud/first.js

```
1  /**
2   * Created by jaro on 08.01.17.
3   */
4
5  var num_words = 150;
6  var text_multiply = 20;
7  var padding = 2;
```

8.1.33 frontend/params/force/first.js

```
1  /**
```

```
2  * Created by jaro on 07.01.17.
3  */
4
5 var num_nodes = 200;
6 var max_node_size = 50;
7 var distance_function = (a, b) => 1.0 / Math.pow(Math.abs(a -
      b), 0.001);
8 var num_min_links = 2;
9 var num_top_links = num_nodes;
10 var layout_params = {
11     /**
12      * Ideal length for links (springs in physical model).
13      */
14     springLength: 35,
15
16     /**
17      * Hook's law coefficient. 1 - solid spring.
18      */
19     springCoeff: 0.0008,
20
21     /**
22      * Coulomb's law coefficient. It's used to repel nodes thus
            should be negative
23      * if you make it positive nodes start attract each other :).
24      */
25     gravity: -2.5,
26
27     /**
28      * Theta coefficient from Barnes Hut simulation. Ranged
            between (0, 1).
29      * The closer it's to 1 the more nodes algorithm will have
            to go through.
30      * Setting it to one makes Barnes Hut simulation no
            different from
31      * brute-force forces calculation (each node is considered).
32      */
33     theta: 1,
34
```

```
35      /**
36       * Drag force coefficient. Used to slow down system, thus
             should be less than 1.
37       * The closer it is to 0 the less tight system will be.
38       */
39      dragCoeff: 0.03,
40
41      /**
42       * Default time step (dt) for forces integration
43       */
44      timeStep : 5,
45
46      /**
47       * Maximum movement of the system which can be considered as
             stabilized
48       */
49      stableThreshold: 0.09
50  };
```

Quellen

[1] Kriesel, David: SpiegelMining – Reverse Engineering von Spiegel-Online. In: *33c3* (2016)

[2] Schröder, Jens: AGOF-News-Top-50: N24-Zahlen explodieren, auch manager magazin und taz mit Riesen-Plus. In: *MEEDIA* (2015)

[3] RSS. *Wikipedia*. URL: https://en.wikipedia.org/wiki/RSS abgerufen am 30.1.2017

[4] Python. URL: https://www.python.org/ abgerufen am 30.1.2017

[5] NoSQL. *Wikipedia*. URL: https://de.wikipedia.org/wiki/NoSQL abgerufen am 20.12.2017

[6] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: Simplified Data Processing on Large Clusters. In: *Sixth Symposium on Operating System Design and Implementation* (2004)

[7] MapReduce. *Wikipedia*. URL: https://de.wikipedia.org/wiki/MapReduce

abgerufen am 7.1.2017

[8] Regulärer Ausdruck. *Wikipedia*. URL: https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck abgerufen am 8.2.2017

[9] Multimenge. *Wikipedia*. URL: https://de.wikipedia.org/wiki/Multimenge abgerufen am 7.1.2017

[10] MongoDB. URL: http://mongodb.com/ abgerufen am 3.2.2017

[11] CouchDB. URL: https://couchdb.apache.org/ abgerufen am 6.2.2017

[12] Fielding, R. ; Reschke, J.: *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing* (RFC Nr. 7230) : RFC Editor; Internet Requests for Comments; RFC Editor, 2014 – Maßstab. — http://www.rfc-editor.org/rfc/rfc7230.txt

[13] D3.js. URL: https://d3js.org/ abgerufen am 15.12.2016

[14] Viagraph.js. URL: https://github.com/anvaka/VivaGraphJS abgerufen am 5.2.2017

[15] Chart.js. URL: http://www.chartjs.org/ abgerufen am 6.2.2017

[16] Microservices. *Wikipedia*. URL: https://de.wikipedia.org/wiki/Microservices abgerufen am 1.2.2017