Was ist Los?

Erhebung und Visualisierung von empierischen Daten verschiedener Internet-Zeitungen

Halbjahresarbeit 2016/2107 von Jaro Habiger

Inhaltsverzeichnis

Idee

Am Anfang dieser Halbjahresarbeit stand die Idee, die aktuelle Nachrichtenlage einfach sichtbar zu machen.

Umsezung

Die Umsetzung 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 zur fertigen Visualisierung zu betrachten. In den nachfolgenden Abschnitten wird dieser Verarbeitungsprozess beschrieben.

Daten Sammeln

Als erstes müssen Daten zur weiteren Verwertung von den Verschiedenen Nachrichtenquelln Gesammelt werden. Dies geschieht über die sogenannten "RSS-Feeds". Bei diesen handelt es sich um ein standartiesiertes Format, über das Nachrichtenanbieter ihre Artikel, inklusive Metadaten wie z.B. den Zeitpunkt der Veröffentlichung, in maschienenlesabrer Form bereitstellen. Im Prinzip werden also die Gleichen Daten bereitgestellt, wie auf der normalen Internetseite, mit dem Unterschied, das sie einfacher mit Programmen verarbeitbar sind.

Diese Darstellung als RSS-Feed ermöglicht es, die Artiel verscheidener Online-Zeitungen zu betrachten, ohne für jede einen komplett neuen Datensammler programmieren zu müssen.

Der Letztendliche Datensammler ist ein Programm, welches ich in Python geschriben habe. Es läd sich den RSS-Feed einer einzelnen Nachrichtenquelle periodisch herunter und verarbeitet ihn weiter. Aktuell wird jeder RSS-Feed alle 10s neu analysiert. 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

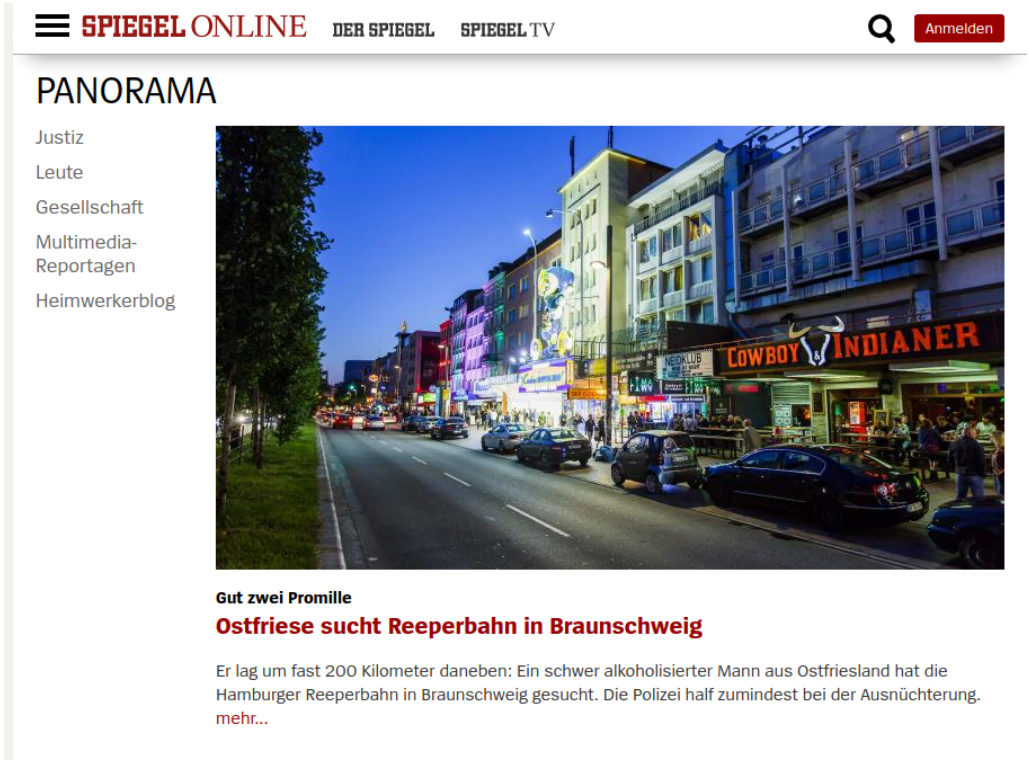

Fig. 1: Ein Artikel einal in der normalen Darstellung...

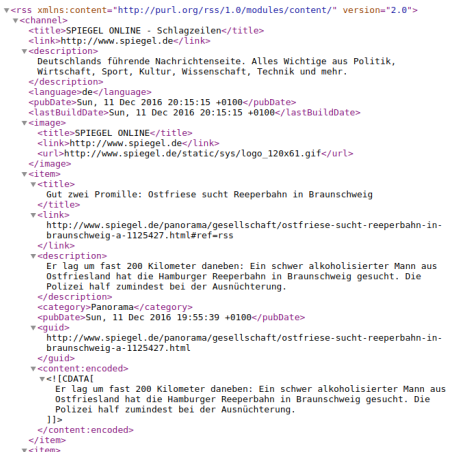

Fig. 2: ... und einmal als Teil eines RSS-Feeds

z.B. den Erstellungszeitpunkt und den herausgeber des Feeds. Die nachflogenden "Äste" enthalten jeweils einen Artikel. Diese Artikel wiederum beinhalten verscheidene unterelemente, d.h. die "Äste" verzweigen sich in weitere, kleinere "Ästchen". In diesen stehen nun z.B. der Author des Textes, der Titel, oder eben der eigentliche Inhalt des Textes. Diese Textuelle Repräsentation einer Baumstruktur gilt es nun zunächst in eine einfacher verwendbare native representation im Speicher des Python Programms umzuwandeln. Hierfür wird eine Programmbiliothek verwendet, die einen sogenannten XML-Parser beinhaltet. Nach diesem schritt können die Einzelnen Artikel betrachtet werden. Hierbei wird zu aller erst überprüft, welche Artikel schn 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 wie folgt aus:

Diese umwandlung ist nötig, da der RSS-Standart zwar die grobe Struktur und ihre repersentation als XML-Datei spezifiziert, letztenendes jeder Nachichtenanbiter allerdings doch nicht ganz Kompatible Feeds ausliefern. Diese Kleinen unterscheide werden Hier also angeglichen, damit die weitere Verarbeitung leichter von statten gehen kann, und keine unterschiede mehr beachtet werden müssen. Zusätzlich hierzu enthält der RSS-Feed nur Kurzfassungen der Artikel. In der Umwandlung werden zusätzlich hierzu noch die Gesammten Artikel von der Internetseite heruntergeladen und der Volltex aus dieser Extrahiert. Hierzu wird zuerst das HTML Dokument der Seite geparsed und die relevanten teile 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.

Am ende der Datenaquirierung werden die gesammelten Datan an die nächste stufe weitergegeben: Die Speicherung.

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 dise flexiblere abfragemethoden ermöglichen. “noSQL” steht hierbei dafür, dass die Datenbank nicht über die Datenbanksprache SQL angesprochen wird, was normalerwiese der fall ist, sondern eine andere schnittstelle bietet. Relativ früh habe ich mich dafür entschieden, dass ich meine Datenbankabfragen als “map/reuce” funktionen formulieren möchte.

map/reduce

Das map/reduce 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, welches wie folgt funktioniert:

1. Am Anfang des Przesses steht eine Menge aus N Eingangsdatensätzen. In meinem Fall sind das die Zeitungsartike, 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, das wir die zu jedem vorkommenden Wort die Häufigkeit bestimmen wollen, wordnet die map Funktion also jedem Artikel die Menge der Darin enthaltenen Wörter zu. Da diese Funktion auf jeden Artikel andgewnd wird, kann man sich in diesem Fall den gesammten map Prozess als eine Funktion vorstellen, in die

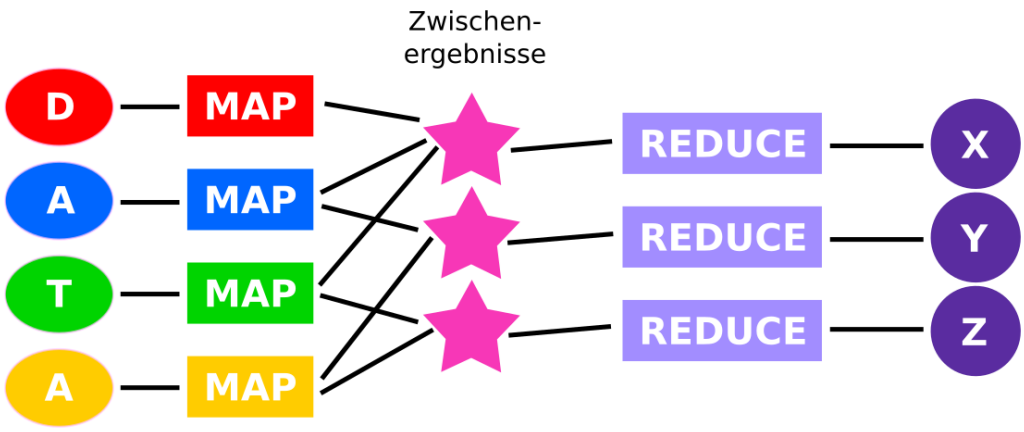

Fig. 3: Der map/reduce prozess als Grafik

die die Menge aller Artikel der Menge der darin enthaltenen Worte zuordnet. Der hierzu zugerhöige Code der map funktion ist:

```
function map() {
var text = this.text.replace(/[^A-Za-zÄäÖöÜüß ]/g, " ");  // entferne alle üb
var words = text.split(" ");  // Teile den text in Wörter
for (var i = 0; i < words.length; i++) {  // für jedes Wort
    var word = words[i];
    if(word) {
      emit(word, 1);  // füge der Ausgabemenge das Wort hinzu
    }
}
}
```

2. Im nächsten Schritt werden Elemente mit Gleichen Schlüsseln Gruppiert. Nach diesem Schritt also eine Menge aus Schlüssel-Wertmengenpaaren (Auch wenn ich immer wieder das Wort Menge verwede, meine ich eigentlich Multimengen, da es relevent ist, wie oft ein bestimmtes Element in der Menge vorkommt). 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:

```
{
```

```
  "der": {1; 1; 1; 1; 1; 1; 1; 1; 1};
  "die": {1; 1; 1; 1};
  ...
}
```

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 einfach nur die elemente der Menge aufummieren muss:

   ```
   reduce(key, values) {
   return values.reduce((previousValue, currentValue) => currentValue + previous
   }
   ```

4. Der vierte und letzte schritt gehört eigentlich nicht mehr zum map/reduce verfahren. Dieser wird nach diesem ausgeführt und dient dazu die ergebisse zu sortieren und eventuell feinheiten zu verbessern. So ist es zum beispiel möglich selten genutzte- oder sogenannte stoppworte in diesem schritt auszusortieren:

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

   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, das die Wörter "Trump" und "Trumps" zusammengezählt werden. Hierbei wird immer das Kürzeste der Worte behalten, da dies meist die Grundform ist.

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" zu arbeite, störte mich aber shr stark an der Komplexität von dieser. 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", welche vom Apache Projekt entwicket wird. Diese erfüllte die meisten meiner anforderungen relativ gut, weshalbi ich meinen gesammten bis dahin existierenden Code an CoucDB anpasste. Nach einiger Zeit des Testens stellte sich allerdings heraus, das CouchDB wahrchienlich zu langsam sein würde und ein unpassendes Rechteverwaltungssystem mitbringt, was die Arbeitsersparniss im gegensatz zu MongoDB zu nichte machen würde. In Folge dieser Erkentniss entschied ich mich dazu, meine Software zurück auf MongoDB zu portieren und für diese Eine HTTP schnitschtelle zu schreiben.

Middleware

Diese HTPP API ist in gewissermaßen das Bindeglied zwischen der Datenbank und der Visualisierung, weshalb es im nacholgenden "Middleware" genannt wird. Sie nimmt HTTP anfragen vom Frontend entgegen, leitet diese an die Datenbank weiter und schickt die Antwort an das Frontend zurück. In sofern könnte man sie auch als übersetzer zwischen verschiedenen Protokollen verstehen. Diese ist notwendig, da ich die Visualisierung im Browser implementieren möchte und im Browser nur sehr wenige Protokolle verfügbar sind. HTTP ist eines dieser wenigen Möglichkeiten und bietet scih an, da es verhältnissmäßig einfach zu implementieren ist.

Visualisierung

Ich habe mich entschieden die Visualisierung im Browser zu schreiben. Dies

- d3.js
- Flexibel

Zusammenführung

Um die einzelen Teilkomponenten oder auch "Microservices" zusammenzuführen habe ich DOcker verwendet.

- Mehr details

Beobachtungen

Evaluation des Verfahrens

Ergebniss

Technisch

- Map/Reduce bei jedem Query skaliert nicht :(

Verschiedene Zeitungen im Vergleich

Fazit