GRAPHAUSWERTUNG

Ermittlung der Stimmungsmacher im Bundestag

Markus Christopher Glutting, Marie Bittiehn, Miriam Lischke

20. Februar 2021

Deutscher Bundestag

Betreut von Prof. Dr. Thomas Hoppe
Informationssysteme
M.Sc. Angewandte Informatik
Hochschule für Technik und Wirtschaft
Treskowallee 8, 10318 Berlin, Deutschland

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der folgende Teilabschnitt der Ausarbeitung beschäftigt sich mit dem sechsten Teilprojekt: die Graphauswertung. Die Projektgruppe, welche an dem Teilprojekt gearbeitet hat, besteht aus Markus Glutting, Miriam Lischke und Marie Bittiehn.

1.1 Hintergrund

Das Teilprojekt "Graphauswertung" baut auf den Ergebnissen der Teilprojekte "Interaktion zwischen Personen" (Gruppe 4) und "Interaktion zwischen Fraktionen" (Gruppe 5) auf. Genauer formuliert, besteht die Aufgabenstellung darin, die Graphen, welche von Gruppe 4 und 5 erstellt werden, auszuwerten und die Ergebnisse der Auswertung der nachfolgenden Gruppe 8 für ihre Benutzeroberfläche zur Verfügung zu stellen.

1.2 Problemstellung

Durch die Auswertung der Graphen sollen die sogenannten "Stimmungsmacher" im Bundesstag ermittelt werden. Unter einem Stimmungsmacher ist im vorliegenden Kontext eine Person gemeint, welche viel mit vielen verschiedenen Personen redet und somit eine Stimmung verbreitet. Ob diese verbreitete Stimmung positiv oder negativ ist, ist dabei nicht entscheidend.

Neben der Ermittlung von Stimmungsmachern sollen ebenfalls simple mathematische Analysen auf den Graphen durchgeführt werden. Dadurch soll eine Gesamtbetrachtung der Sitzungen einer Wahlperiode ermöglicht werden, ebenso wie die Option die Sitzungen in Vergleich zueinander stellen zu können.

1.3 Zielsetzung

Um Stimmungsmacher zu ermitteln, soll der PageRank-Algorithmus (siehe Kapitel 2) verwendet werden. Dies bietet sich an, da Stimmungsmacher gleichbedeutend sind zu Personen, welche viele Nachrichten mit einem positiven und/oder einem negativen Sentiments empfangen bzw. versenden. Mithilfe des PageRank-Algorithmus werden eben diesen Personen bzw. Knoten im Graphen hohe Ränge vergeben, wodurch sie identifiziert werden können.

Für die mathematischen Analysen sollen Berechnungen auf den gesamten Graphen durchgeführt und statische Größen wie bspw. der Median oder ein Quartil der Sentiments berechnet werden.

Die erstellten Analysen sollen der nachfolgenden Projektgruppe (Nutzeroberfläche) über Schnittstellen zur Verfügung gestellt werden.

Für die Bearbeitung des Teilprojekts wurden zu Projektbeginn folgende Lernziele identifiziert:

- Kennenlernen des PageRank Algorithmus
- Kennenlernen von graphenbasierten Datenbanken, insb. Neo4j mit Cypher
- Wissensaufbau im Bereich Backend Web Applikationen

1.4 Prozess

Die Bearbeitung des Teilprojekts lässt sich in vier Phasen gliedern:

- Phase 1 (Oktober 2020): Thematische Einarbeitung und Projektplanung
- Phase 2 (November - Dezember 2020): Implementierung der geplanten Features
- Phase 3 (Januar 2021): Anpassungen in Rücksprache mit der nachfolgenden Projektgruppe (Nutzeroberfläche)
- Phase 4: (Februar 2021): Dokumentation

Die Umsetzung des Projekts erfolgte arbeitsteilig durch alle Mitglieder des Projektteams. Darüber hinaus bestehen folgende Verantwortlichkeiten:

Tabelle 1: Gruppe 7 (Graphauswertung) - Aufgabenverteilung

Aufgabe	Gruppenmitglieder
Server-Administration	Markus Glutting
Implementierung: PageRank	Marie Bittiehn
Implementierung: Stammdaten	alle
Implementierung: Statistische Analysen	Miriam Lischke
Organisation und Projekt Setup	Markus Glutting
Dokumentation	alle

Die Entwicklung erfolgte in wöchentlichen Zyklen. Zu Beginn eines jeden Zyklus fand die wöchentliche Plenarsitzung im Rahmen der Lehrveranstaltung statt. Im Anschluss daran erfolgte eine gruppeninterne Absprache, in der die zuletzt abgeschlossenen Aufgaben besprochen und die nächsten zu

erledigenden Aufgaben identifiziert wurden. Diese Aufgaben wurden mithilfe von GitHub Issues dokumentiert und der verantwortlichen Person zugewiesen. Bis zur nächsten Plenarsitzung erfolgte die eigenverantwortliche Bearbeitung der Aufgaben.

2 Grundlagen

In diesem Kapitel werden die notwendigen Kenntnisse für das Verständnis der nachfolgenden Kapitel vermittelt. Zuerst wird der PageRank-Algorithmus erläutert mit Fokus auf eine effizientere Berechnung der Ränge mithilfe von Matrizenmultiplikationen. Nachfolgend wird auf die wichtigsten verwendeten Entwicklungsframeworks, -Tools und -Konzepte eingegangen.

2.1 PageRank

Der PageRank-Algorithmus ist ein Algorithmus zur Gewichtung von Knoten innerhalb eines Netzwerks anhand der Anzahl ihrer eingehenden Beziehungen. Dabei geht in die Berechnung des PageRanks eines Knoten, neben der Anzahl an eingehenden Beziehungen, ebenso der PageRank der auf ihn verweisenden Knoten mit ein.

Aufgrund der daraus entstehenden direkten Abhängigkeit des PageRank eines Knoten von den PageRanks der auf ihn verweisenden Knoten, ist vor allem bei großen Netzwerken ein genaue Berechnung aller PageRanks nicht immer in absehbarer Zeit möglich. Aus diesem Grund werden die Werte für die PageRanks bei der Berechnung meist iterativ angenähert. Dabei wird allen Knoten ein einheitlicher Startwert als PageRank vergeben. Meist wird als Startwert der Wert $1/N$ verwendet, wobei N die Anzahl aller Knoten des Netzwerks ist. Danach wird der PageRank für alle Knoten mehrfach berechnet und so die Werte iterativ angenähert.

Eine Möglichkeit dieser iterativen Annäherung ist ein mehrfaches Iterieren über das gesamte Netzwerk und die rekursive Berechnung des PageRanks für jeden Knoten. Da im vorliegenden Anwendungskontext mit mehreren hundert Knoten und zig Tausend von Beziehungen zwischen den Knoten gerechnet werden kann, wäre das Iterieren über alle Knoten und Beziehungen nicht sehr effizient. Stattdessen können allerdings Matrizenmultiplikationen und eine Eigenvektorberechnung für die PageRank-Berechnung verwendet werden.

Das Netzwerk, welches aus mathematischer Sicht als gerichteter Graph betrachtet werden kann, kann als eine quadratische, stochastische Matrix abgebildet werden [**pagerank_eigenvector**]. Stochastisch bedeutet in diesem Kontext, dass die Summen der Spalten der Matrix alle 1 betragen [**pagerank_eigenvector**]. Wenn im Graphen von einem Knoten j eine Beziehung zum Knoten i besteht, dann wird in der Matrix an der Stelle ij der Wert $1/d^+(j)$ eingetragen, wobei $d^+(j)$ für den Ausgangsgrad von j steht [**pagerank_eigenvector**]. Wenn keine Beziehung besteht, wird an der Stelle ij eine 0 eingetragen [**pagerank_eigenvector**]. Für den Fall, dass ein

Knoten gar keine ausgehenden Beziehungen besitzt, wird in seiner gesamten Spalte in der Matrix der Wert $1/N$ eingetragen [**pagerank_eigenvector**]. Dadurch wird verhindert, dass der PageRank in Sackgassen-Knoten gewissermaßen "versickert".

Wurde der Graph nun als Matrix abgebildet, so kann der PageRank iterativ mithilfe von Matrixmultiplikationen berechnet werden. Dabei wird der PageRank als Vektor der Länge N dargestellt, meist befüllt mit dem Startwert $1/N$ für alle Knoten. Statt einer gesamten Iteration über das Netzwerk und den rekursiven Berechnungen neuer PageRanks für alle Knoten, muss nun nur eine simple Multiplikation der Matrix mit dem Vektor durchgeführt werden. Das Ergebnis der Multiplikation ist wieder ein Vektor und dieser enthält die neuen PageRanks aller Knoten. Nach mehrfacher Multiplikationen der Matrix mit dem PageRank-Vektoren ändern sich die Werte innerhalb des Vektor nicht mehr. Es wurde iterativ ein dominanter Eigenvektor der Matrix berechnet, welcher gleichzeitig den endgültigen PageRanks der Knoten des Netzwerks entspricht [**pagerank_eigenvector**].

Auf gleiche Art und Weise kann der sogenannte "Reverse PageRank" berechnet werden. Bei dem Reverse PageRank handelt es sich um den PageRank berechnet auf dem Reverse-Graphen des Originalgraphen [**reverse_pagerank**]. Dies bedeutet, dass der Reverse PageRank nicht basierend auf den eingehenden Beziehungen der Knoten berechnet wird, sondern auf den ausgehenden Beziehungen.

Um im ersten Schritt den Reverse-Graphen des Originalgraphen zu erhalten, werden die Richtungen der Beziehungen zwischen den Knoten des Originalgraphen umgedreht [**reverse_pagerank**]. Danach wird der so erzeugte Reverse-Graph ebenfalls als Matrix abgebildet und die PageRank-Werte der Knoten werden mithilfe von Matrizenmultiplikationen berechnet. Die dadurch ermittelten Werte für den Reverse PageRank ähneln den Ergebnissen des klassischen PageRank dahingehend, dass die einzelnen Werte gleich sind, aber anders verteilt. Dies ist dadurch zu erklären, dass bei der Berechnung des Reverse PageRank zwar die Richtungen der Beziehungen umgedreht werden, jedoch keine Beziehungen entfernt oder hinzugefügt werden. Aus diesem Grund sind die Werte gleich, aber durch das Umdrehen der Beziehungen sind sie anders verteilt.

2.2 Entwicklungsframeworks, -Tools und -Konzepte

Vor der Wahl eines Entwicklungsframeworks galt es zunächst zu entscheiden, welche Programmiersprache verwendet wird. Innerhalb der Gruppe war die Python-Bibliothek numpy bekannt, welche zur performanten Berechnung von Matrizen geeignet ist.[**numpy**] Daher fiel die Entscheidung auf die Pro-

grammiersprache Python. Zur Bereitstellung von HTTP-Schnittstellen für die Benutzeroberfläche wurde das Microframework Flask verwendet. Flask zeichnet sich dadurch aus, dass es einen sehr schlanken Kern hat, der nach Bedarf erweitert werden kann.[**flask**] Dadurch unterscheidet es sich stark von anderen Frameworks wie bspw. Django, welches stattdessen eher darauf abzielt, ein vollständiges Framework für Webanwendungen zu sein.[**django**]

Der Zugriff auf die Neo4j Datenbaken erfolgt mithilfe des Neo4j Python Treibers, welcher im Python Paket `neo4j` bereitgestellt wird.[**pip_neo4j**] Es wurde kein Object Graph Mapper verwendet, da eines der Lernziele darin bestand, die Abfragesprache Cypher kennenzulernen.

3 Anforderungsanalyse und Konzept

Die allgemeinen Anforderungen ergeben sich aus der Aufgabenstellung:

- Ermittlung von Stimmungsmachern im Bundestag
- Mathematische Analysen bezüglich des Sentiments im Bundestag

Zur Identifikation detaillierter Anforderungen fand im frühen Projektverlauf eine Absprache mit Gruppe 8 statt, welche die Nutzeroberfläche entwickelt. Im Rahmen dieses Meetings wurde festgelegt, dass folgende Funktionalitäten notwendig sind:

- Abfrage von Stammdaten:
  - Sitzungen
  - Fraktionen
  - Abgeordnete
- Abfrage des Graphen für Fraktionen und Abgeordnete. Die Übertragung des Graphen erfolgt durch gleichzeitige Übertragung der Beteiligten Knoten (Abgeordnete bzw. Fraktionen) und der Nachrichten, die zwischen diesen verschickt wurden. Dabei werden Nachrichten zwischen zwei beteiligten Knoten aggregiert, um die übertragene Datenmenge zu reduzieren.
- Mathematische Analysen bezüglich des Sentiments im Bundestag. Als Ziel wurde die Darstellung von Boxplots ausgegeben. Daher wurde festgelegt, dass Minimum, unteres Quartil, Median, oberes Quartil und Maximum berechnet werden. Zum Vergleich des Sentiments innerhalb verschiedener Sitzungen soll die Möglichkeit der Filterung nach Sitzung vorgesehen werden.

3.1 Architektur

TODO: Diagramm + Beschreibung

3.2 Schnittstellen

1) HTTP Endpoints für Nutzeroberfläche
2) Datenbankzugriff auf Neo4j Datenbanken

4 Implementierung

4.1 Umsetzung

Die Backend-Anwendung wurde als Python Flask Applikation entwickelt. Der gesamte Quellcode ist im GitHub Repository *Sentiments-of-Bundestag/ Graphenauswertung* [**github_graphenauswertung**] verfügbar.

Die Applikation wird durch Ausführen der Datei `app.py` gestartet. Diese stellt zunächst die Verbindung mit den beiden Neo4j Datenbanken her. Zur Kommunikation mit den beiden Datenbanken existiert jeweils eine Klasse, die den Zugriff auf die jeweilige Datenbank verwaltet und Methoden zur Abfrage der Inhalte der Datenbank bereitstellt. Die Zugangsdaten für die Datenbanken werden als Umgebungsvariablen bereitgestellt. Diese werden beim Start der Applikation aus der Datei `.env` eingelesen. Aus Sicherheitsgründen, ist diese Datei nicht Teil der Quellcodeverwaltung. Einige komplexere Prozeduren wie bspw. die Berechnug des PageRank wurden ausgelagert und werden von den Datenbankklassen importiert. Die Defintion der Schnittstellen erfolgt in Form von Methoden, die mit dem Pfad annotiert sind, unter dem die jeweilige Schnittstelle erreichbar ist. Aufgrund der großen Datenmenge und der damit verbundenen hohen Laufzeit werden alle Anfragen für 12 Stunden gecached. Dazu wird die Bibliothek Flask-Caching mit dem Cache Typ SimpleCache verwendet, welcher einem In-Memory Python Dictionary entspricht.[**flask_caching**] Die Implementierung der Schnittstellenmethoden besteht in der Regel aus einem Aufruf einer Methode der Datenbankklassen, wobei das Ergebnis zur Übertragung in der HTTP Response in das JSON Format umgewandelt wird.

Die Methoden der Datenbankklassen sind alle nach dem gleichen Muster aufgebaut. Die nachfolgende Methode, welche die verfügbaren Fraktionen abfragt, kann als Beispiel dienen.

```
def get_factions(self,
                 sentiment_type="NEUTRAL",
                 session_id=None):

    where = self.get_where_clause(sentiment_type, session_id)
    with self.driver.session() as session:
        query = \
            "MATCH (sender:{0})-[r:{1}]-(recipient:{0}) " \
            "{2} " \
            "with sender, " \
            "collect(distinct r.sessionId) as sesList, " \
            "collect(r) as rlist unwind rlist as r " \
            "RETURN DISTINCT sender.name as name, " \
            "sender.size as size, " \
```

```
15              "sender.factionId as factionId, " \
16              "sesList as sessionIds" \
17              .format(NODE_FACTION, REL_COMMENTED, where)
18
19      factions = session.run(query)
20      return factions.data()
```

Listing 1: Zugriff auf Neo4j DB: get_factions

Die Methoden sind stets so aufgebaut, dass zunächst eine Datenbanksitzung geöffnet wird. Zur Datenbankabfrage wird eine Cypher Query konstruiert. Diese hängt von den Parametern der HTTP-Anfrage ab, welche in der `WHERE` Klausel der Query abgebildet werden. Da viele Methoden die gleichen Parameter unterstützen, ist die Konstruktion der `WHERE` Klausel in eine eigene Methode ausgelagert. Die Query wird anschließend an die jeweilige Neo4j Datenbank geschickt. Das Resultat wird in einigen Fällen noch weiter verarbeitet und dann als Resultat zurückgegeben. Mit dem Abschluss der Methode wird auch die Datenbanksitzung beendet.

Eine Herausforderung bei der Anbindung der Datenbanken bestand darin, dass beide zwar auf Neo4j basieren, jedoch wurde die Beziehung zwischen zwei Knoten (Abgeordnete bzw. Fraktionen) unterschiedlich abgebildet. Im Graphen der Abgeordneten wurde eine Nachricht zwischen zwei Knoten als zusätzlicher Knoten abgebildet, während im Graphen der Fraktionen eine Nachricht als Beziehung implementiert wurde. Die Arbeit mit der Datenbank der Abgeordneten wurde uns dadurch erleichtert, dass die verantwortliche Gruppe uns beispielhafte Cypher Queries zur Verfügung stellte.

4.2 Bereistellung

Zur Bereitstellung der Anwendung auf einem Server wurde ein Docker Image erstellt, mithilfe dessen die Anwendung gestartet werden kann. Dabei wurde wie in [**flask_docker**] beschrieben vorgegangen. Das Docker Image basiert auf dem Standard Docker Image für Python und wird gebaut, indem zunächst die Bibliotheken, von denen die Anwendung abhängig ist installiert werden. Anschließend wird der Quellcode und die Konfigurationsdateien in das Docker Image kopiert. Zum Start der Anwendung wird ein Shell Skript aufgerufen, welches den uWSGI Applikationsserver startet. HTTP-Anfragen auf dem Standard Port 80 werden mithilfe von nginx an den uWSGI Server weitergeleitet. Um den Start der Anwendung ohne Angabe des Port Mappings zu gestalten wurde außerdem eine `docker-compose` Datei erstellt.

Zur Bereistellung auf dem HTW Server (http://infosys6.f4.htw-berlin.de/) wurde Port 80 in der Firewall geöffnet und git und Docker installiert. Der

Start der neuesten Version der Anwendung auf dem Server erfolgt durch folgende Schritte:

- Zugriff auf den Server via `ssh`
- Herunterladen der neuesten Version des Quellcodes mithilfe von `git clone` bzw. `git pull`
- Erstellung des neuen Docker Images mithilfe von `docker-compose build`
- Start der Applikation mithilfe von `docker-compose up`

Die verfügbaren Schnittstellen wurden in der `README` des Repositories beschrieben und so der nachfolgenden Gruppe (Nutzeroberfläche) zugänglich gemacht. Die Nutzeroberfläche verwendet die Schnittstellen seither als Datengrundlage für sämtliche Diagramme auf der Website.

5 Fazit

5.1 Zusammenfassung

5.2 Lernziele

5.3 Ausblick