Allgemein

Herausforderungen (Felix)

Neben technischen Herausforderungen in der Umsetzung des Projekts stellten sich auch einige zusätzliche organisatorische Herausforderungen.

Durch die nach dem Projektstart erfolgte Zuteilung eines Teammitglieds in das Team, war eine schnelle Kontaktaufnahme und Integration des neuen Teammitglieds notwendig. Dies erforderte eine schnelle Einarbeitung und Anpassung der Teamdynamik. In Folge sollten auch die (wöchentlichen) Meetings remote absolviert werden.

Trotz dieser zusätzlichen Schwierigkeiten wurden alle Herausforderungen bisher hervorragend bewältigt. Um die Ressourcen für alle Teammitglieder ständig bereit zu halten, war bereits ein Repository auf GitHub angelegt, dieses wurde um eine Datei `Meetings.md` zur Protokollierung der in den remote Meetings besprochenen Inhalte erweitert. Die Meetings wurden fest wöchentlich und zusätzlich nach Bedarf angesetzt und finden virtuell über Zoom statt, des Weiteren ist ein ständiger Kommunikationskanal zum Austausch über WhatsApp, für kurzfristige Änderungen oder dringende Probleme, verfügbar. Neben der bekannten Herausforderung von Videokonferenzen, bietet Zoom die Chance, den Bildschirm für "Code-Reviews" und "Code-Vorstellungen" zu teilen.

Matching

Base Matcher (Felix)

Für die Matcher die eignen Python Code benötigen, ist ein ähnlich modularer Aufbau geplant wie bei den Scrapern. Dies erfüllt vor allem den Zweck die Module übersichtlich zu halten und redundanten Code zu vermeiden. Ziel ist es Fähigkeiten, die jeder Matcher benötigt im `BaseMatcher` zu bündeln und zu vereinheitlichen. Zum aktuellen Zeitpunkt enthält der `BaseMatcher` die Funktionalität zum Schreiben in das Matching File `matches_<Nachrichtenquelle>.csv`.

Simple Matcher (Felix)

Wie im Späteren beschrieben, sollen zum Matchen auch aufwendigere Methoden (z.B. LLMs) zum Einsatz kommen. Dies ist allerdings nicht immer erforderlich und teilweise lassen sich Matches auch ganz trivial ermitteln. So stellt beispielweise MDR zum jeweiligen Artikel in leichter Sprache einen Link zum Artikel in normaler Sprache zur Verfügung. Der `SimpleMatcher` bündelt diese trivialen Arten des Matchens, beispielsweise über eine Funktion, der man bereits die zusammengehörenden URLs übergibt und dieser die passenden Dateipfade in das Matchingfile schreibt.

Datenstruktur (Felix)

Auf Anraten von Professor Baumann wird für die Speicherung keine SQL-Datenbank benutzt sondern wie in der Abbildung dargestellt eine Ordnerstruktur.

```
|-- data
|   |-- deutschlandfunk
|   |   |-- easy
|   |   |   |-- 2024-03-15-Bundes-Wehr_beteiligt_sich_an_Luft
-Bruecke_fuer_den_Gaza-Streifen_
|   |   |   |   |-- audio.mp3
|   |   |   |   |-- content.txt
|   |   |   |   |-- metadata.json
|   |   |   |-- lookup_deutschlandfunk_easy.csv
|   |   |-- hard
|   |   |   |-- 2024-04-25-Angebliche_Drohnenangriffe_Belarus_
erhebt_Vorwuerfe_gegen_Litauen_-_Dementi_aus_Vilnius
|   |   |   |   |-- content.txt
|   |   |   |   |-- metadata.json
|   |   |   |-- lookup_deutschlandfunk_hard.csv
|   |   |-- matches_deutschlandfunk.csv
```

Abbildung: Struktur des zur Speicherung genutzten Dateisystems am Beispiel von Deutschlandfunk (hard) und Nachrichten Leicht (easy) (reduziert auf jeweils einen Artikel)

Für jede Nachrichtenquelle findet sich im `data` Verzeichnis ein Unterordner. Da das Matchen lediglich innerhalb derselben Quelle (also z.B. nur Deutschlandfunk zu Nachrichtenleicht, nicht DLF zu MDR) stattfinden soll, findet sich die Datei mit den jeweiligen Matches (`matches_<Nachrichtenquelle>.csv`) auf dieser Ebene (z.B. `deutschlandfunk`). Jedes der Unterverzeichnisse ist wiederrum aufgeteilt in die Ordner `easy` und `hard`, wobei easy die Nachrichten in leichter Sprache enthält und hard die Nachrichten in Standardsprache. Hier findet sich für jeden gespeicherten Artikel ein eigener Ordner mit der Benennungsstruktur `<Jahr>-<Monat>-<Tag>_<Titel>`. Im Ordner zum jeweiligen Artikel findet sich jeweils `content.txt`, der Haupttext des Artikels, `metadata.json`, der Verschiedene Metadaten wie URL, Autor und Datum in einem über alle Nachrichtenquellen standarisierten JSON- Format enthält, sowie `audio.mp3`, falls der Artikel als vorgelesene Version als Audio verfügbar ist.

Für eine effiziente Suche der Artikel nach ihren jeweiligen Links ist jeweils eine so genannte `lookup-<Nachrichtenquelle>-<easy oder hard>.csv` implementiert. In diesem wird für jeden gespeicherten Artikel jeweils der Dateipfad und der Link im CSV-Format abgespeichert. Die URL des Artikels wird auch in den Metadaten gespeichert, dennoch entstand die Idee des redundanten Speicherns um für die Suche nach der URL nicht über das ganze Verzeichnis iterieren, sondern lediglich eine CSV-Datei analysieren zu müssen. Besonders bei großen Datenmengen ist so eine bessere Effizienz erhofft. Leider standen für Tests

zum Midterm Review noch keine großen Mengen an gescrapten Artikeln zur Verfügung, dennoch zeigte sich bereits bei wenigen gespeicherten Artikeln eine minimal bessere Effizienz in der Suche nach der URL (über den Lookuptable), gegen eine Suche nach dem Titel desselben Artikels (über Iteration über das Verzeichnis).

```
'''Getestet wurde die Suche über den Link des Artikels
sowie den Titel. Die Funktion sucht bei der url automatisch
im Lookuptable ansonsten iteriert sie über das Unterverzeichnis.
Das erste Argument e steht dafür, dass das easy Unterverzeichnis
durchsucht werden soll'''
dh.search_by("e", "url", "https://www.deutschlandfunk.de/ \
zahl-der-arbeitslosen-sinkt-im-april-um-20-102.html")
dh.search_by("e", "title", "Zahl der Arbeitslosen sinkt \
im April um 20.000")
# Output
"Time taken for search_by url: 0.0020 seconds"
"Time taken for search_by title: 0.0071 seconds"
```

Die Speicherung im eigenen Format bietet viel Flexibilität und Unabhängigkeit von Versionen eines Datenbankmanagementsystems. Allerdings stellt sich die Herausforderung eines komfortablen, einheitlichen und effizienten Zugriffs auf die Daten. Hierfür wurde im Projekt die Klasse `DataHandler` definiert. Diese bietet ein Interface für den **Zugriff** auf die Daten durch Funktionen wie `head`, welcher die ersten n Artikel als Pandas DataFrame zurück gibt. Des Weiteren soll eine einheitliche **Speicherung** durch vordefinierte Speicherfunktionen sichergestellt werden. Auch ermöglicht der DataHandler eine **Suche** im Verzeichnis nach Metadaten. Um keine Artikel doppelt zu Scrapen gibt es außerdem die Funktion `is_already_saved`, welche sich die bessere Sucheffizenz der Lookuptable zunutze macht. Sie gibt zurück, ob die URL bereits gescraped und gesaved wurde. Das DataHandler Objekt muss mit der jeweiligen Nachrichtenquelle initialisiert werden (aktuell `"dlf"`, oder `"mdr"`) und kann dann für das jeweilige Unterverzeichnis genutzt werden. Die Initialisierung mit der Nachrichtenquelle soll unter anderem einer Vermischung der Daten vorbeugen. Den meisten Funktionen muss übergeben werden, ob im `"hard"` (`"h"`), oder `"easy"` (`"e"`) Unterverzeichnis gelesen oder geschrieben werden soll.