Praktikum
Multilingual systems with AI

Projektdokumentation

Telegram and News Sites NLP

Gruppe 03
Ahmad Ata Ul Haye
Clarissa Landzettel

SoSe 2022
Prof. Dr. Alexander Mehler
Mevlüt Bagci

August 29, 2022

Contents

1 Extraktion von Telegram

1.1 Aufgabenbeschreibung

Bei dem Telegram NLP geht es zunächst um die Extraktionen von allen Inhalten, die in privaten und auch öffentlichen Chats vorhanden sind [6, p.5]. In einem zweiten Schritt sollen die extrahierten Daten im JSON-Format strukturiert gespeichert werden. Hierbei wird zwischen Metadaten (Infos zum Chat und zu den Mitgliedern) und Chatverlauf (alle Nachrichten) unterschieden. Als letzter Schritt wird der Chatverlauf einer beliebigen Gruppe als ein Netzwerk visualisiert. Hierdurch können die Zusammenhänge zwischen den einzelnen Nachrichten dargestellt werden, wenn beispielsweise eine Nachricht auf eine andere antwortet.

1.2 Umsetzung

Die folgenden Unterpunkte beziehen sich auf die Umsetzung der drei Schritte der Aufgabenstellung: Daten-Extraktion aus Telegram, Erstellung der JSON-Dateien und die Visualisierung des Chatverlaufs.

1.2.1 Daten Extraktion

Für die Möglichkeit der Extraktion von Telegram muss zunächst über die offizielle Telegram-Website eine Applikation erstellt werden [4, p.5]. Dafür wird die Telefonnummer eines Klienten mit Telegram-Account benötigt. Nach Erstellung der App können die dabei erstellten ID und Hash für die Verwendung einer API benutzt werden. Telegram bietet hierfür zwei verschiedene kostenlose APIs an. Zum einen die Telegram API, die es ermöglicht, einen Client zu erstellen, also ein Programm, das Daten oder Dienste vom Telegram Server abrufen kann. Die zweite API ist die Bot API. Diese ermöglicht die Interaktion zwischen dem Programm-Code und einem Bot, der vorher mithilfe des Telegram Bots *BotFather*[1] erstellt werden muss.
Bei dem Bot handelt es sich um eine Erweiterung des Clients. Alle Funktionen, die der Client besitzt, kann der Bot ebenfalls vorweisen. Zudem hat er Erweiterungen, wie z.B. Inline-Befehle und andere Abfragen. Aufgrund der breit-gefächerten Möglichkeiten des Bots und der gezielten Werbung dafür durch die Online-Community, war die Bot API die erste Wahl zur Bearbeitung der Aufgabe.
Nach einigen Testläufen zeigten sich allerdings einige Nachteile, welche die Bot API für das Projekt ungeeignet macht [3]. Zum einen besitzt ein Bot nur einen begrenzten Cloud-Speicher. Als Folge davon werden Nachrichten vom Server direkt gelöscht, nachdem sie bearbeitet wurden. Somit kann nicht auf ältere Chatverläufe zugegriffen werden. Ein weiterer Nachteil bieten die begrenzten Rechte. Ein Bot muss in alle Gruppen als Mitglied hinzugefügt werden, um dort genutzt werden zu können. Falls bestimmte Informationen zu den Nachrichten und den Mitgliedern abgefragt werden sollen, muss der Bot außerdem Admin-Rechte erhalten. Außerdem dienen die Bots in erster Linie zur Interaktion im Telegram-Chat. Die Funktionen sind darauf ausgelegt, ein Spiel, eine Umfrage oder ähnliches zu erstellen. Bots können hierbei nur auf Events bzw. Befehle reagieren. Die User müssen einen Bot durch einen Befehl aktivieren, somit ist er am ehesten für eine chatinterne Kommunikation geeignet und wir haben für den weiteren Verlauf des Projektes die Telegram API mit dem Client gewählt.

[1] In Telegram: @BotFather

Eine sinnvolle Ergänzung zur Telegram API ist die Bibliothek Telethon, da sie die Interaktion mit den Python-Programmen erleichtert [5]. Die ausführliche Dokumentation und aktive Online-Community half bei der Extraktion der gewünschten Daten.

1.2.2 Erstellung der JSON Dateien

Mithilfe der Telethon Bibliothek konnten drei JSON-Datein erstellt werden (siehe 1.4 Ergebnisse):
In *get_raw_messages.json*) (entsprechend der Telethon-Methode *get_raw_messages()*) ist der komplette Chatverlauf eines Chats enthalten. Zu jeder einzelnen Nachricht sind dabei alle erhältlichen Infos gespeichert. Anhand dieser Sammlung können die für das Projekt nötigen Variablenamen eingesehen werden und sie dient des Weiteren als Vollständigkeitskontrolle.
In *Metadata.json* sind alle relevanten Informationen zu dem jeweiligen Chat und alle darin enthaltenen Mitglieder aufgeführt.
In *Chathistory.json* ist der komplette Chatverlauf eines Chats ordentlich und gefiltert aufgeführt. Die Struktur entspricht dabei den Bedingungen für die Visualisierung. Gleichzeitig mit der Erstellung dieser Datei werden zudem alle Medien heruntergeladen, die mit den Nachrichten geschickt wurden. Die JSON-Dateien und heruntergeladenen Bilder und Medien werden alle lokal in dem Ordner *Telegram_Data* gespeichert.

1.2.3 Visualisierung

Für die Visualisierung von *Chathistory.json* wurde die JavaScript Bibliothek d3.js genutzt [1]. Im Vergleich zu anderen Möglichkeiten hat diese Bibliothek am besten zu diesem Projekt gepasst und konnte durch die ausführliche Dokumentation leicht genutzt werden.

1.3 Anleitung

Für das Durchführen von *TelegramExtractor.py* müssen sieben Schritte befolgt werden:

1. Führen Sie *TelegramExtractor.py* aus.
2. Folgen Sie den Anweisungen im Terminal und geben Sie eine Telefonnummer ein, die mit einem Telegram Account verbunden ist. Das Format muss +49... sein.
3. Geben Sie den Code ein, den Sie über Telegram erhalten.
4. Es werden alle Chats angezeigt, bei denen die angegebene Nummer Mitglied ist. Der Name des Chats, dessen Inhalte extrahiert werden sollen, müssen Sie in dem Code angegeben. Hierfür muss der String in Zeile 31 für die Variable *chat_name* ersetzt werden.
5. Führen Sie *TelegramExtractor.py* erneut aus, um die JSON-Dateien zu erstellen.
6. Starten Sie einen Webserver im Terminal mit: python -m http.server
7. Öffnen Sie im Browser die Website, um die Visualisierung anzuzeigen: http://localhost:8000/socialmediaextractornlp/index.html.

1.4 Ergebnisse

Zur Vorstellung der Ergebnisse sind nachfolgenden einige Screenshots zu sehen, die die Extraktion aus zwei Testgruppen zeigen. Zum einen eine für die Testzwecke dieser Gruppenarbeit erstellten Gruppe, bestehend aus den Gruppenmitgliedern und einem Bot und zum anderen eine öffentliche Gruppe für den Austausch zu dem Computer-Spiel *Stardew Valley* mit insgesamt 26 Teilnehmern.

1.4.1 Metadaten

In Abb. 1 sind die extrahierten Metadaten der Gruppe *TestGruppe03* abgebildet. Jeder Chat ist einer individuellen Chat-ID zugeordnet und besitzt einen Titel und einen Username. Optional kann eine Beschreibung hinzugefügt werden. Weitere Informationen sind das Erstelldatum, die Anzahl aller Nachrichten und aller Mitglieder und, falls vorhanden, der Dateiname, unter dem das Profilbild gespeichert wurde (siehe Abb. 2). Der Name entspricht dabei dem Username des Chats. Anschließend an diese allgemeinen Informationen werden alle Mitglieder des Chats aufgelistet mit den Daten User-ID, Vorname, Nachname, Username und Speichername des Profilbildes.


```
"Chat-ID": 1712564047,
"Title": "Testgruppe",
"Description": "Diese Telegram-Gruppe dient dem Praktikum MSwAI.",
"Username": "TestGruppe03",
"Created": "2022-05-18T15:59:24+00:00",
"Number of Messages": 89,
"Chat Photo": "TestGruppe03.jpg",
"Number of Participants": 3,
"Number of Admins": 3,
"All Users": {
    "408370873": [
        {
            "User-ID": 408370873,
            "Firstname": "Clarissa",
            "Lastname": "Landzettel",
            "Username": "ClarissaLand",
            "Profile Photo": "408370873.jpg"
        }
    ],
    "1101657635": [
        {
            "User-ID": 1101657635,
            "Firstname": "Ata",
            "Lastname": null,
            "Username": null,
            "Profile Photo": "N/A"
        }
    ],
    "5296969873": [
        {
            "User-ID": 5296969873,
            "Firstname": "ExtractorBot",
            "Lastname": null,
            "Username": "Gruppe3_bot",
            "Profile Photo": "N/A"
        }
    ]
}
```


Abb. 1: Ausschnitt aus *Metadata.json* der Telegram-Gruppe *TestGruppe03* (09.08.2022)

Abb. 2: Chat Foto der Telegram-Gruppe *TestGruppe03* (09.08.2022)

1.4.2 Chatverlauf

Nachfolgend sind drei Chatnachrichten aus der Gruppe *Stardew Valley Germany* abgebildet, wie sie im JSON-Format in *Chathistory.json* gespeichert werden. Beim betrachten von Abb. 3 wird die Struktur einer Nachreicht ersichtlich:

Abb. 3: Ausschnitt aus *Chathistory.json* der Telegram-Gruppe *Stardew Valley Germany* (09.08.2022)

Jede Nachricht besitzt eine Message-ID. Hierbei werden im Chat die Nachrichten chronologisch durchnummeriert, angefangen mit der ersten Nachricht, die geschrieben wurde. Der Absender wird im Format "Vorname Nachname (Username)" angegeben. Zudem kommt das Datum und die Uhrzeit des Schickens und ob die Nachricht im Chat zurzeit angeheftet ist. Mit Hinblick auf die Visualisierung der Verlinkungen ist gespeichert, wie oft eine Nachricht weitergeleitet wurde und welche Message-ID die ursprüngliche Nachricht hatte, falls es sich selbst und eine Weiterleitung handelt. Ebenso wird die Anzahl aller Antworten auf diese Nachricht angegeben und die Message-ID der Nachricht, auf welche diese Nachricht antwortet. Anschließend sind in zwei Listen einmal der Nachrichtentyp gespeichert und dann der Inhalt selbst. Eine Nachricht kann einen Text, ein Photo, ein Document, eine Invoice, andere Media (alle restlichen Medientypen) oder eine Message Service beinhalten, oder auch eine Kombination aus mehreren dieser Typen. Der Content ist dementsprechend entweder ein Text, der Dateiname der heruntergeladenen Medien oder die Message Service (beispielsweise ein neues Gruppenfoto oder ein neues Mitglied).

1.4.3 Visualisierung

Die zwei Dateien **Metadata.json** und *Chathistory.json* werden in eine HTML so eingebunden, dass die Metadaten links im Browser alle relevanten Informationen zu dem jeweiligen Chat

anzeigen, während im Hauptfenster der Nachrichtenverlauf als Graph interaktiv gestaltet wird (siehe Abb. 4. Jede Nachricht wird durch einen Punkt repräsentiert, der je nach Nachrichtentyp eine andere Farbe hat. Nachrichten, die aufeinander antworten, sind durch einen Pfeil markiert (der Pfeil führt von der Antwortnachricht zu der originalen Nachricht). Durch ein Hover-Effekt werden die weiteren Infos aus *Chathistory.json* angezeigt.

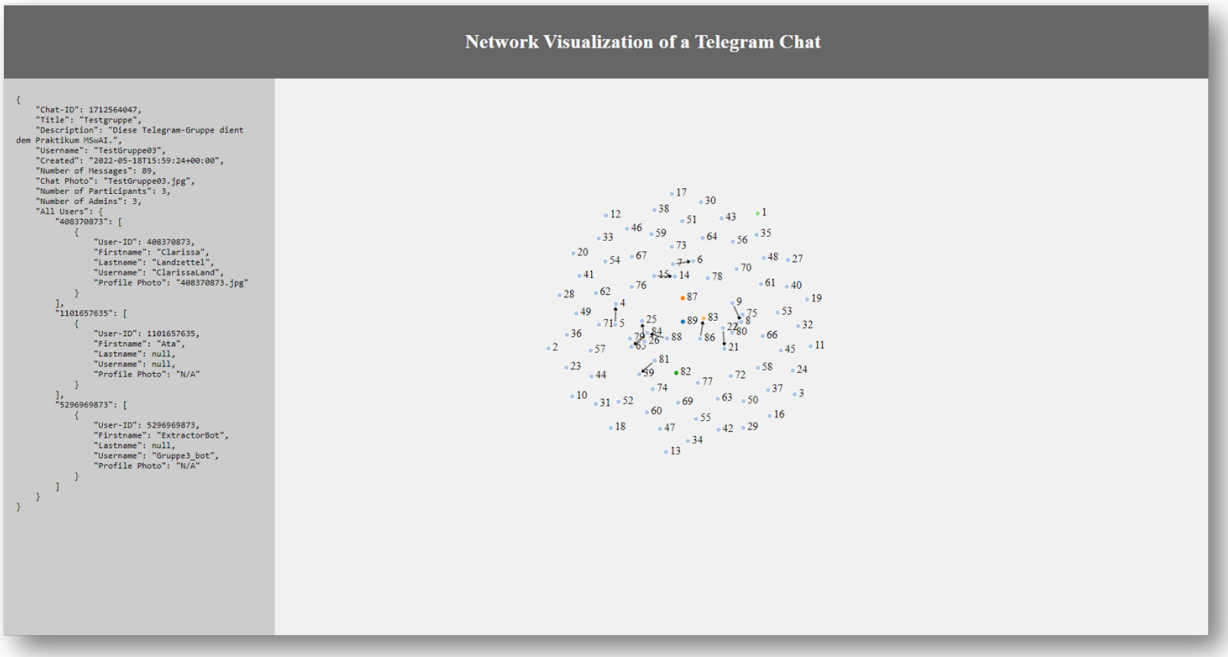

Abb. 4: Visualisierung der Telegram-Gruppe *TestGruppe03* (09.08.2022)

Da größere Chats mit vielen Nachrichten für sehr große unübersichtliche Graphen sorgen, kann die Ansicht durch Scrollbars angepasst werden. Ein Chat, bei dem das für eine leichterer Navigation führt, ist die Telegram-Gruppe *Stardew Valley Germany* in Abb. 5.

1.5 Evaluation

Um die Ergebnisse zu bewerten, sind im Folgenden Erfolge, Grenzen und Optimierungsideen aufgelistet. Zu den Erfolgen gehören persönliche Meilensteine, die sich während der Projektarbeit ergaben. Die Grenzen beziehen sich auf Dinge, die nicht geklappt haben und die vor allem gegen Ende der Projektarbeit aufkamen. Die Optimierungsvorschläge beinhalten Ideen, die nicht umgesetzt wurden, die allerdings für nützlich erachten werden könnten, falls dieses oder ein ähnliches Projekt fortgeführt werden sollte.

1.5.1 Erfolge

Nachdem viel Zeit durch die Arbeit mit der Bot API verloren wurde, konnte mithilfe der Telegram API und des Clients schließlich der Zugriff auf einen vollständigen Chatverlauf einer Gruppe erlangt und dieser extrahiert werden.

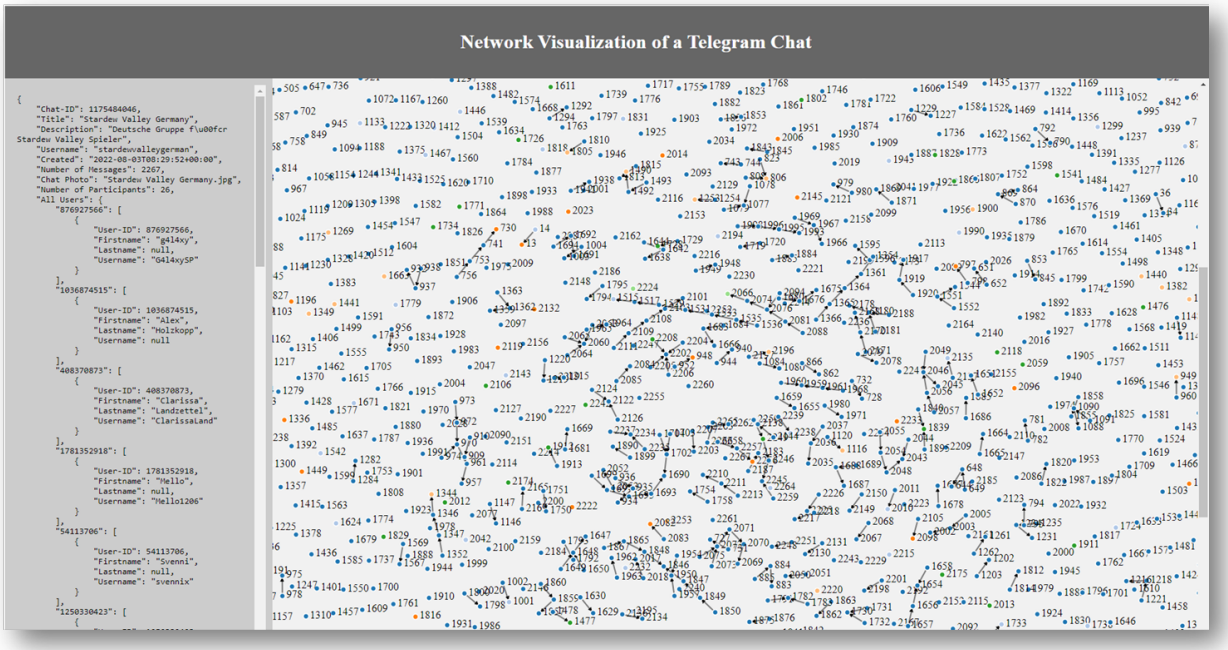

Abb. 5: Visualisierung der Telegram-Gruppe *Stardew Valley Germany* (09.08.2022)

Die JSON-Dateien (vor allem *Chathistory.json*) konnten richtig formatiert werden, um den Bedingungen der Visualisierungs-Bibliothek gerecht zu werden. Beispielsweise müssen die Verlinkungen der Nachrichten in einer ganz bestimmten Struktur im JSON-Format gespeichert werden, damit die d3.js Bibliothek damit arbeiten kann.
Schließlich hat auch die Umsetzung der Visualisierung gut geklappt. Durch Scroll Bars kann das komplette Netzwerk angesehen und auch die Metadaten angezeigt werden. Insgesamt ist der Graph übersichtlich und informativ.

1.5.2 Grenzen

Eine nach einiger Zeit auftauchende Fehlermeldung war der *FloodWaitError*. Telegram möchte verhindern, dass in den Gruppen gespammt wird oder die Nutzer auf jegliche andere Weise belästigt werden. Deshalb wurde eine Pause eingeführt, die abgewartet werden muss, wenn zu oft oder zu lange viele Nachrichten geladen werden oder gehäufte Anfragen zu den Nutzern gemacht wurden. Die Pause beträgt zwischen mehreren Minuten und mehreren Tagen. Dieser Umstand sorgt vor allem in der Testphase für Verhinderungen. Ein Weg, um diese Fehlermeldung zu umgehen wäre vermutlich, die extrahierten Chatdaten in einer Datenbank zu speichern, um die Anfragen nicht ständig an den Telegram Server zu schicken, sondern an die Datenbank.
Neben der Fehlermeldung gab es vereinzelt Probleme bei der Erkennung von Emojis und Sonderzeichen. Beispielsweise wurde eine Daum-hoch Animation von der Telethon Bibliothek als eine Zeichenkette gespeichert, aber anschließend von ASCII und Unicode nicht erkannt. Man könnte alle problematischen Zeichen sammeln und manuell abfangen [2], allerdings kommen mit jeder Gruppe und mit jedem Telegram Update vermutlich neue Fälle dazu.
Schließlich gab es noch Grenzen bezüglich den Informationen, die zu den Mitgliedern und Nachrichten abgefragt werden können. Beispielsweise kann nicht abgerufen werden, wann ein Mitglied einem Chat beigetreten ist oder aus welcher Gruppe eine Nachricht weitergeleitet

wurde. Die Mitglieder werden von Telegram sehr gut geschützt und für die Weiterleitungen ist lediglich die ursprüngliche Message-ID und nicht die Chat-ID angegeben.
Um nun zu dem wohl schwerwiegendsten Fehler zu kommen: Es werden nur ganz bestimmte Chats von dem Code akzeptiert, während vor allem private Chats oder private Gruppen scheinbar von Telegram geschützt werden. Einige Chats, die zu Beginn der Projektarbeit einwandfrei verwendet werden konnten, werfen nun Fehlermeldungen auf. Vermutlich sorgt die Kombination von mittlerweile so vielen Abfragen im Code dazu, dass nur bestimmte Chats für eine Extraktion in Frage kommen. Die Arbeit mit sogenannten Broadcasting-Chats oder öffentlichen Gruppen mit einer großen Anzahl an Mitgliedern scheint allerdings weiterhin gut zu funktionieren. Auch hier würde die Verwendung einer Datenbank vermutlich eine geeignete Lösung sein.

1.5.3 Optimierungsideen

Was vor allem die Visualisierung aufwerten könnte, wäre eine Filteroption. Damit könnte nach bestimmten Nutzern oder nach einem bestimmten Content gesucht werden, was besonders bei einer großen Anzahl an Nachrichten hilfreich wäre. Hierfür wäre allerdings auch der Zwischenschritt über eine Datenbank nötig, da die Server-Abfragen nicht gleichzeitig mit der Visualisierung über einen Localhost stattfinden können. Auch um einen *FloodWaitError* zu verhindern, wäre es besser die Filtermöglichkeit mit einer Datenbank zu verbinden.
Die Informationsdarstellung in der Visualisierung könnte insofern verbessert werden, dass beim drüber hovern über eine Nachricht zusätzlich die Medieninhalte angezeigt werden (und nciht nur der Dateiname). Die Gründe, warum bisher darauf verzichtet wurde, sind zum einen die Übersichtlichkeit und zum anderen die Frage der Relevanz. Beim Anzeigen von Bildern wurde schnell deutlich, dass es zu unübersichtlich ist, wenn diese direkt im Graphen eingebaut sind. Eine elegantere Lösung wäre, dass beim Anklicken einer Nachricht alle Infos in einem separaten Fenster neben dem Graphen geöffnet werden. Hier wäre auch genug Platz für die Medien, ohne dass diese stören. Bei einem Live Standort oder ähnlichem würde es allerdings reichen, wenn nur die Info des Nachrichtentyps angezeigt werden würde, da der Standort selbst ein Ablauf Datum hat.
In diese erweiterte Ansicht könnte außerdem der Extrahierte Text aus Audio- und Video-Dateien eingefügt werden. Zurzeit werden die Medien nur heruntergeladen, da eine direkte Textextraktion eine deutlich längere Laufzeit bedeuten würde. Alternativ könnte in der Visualisierung die Option hinzugefügt werden, dass nur bei Bedarf eine Datei angezeigt wird. Auch zu diesem Zweck würde eine Datenbank diese Idee gut umsetzen können, um nicht wieder einen *FloodWaitError* zu riskieren.

1.6 Mögliche Anwendungen

Der Zweck einer jeden Daten Visualisierung ist es, alle wesentlichen Informationen auf einem leicht lesbaren Bild, oder in diesem Fall einem Graphen, darzustellen [7]. Dadurch kann die Analyse der Daten erleichtert werden und es können Aussagen oder Entscheidungen anhand davon getroffen werden.
Die JSON-Dateien und die Chat-Visualisierung bieten eine gute Übersicht über alle Aktivitäten innerhalb einer Gruppe, sodass beispielsweise durch eine Privatperson oder im Rahmen von polizeilichen Ermittlungen viele Dinge einfach ablesen werden können, ohne durch den Chatverlauf scrollen zu müssen (wer beteiligt sich wie oft und mit welchem Content, wer ist alles Mit-

glied der Gruppe, in welchen Chats gibt es die meiste Interaktion). Für kommerzielle bzw. Marketing Zwecke kann analysiert werden, welche Themen bei einer bestimmten Zielgruppe im Trend sind, indem man schaut, auf welche Nachrichten am häufigsten geantwortet wurden. Wenn beispielsweise eine Firma ein neues Produkt vorstellt und die Reaktionen in verschiedenen Gruppen sammelt, können diese anschließend ausgewertet werden (Sentiment Analyse). Es können auch einfach Daten gesammelt werden. Besonders durch die JSON-Dateien besteht Zugriff auf einen großen Content an Text und Medien und Informationen zu Telegram-Nutzern.

1.7 Fazit

Zusammenfassend bestand das Projekt aus drei Schritten: Zunächst wurden alle Daten aus Telegram extrahiert werden, indem eine Telegram Applikation erstellt und diese mit der Telegram API und der Telethon Bibliothek kombiniert wurde. Dadurch können drei JSON Dateien erstellt werden(raw_messages, Metadaten, Chatverlauf) und alle im Chat enthaltenen Medien heruntergeladen werden. *Chathistory.json* dient anschließend als Datensatz für die Visualisierung, die mittels der d3.js Bibliothek als ein Netzwerk-Graph umgesetzt werden kann.
Die Hauptprobleme nach Abschluss des Projektes sind die Folgenden: Es kann nicht aus alle Chattypen extrahiert werden, sondern nur aus einer Auswahl an öffentlichen Gruppen. Des Weiteren taucht gelegentlich der *FloodWaitError* auf, wenn zu viele Server-Anfragen in kurzer Zeit getätigt wurden. Als ein dritter Punkt könnten noch viele weiter Informationen in die JSON-Dateien und in die Visualisierung integriert werden, beispielsweise die Extraktion von Text aus allen Audio- und Video-Nachrichten.
Neben den teils nützlich teils notwendigen Verbesserungsvorschlägen (siehe 1.5 Evaluation) würde es bei zukünftigen, ähnlichen Projekten Sinn machen, die erzeugten JSON-Dateien in eine Datenbank einzupflegen. Auf diese Wiese kann der *FloodWaitError* umgangen und die generelle Datenverarbeitung vereinfacht werden.

2 Extraktion von News Sites

2.1 Aufgabenbeschreibung

Bei der News Sites Extraktion geht es um die Daten Extraktion von den offiziellen Webseiten der Sender ARD[2] und ZDF[3] sowie der Struktur von einer Website zu erkennen. Die extrahierten Daten, die in der Struktur bei der entsprechenden Website vorkommen, sollen in einer JSON Datei exportiert werden.

2.2 Umsetzung News Sites

Für die News Sites Extraktion wurde einer Klasse nämlich *WebSpider* implementiert. Diese Klasse bietet die ganze Funktionalität der Datenextraktion und der Strukturerkennung.Die Datenextraktion durch *WebSpider* Klasse ist mittels Rekursion gelöst. Ein Objekt der Klasse *WebSpider* wird erstellt, die 5 Parameter erwartet. Die Parameter sind Folgenden:

1. **url**
   Das ist ein pflicht Parameter und erwartet der Link von der Webseite aus dem man Daten

[2]https://www.ard.de/
[3]https://www.zdf.de/

extrahieren möchte.

2. **browser_mode=True**
   Das ist standardmäßig auf wahr gesetzt. Das heißt, das browsen im Web ist im Fensterlosenmodus.

3. **recursive=False**
   Das ist standardmäßig auf falsch gesetzt. Das heißt, die Daten werden nur von gegebenen URL extrahiert, anderen Falls wird ganzen Gebiet von dem gegebenen URL extrahiert.

4. **crawl_portal=False**
   Das ist standardmäßig auf falsch gesetzt. Wenn man auf Wahr setzt wird ganze Portal extrahiert.

5. **data_saving_Directory='DataOutput'**
   Das ist das Verzeichnis, wo die extrahierte Daten gespeichert werden. Das heißt *DataOutput* auf standard Einstellungen.

Um zu browsen mit *Selenium* wird Google Chrome verwendet. Mithilfe von *Selenium* wird HTML-Inhalt abgeholt, was später durch *Beautiful Soup* bearbeitet wird. Für die Implementierung der Klasse *WebSpider* wurden zwei Bibliotheken *Beautiful Soup*[4] und *Selenium*[5] verwendet.

2.2.1 Code Ausführung News Sites

In der Klasse *main.py* ist ein Beispiel Methode: *extract_news()* definiert. Mit dem Aufruf von *main.py* kann man diese Methode Aufrufen. In dieser Methode wird eine Instanz mit den bestimmten Parameter der Klasse *WebSpider* erstellt. Es sind zwei Rekursionstiefen definiert. In dem Fall wird nur auf einem Teilgebiet "Nachrichten-Politik" fokussiert, deshalb wird *recursive=True* gesetzt. Wenn man zum Beispeil das ganze ZDF Potral extrahieren möchte, setzt man *crawl_portal=True*.

```python
def extract_news():
    base_url = 'https://www.zdf.de/nachrichten/politik'
    news_extractor = WebSpider(url=base_url, recursive=True, data_saving_Directory="Nachrichten-Politik")
    news_extractor.scrape_data()
    video_link_file = news_extractor.write_video_links()

    video_links = []
    try:
        file = open(video_link_file, 'r')
        video_links = file.readlines()
    except Exception as e:
        print(e)

    for video in video_links:
        video_to_text(video.strip())
```

Abb. 6: Ausschnitt aus *main.py* der Methode *extract_news()*

[4]https://www.crummy.com/software/BeautifulSoup/bs4/doc
[5]https://www.selenium.dev/documentation

2.2.2 Speicher Struktur News Sites

Für jeden Aufruf von *extract_news()* wird ein Verzeichnis erstellt. Auf Voreinstellung werden Daten in *data_saving_Directory='DataOutput'* gespeichert. Daten von einer Seite zu speichern, wird dynamisch eine JSON Datei mit dem Namen, wie die Seite heißt, generiert und in das hinterlegte Verzeichnis gespeichert. Jede generierte Name hat ein Präfix Hauptname von der Webseite in der Form: Hauptname__ zum Beispiel bei der Extraktion von *https://www.zdf.de/nachrichten/politik* ergibt folgende Präfix "zdf__".

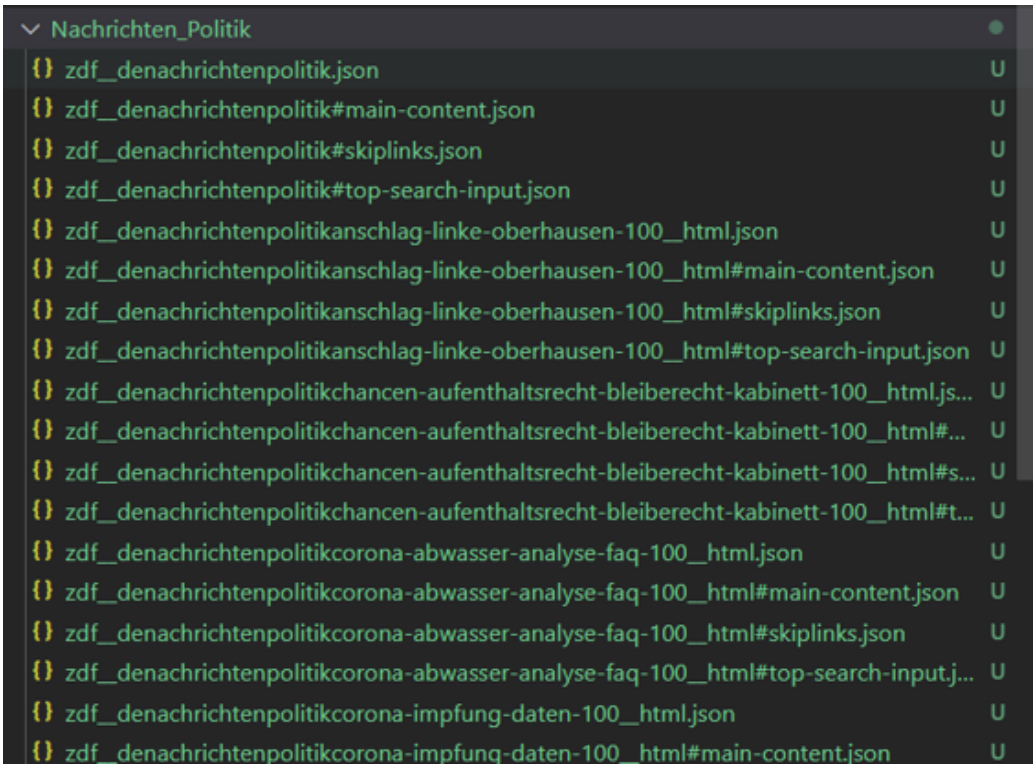

Abb. 7: Aussicht aus *Nachrichten-Politik* Verzeichnis

2.3 JSON Datei Struktur

Alle Daten werden in der Reinfolge, in der die bei einer Webseite vorkommen, gespeichert. Zum Speichern ein Link bzw. einer Unterseite, Bilderdaten, Text Absatz, Videodaten werden spezial Schlüssel generiert. Speziale Schlüssel sorgen dafür, dass keine Daten verloren gehen, da zum Beispiel verschiedene Links gleiche Name haben können. Der Schlüssel von einem Link sieht wie folgt begaut: "LnkS_VMJGO55MQQL_LnkE__Nachrichten"

- Präfix: Lnk_
- 10 Stellige zufällige Schlüssel: VMJGO55MQQL
- Suffix: _LnkE__
- Name der Link: Nachrichten

```
"LnkS_VMJGO55MQQ_LnkE__Nachrichten": "https://www.zdf.de/nachrichten",

"ImgS_J5GLWMSICH_ImgE__ZDFheute": {
    "Src": "https://www.zdf.de/assets/zdfheute-keyvisual-100~2850x300?cb=1644916262455",
    "data-src": "https://www.zdf.de/assets/zdfheute-keyvisual-100~2850x300?cb=1644916262455"
},

"PS_O0B1S8JINY_PE": {
    "paragraph-content": "Russland will das größte Kernkraftwerk an das Stromnetz der
}

"VideoLnkS_7WTCLQIILE_VideoLnkE": {
    "video-source": "https://nrodlzdf-a.akamaihd.net/none/zdf/22/08/220808_gesamt_hli/1/220808_gesamt_hli_2128k_p18v15.webm",
    "video-description": "Erneut laden Video spielt auf  Google Cast ab.    Nachrichten | ZDFheute live  AKW in der Ukraine unter
},
```


Abb. 8: Aussicht aus *JSON* Datei Struktur

Ebenso werden die anderen Schlüssel gebaut. Siehe bitte Abbildung: 8

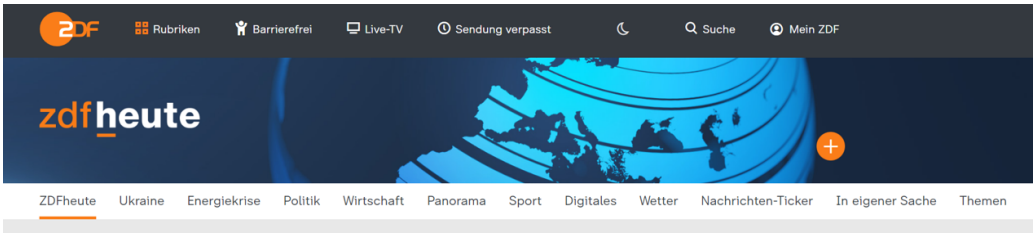

Abb. 9: Navigationsmenü der *ZDF*


```
"ZDFheute-Ukraine-Energiekrise-Politik-Wirtschaft-Panorama-Sport-Digitales-Wetter-Nachrichten-Ticker-In-eigener-Sache-Themen": {
    "LnkS_LFTQEK55N9_LnkE__ZDFheute": "https://www.zdf.de/nachrichten",
    "LnkS_JLJKRMTI9I_LnkE__Ukraine": "https://www.zdf.de/nachrichten/thema/ukraine-198.html",
    "LnkS_PCH9HJHJSR_LnkE__Energiekrise": "https://www.zdf.de/nachrichten/thema/energiesparen-100.html",
    "LnkS_QVAO6TWKC4_LnkE__Politik": "https://www.zdf.de/nachrichten/politik",
    "LnkS_ZCHEFWQQGL_LnkE__Wirtschaft": "https://www.zdf.de/nachrichten/wirtschaft",
    "LnkS_7DSNYM4I5S_LnkE__Panorama": "https://www.zdf.de/nachrichten/panorama",
    "LnkS_SXYYRI2RMX_LnkE__Sport": "https://www.zdf.de/nachrichten/sport",
    "LnkS_V6N99X5K69_LnkE__Digitales": "https://www.zdf.de/nachrichten/digitales",
    "LnkS_BFPYGBV0KY_LnkE__Wetter": "https://www.zdf.de/nachrichten/wetter",
    "LnkS_Z58KLMCL7N_LnkE__Nachrichten-Ticker": "https://www.zdf.de/nachrichten/nachrichtenticker-100.html",
    "LnkS_CDGNZK4YDA_LnkE__In eigener Sache": "https://www.zdf.de/nachrichten/in-eigener-sache",
    "LnkS_VXF77HE177_LnkE__Themen": "https://www.zdf.de/nachrichten/thema"
},
```


Abb. 10: Aussicht der Navigationsmenü der *ZDF* in JSON Format

3 Video to Text Extraktion

3.1 Aufgabenbeschreibung

Bei *Video To Text* geht es um die Text Extraktion. Das extrahierte Text soll man in einer Text Datei exportieren.

3.2 Umsetzung Video To Text

Eine Klasse *VideoToText* wurde implementiert. Diese Klasse bietet die ganze Funktionalität für die Textextraktion. Die Videodatei wird in Audiodatei umgewandelt. Die Umwandlung von Video nach Audio erfolgt mithilfe der Python Bibliothek *moviepy*[6]. Die Audiodatei wird lokal runtergeladen. Mithilfe der *pydub*[7] Bibliothek wird nach Pausen(Silence) die Audiodatei gesplittet. Danch werden die Audio-Chunks Einzel mithilfe der Python-Bibliothek *SpeechRecognition*[8] bearbeitet. Der Text wird aus den gesplitteten Audioteilen extrahiert und wenn alle Audioteile bearbeitet sind, wird der Text von alle Audio-Chunks zusammengeführt. Die Bearbeitung der Audioteile wird mithilfe der *multiprocessing*[9] Bibliothek dynamisch parallelisiert. Nach der Bearbeitung, unabhängig davon, ob die Bearbeitung erfolgreich war oder nicht, werden die runtergeladene Audiodatei sowie die Audio-Chunks gelöscht.

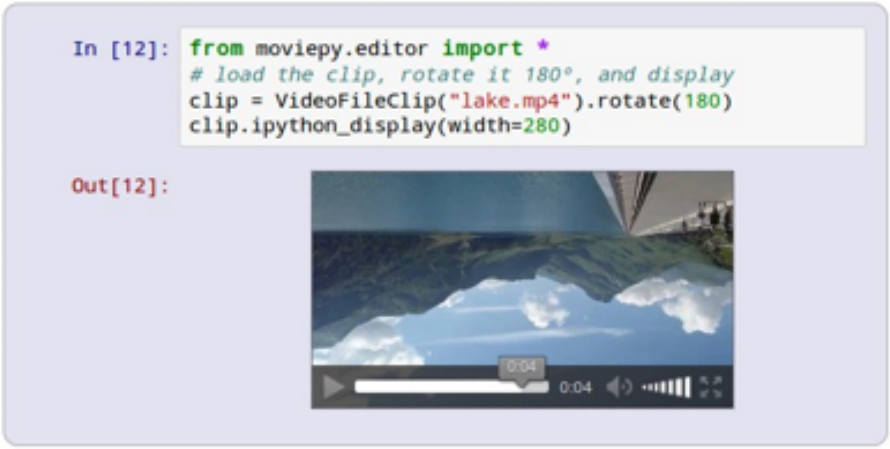

Abb. 11: Beispiel Code von *moviepy* (Zulko 2017)

3.3 Code Ausführung VideoToText

Ein Objekt der Klasse *VideoToText*, die 6 Parameter erwartet, wird erstellt. Die Parameter sind Folgenden:

1. **video-source**
   Das ist ein Pflichtparameter und erwartet der Videolink, von dem das text extrahiert werden soll.

2. **audio_formate='wav'**
   Das ist standardmäßig auf 'wav' gesetzt, da diesem Format für die Textextraktion beste Ergebnisse liefert.

3. **language='de-DE'**
   In erste Linie ist die deutsche Sprache vorgesehen, deshalb ist standardmäßig auf 'de-DE' gesetzt.

[6]https://pypi.org/project/moviepy
[7]https://github.com/jiaaro/pydub/blob/master/API.markdown
[8]https://pypi.org/project/SpeechRecognition/
[9]https://docs.python.org/3/library/multiprocessing.html

4. **min_silence_len=500**
   Durch diese Parameter kann man die Länge der Pause(Silence) steuern, also alles, was leiser als dieser Wert ist, wird als Stille betrachtet. Das ist standardmäßig auf 500 Millisekunden gesetzt.

5. **silence_thresh=-36**
   Durch diese Parameter kann man Silence Schwelle anpassen. Alles, was leiser als dieser Wert ist, wird als Stille betrachtet. Auf Standardeinstellung ist auf -36 dBFS.

6. **keep_silence=400**
   Durch diese Parameter wird etwas Stille am Anfang und am Ende der Chunks gelassen. Auf Standardeinstellung ist auf 400ms.

In der Klasse *main.py* ist ein Beispiel Methode: *video_to_text()* definiert. Mit dem Aufruf von *main.py* kann man diese Methode aufrufen. In dieser Methode wird eine Instanz mit der bestimmten Parameter der Klasse *VideoToText* erstellt. Danach wird die Videodatei in Audiodatei umgewandelt, die Audiodatei wird gesplittet, dass Text wird von alle Chunks parallel extrahiert, zum Schluss werden alle Audiodateien gelöscht.

```python
def video_to_text(video_link):

    video_text = VideoToText(video_link)
    try:
        video_text.video_to_audio()
        video_text.split_audio_file_on_silence()
        video_text.read_text_parallel()
    except Exception as e:
        print("The following Error occurred while converting speech to text:", e)
    finally:
        video_text.delete_directory()
```

Abb. 12: Ausschnitt aus *main.py* der Methode *video_to_text()*

3.4 Speicher Struktur VideoToText

Extrahierte Video-Links werden von News Sites Extraktion in einer Textdatei gespeichert. Name der Datei, wo die Links gespeichert sind, hat folgende Form:

- Präfix: Name von der Webseite mit den Videolinks
- Suffix: _VideoLinks.txt

zdf__denachrichtenpolitikatomkraftwerk-saporischschja-krim-ukraine-krieg-russland-100__html_VideoLinks.text

```
1  https://nrodlzdf-a.akamaihd.net/none/zdf/22/08/220810_jaeger_hie/1/220810_jaeger_hie_2128k_p18v15.webm
2  https://nrodlzdf-a.akamaihd.net/none/zdf/22/08/220809_saporischschja_xpr/1/220809_saporischschja_xpr_2128k_p18v15.webm
3  https://nrodlzdf-a.akamaihd.net/none/zdf/22/08/220808_gesamt_hli/1/220808_gesamt_hli_2128k_p18v15.webm
4
```

Abb. 13: Ausschnitt einer Textdatei mit Video Links

Der extrahierte Text wird in einer Textdatei gespeichert. Name der Datei, wo das Text gespeichert wird, hat folgende Form:

- Präfix: Name der Video Link
- Suffix: _VideoScript.txt

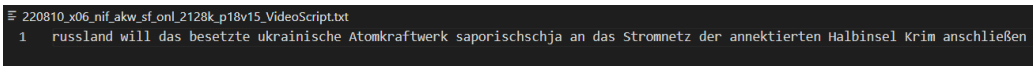

Abb. 14: Ausschnitt einer Textdatei

4 Open Issues Video To Text Extraktion

Die Videos vom ARD im Bereich vom News lassen nicht in Audiodatei umwandeln.

Abb. 15: Fehlermeldung bei Umwandlung einer ARD Video nach Audio

5 Mögliche Lösungen

Videos zuerst runterladen und dann bearbeiten, aber hier muss man schauen, ob das rechtlich erlaubt ist. Videos mithören, dann die Audiodatei bearbeiten, aber da die Hürde ist, das Laufzeit.

6 Literaturverzeichnis

[1] Mike Bostock. *D3.js.* 2021. URL: https://d3js.org/.

[2] Unicode Inc. *Full Emoji List, v14.0.* 2022. URL: https://unicode.org/emoji/charts/full-emoji-list.html.

[3] Telegram FZ LLC and Telegram Messenger Inc. *Bots: An introduction for developers.* 2022. URL: https://core.telegram.org/bots.

[4] Telegram FZ LLC and Telegram Messenger Inc. *Telegram.* 2022. URL: https://telegram.org.

[5] Lonami. *Telethon Documentation (Release 1.24.0).* 2022. URL: https://buildmedia.readthedocs.org/media/pdf/telethon/stable/telethon.pdf.

[6] Alexander Mevlüt Bagci Mehler. *Arbeitspakete kurze Einführung.* PowerPoint Präsentation. 2022.

[7] Wedia. *Data Visualization: Zahlen und Bilder.* 2022. URL: https://www.wedia-group.com/de/library/data-visualisation-tool-%09analyse-entscheidung/#:~:text=Zweck%20der%20Data%20Visualization%20ist,wird%2C%20erleichtert%20das%20content%20scoring.

7 Anhang

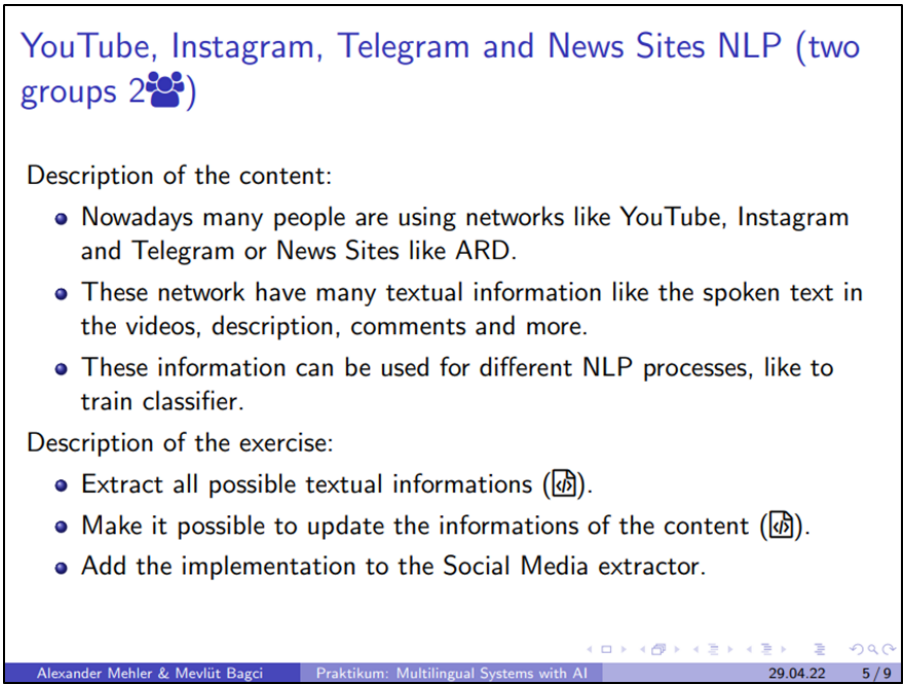

Abb. 16: Aufgabenbeschreibung [6, p.5]