Sonstige Beteiligung

Verwendeter Code zur Analyse von Artikeln des Nachrichtensenders *n-tv*

Kai Herbst, Manuel Zeh, Henrik Popp

August 2024

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des Moduls *Analyse semi- & unstrukturierter Daten* wurden die von *n-tv* veröffentlichten Artikel im Zeitraum des 17.04.2024 - 17.07.2024 gesammelt und nachfolgend analysiert. Das vorliegende Dokument enthält den Code zusammen mit dazugehörigen kurzen Beschreibungen, der zur Sammlung, Verarbeitung und Analyse der Daten verwendet wurde.

Der Code ist in die Abschnitte Datenerhebung, Datenvorbereitung, Upload-Zeiten-Analyse, Sentiment-Analyse und Topic Modeling unterteilt.

Jeglicher Code lässt sich ebenfalls in Form von Jupyter Notebooks im GitHub-Repository n-tv Datenanalyse finden.

2 Datenerhebung

Die folgenden Unterkapitel dokumentieren den Code, der zur Sammlung, Speicherung und Vorverarbeitung für die späteren Analysen verwendet wurde.

2.1 Sammelung und Speicherung der Nachrichtenartikel

Abbildung 1 zeigt den grundlegenden Ablauf, durch den die Nachrichtenartikel gespeichert werden. Zunächst wurde in über *AWS Eventbridge* ein Scheduler definiert, der täglich um 00:00 Uhr Lambda-Funktion startet, die wiederum auf die von *n-tv* angebotenen RSS-Schnittstellen zugreift. In den jeweiligen RSS-Feeds für jede Kategorie werden die zuletzt veröffentlichten Artikel mit Links und weiteren Metadaten aufgelistet. Über die Links werden die Artikel anschließend heruntergeladen und in einem *AWS S3-Bucket* gespeichert.

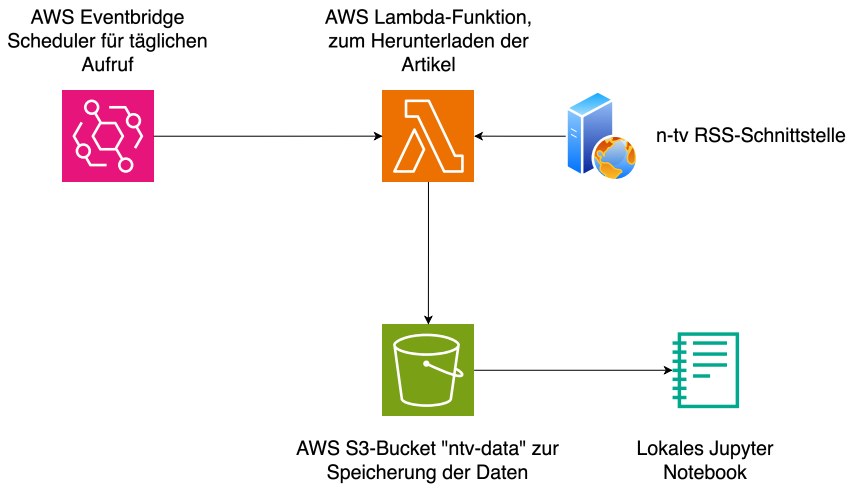

Abbildung 1: Datenpipeline zur Sammeln der *ntv*-Artikel

Die Lambda-Funktion zum Sammeln der Artikel besteht aus vier Dateien. Nachfolgend zu sehen ist die Hauptdatei mit der Einstiegsfunktion:

Listing 1: index.mjs

```
1 import helper from "./helper.mjs";
2 import categories from "./categories.mjs";
3 import S3Uploader from "./s3Helper.mjs";
4
5 export const handler = async (event, context) => {
6     const s3Uploader = new S3Uploader();
7     const bucket = "ntv-data";
8     const baseKey = helper.getYesterdaysDate().toISOString().split("T")[0];
9     for (let category of categories) {
```

```
10          let { links, rss } = await
                helper.getLinksOfYesterdaysArticelsAndRssObjectOf(category);
11          if (links.length == 0) {
12              continue;
13          }
14          let rssKey = baseKey + "/" + category + "/rss.json";
15          let rssJSONString = JSON.stringify(rss);
16          await s3Uploader.uploadContent(bucket, rssKey, rssJSONString);
17          for (let link of links) {
18              let articleKey = baseKey + "/" + category + "/" +
                    helper.getFileNameFromLink(link);
19              let htmlContent = await helper.downloadContentFrom(link);
20              let cleanedHTML = helper.cleanHTMLContent(htmlContent);
21              await s3Uploader.uploadContent(bucket, articleKey, cleanedHTML);
22          }
23      }
24      return {
25          statusCode: 200,
26          body: JSON.stringify("Done!"),
27      };
28  };
```

Die Hauptfunktion (async handler) verwendet wiederum die in den anderen drei Dateien *helper.mjs*, *categories.mjs* und *s3Helper.mjs* definierten Funktionen auf, die nachfolgend aufgelistet werden.

In der *categories.mjs* werden die abzurufenden Kategorien definiert:

Listing 2: categories.mjs

```
1  export default [
2      "dasbeste",
3      "politik",
4      "politik/US-wahl-2024",
5      "wirtschaft",
6      "boersenkurse",
7      "sport",
8      "panorama",
9      "leute",
10     "technik",
11     "ratgeber",
12     "wissen",
13     "auto",
14     "infografik",
```

```
    "regionales",
    "der_tag/alle_tage",
    "meinung",
];
```

Die *s3Helper.mjs*-Datei stellt Funktionen für den Zugriff auf die entsprechenden S3-Buckets bereit:

Listing 3: s3Helper.mjs

```
import { S3Client, PutObjectCommand } from "@aws-sdk/client-s3";

export default class S3Uploader {
    constructor() {
        this.s3Client = new S3Client({ region: "eu-central-1" });
    }

    async uploadContent(bucket, key, contentString) {
        const command = new PutObjectCommand({
            Bucket: bucket,
            Key: key,
            Body: contentString,
        });

        try {
            console.log("Uploading", bucket + "/" + key, "...");
            const response = await this.s3Client.send(command);
            console.log("Finished!");
        } catch (err) {
            console.log(err);
        }
    }
}
```

In der *helper.mjs*-Datei werden schließlich noch Hilfsfunktionen definiert. Unter anderem Funktionen, die einen Großteil des HTML-Codes aus den Artikeln bereits im Vorfeld entfernt, um Speicherplatz zu sparen. Dazu zählen beispielsweise alle *Script*-Tags, da diese für die weitere Bearbeitung nicht benötigt werden. Durch diese Vorgehensweise wird viel Speicherplatz und damit auch Kosten eingespart.

Listing 4: helper.mjs

```
import { XMLParser } from "fast-xml-parser";

```

```
3  function buildRSSLinkFromCategory(category) {
4      let domain = "https://www.n-tv.de/";
5      let postfix = "/rss";
6      return domain + category + postfix;
7  }
8
9  function getYesterdaysDate() {
10     const date = new Date();
11     date.setHours(date.getHours() + 2);
12     date.setDate(date.getDate() - 1);
13     return date;
14 }
15
16 function dateWasYesterday(datestring) {
17     const yesterdaysDate = getYesterdaysDate();
18     const parsedDate = new Date(datestring);
19     return (
20         yesterdaysDate.getFullYear() == parsedDate.getFullYear() &&
21         yesterdaysDate.getMonth() == parsedDate.getMonth() &&
22         yesterdaysDate.getDate() == parsedDate.getDate()
23     );
24 }
25
26 function parseXMLString(xmlstring) {
27     const parser = new XMLParser();
28     return parser.parse(xmlstring);
29 }
30
31 function getRssItemsOfYesterday(rssObject) {
32     if (rssObject.rss.channel.item == undefined) {
33         return [];
34     }
35     let items = rssObject.rss.channel.item;
36     return items.filter((item) => dateWasYesterday(item.pubDate));
37 }
38
39 async function downloadRssXMLOfCategory(category) {
40     let url = buildRSSLinkFromCategory(category);
41     return await downloadContentFrom(url);
42 }
43
44 async function downloadContentFrom(url) {
45     let response = await fetch(url);
```

```
46      return await response.text();
47  }
48
49  function getFileNameFromLink(link) {
50      return link.split("/").at(-1);
51  }
52
53  function getLinksOfRssItems(rssItems) {
54      let links = [];
55      for (let item of rssItems) {
56          links.push(item.link);
57      }
58      return links;
59  }
60
61  function cleanHTMLContent(htmlString) {
62      let cleaned = htmlString.replace(
63          /<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script\s*>/gi, "");
64      cleaned = cleaned.replace(
65          /<nav\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/nav\s*>/gi, "");
66      cleaned = cleaned.replace(
67          /<style\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/style\s*>/gi, "");
68      cleaned =
69          cleaned.replace(/<link\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/link\s*>/gi,
              "");
70      return cleaned;
71  }
72
73  async function getLinksOfYesterdaysArticelsAndRssObjectOf(category) {
74      let rssXMLString = await downloadRssXMLOfCategory(category);
75      let rss = parseXMLString(rssXMLString);
76      let yesterdaysItems = getRssItemsOfYesterday(rss);
77      let links = getLinksOfRssItems(yesterdaysItems);
78      rss.rss.channel.item = yesterdaysItems;
79      rss.rss.channel.category = category;
80      return { links: links, rss: rss };
81  }
82
83  export default {
84      getYesterdaysDate,
85      getLinksOfYesterdaysArticelsAndRssObjectOf,
86      getFileNameFromLink,
87      downloadContentFrom,
```

```
88      cleanHTMLContent,
89  };
```

2.2 Datenvorbereitung

Dieser Code dient dem Erstellen einer CSV-Datei, die anschließend als Grundlage für das weitere Vorgehen verwendet wird. Die Funktionen der einzelnen Methoden sollten dabei aus den Funktionsnamen hervorgehen. Grundsätzlich wird über die Ordner und Dateien des AWS lokal heruntergeladenen Buckets iteriert und die Artikel gesammelt. Mit Hilfe von *BeautifulSoup* werden Datum und Artikeltext extrahiert und gespeichert. Anschließend werden die gesammelten Daten in eine CSV-Datei geschrieben. Die CSV-Datei enthält die Spalten date (Datum), time (Uhrzeit), category (Kategorie), headline (Überschrift), filename (Dateiname) und text (Artikeltext).

Listing 5: Datengenerierung

```
1  import csv
2  import os
3  import re
4  from datetime import datetime
5  from typing import Dict, List
6
7  from bs4 import BeautifulSoup
8
9
10 def get_content_of(article_path: str) -> str:
11     result = ""
12     with open(article_path, "r", encoding="utf-8") as article:
13         result = article.read()
14     if result != "":
15         return result
16     raise Exception("File not found")
17
18
19 def extract_text_from_html_article(html_text: str) -> str:
20     soup = BeautifulSoup(html_text, "html.parser")
21     article_text = soup.find("div", class_="article__text")
22     if article_text:
23         soup = BeautifulSoup(article_text.text, "html.parser")
24         for aside in soup.find_all("div", class_="article__aside"):
25             aside.decompose()
26         output = soup.text.strip()
```

```
27          output = re.sub(" +", " ", output)
28          output = output.replace("\n", "")
29          return output
30      raise Exception("No article text was found")
31
32
33  def extract_exact_date_from_html_article(html_text: str) -> str | None:
34      soup = BeautifulSoup(html_text, "html.parser")
35      date = soup.find("span", class_="article__date")
36      if date:
37          output = date.get_text()
38          return output
39      print("No date was found")
40      return None
41
42
43  def convert_filename_to_article_headline(article_filename: str) -> str:
44      headline = (" ").join(article_filename.split("-")[:-1])
45      return headline
46
47
48  def parse_ntv_datetime(date_string: str):
49      try:
50          datetime_obj = datetime.strptime(date_string, "%d.%m.%Y, %H:%M Uhr")
51          date = datetime_obj.date()
52          time = datetime_obj.time()
53          return date, time
54      except:
55          return None, None
56
57
58  def collect_article_contents_from(path: str) -> List[Dict[str, str]]:
59      collection = []
60      dates = os.listdir(path)
61      count_articles = 0
62      print()
63      for date in dates:
64          categories = os.listdir(f"{path}/{date}")
65          for category in categories:
66              article_list = os.listdir(f"{path}/{date}/{category}")
67              for article in article_list:
68                  if article != "rss.json" and article != ".ipynb_checkpoints":
69                      count_articles += 1
```

```
70                      print("\r", end="")
71                      print(f"Scanned articles: {count_articles} - scanning:
                            {article}", end="")
72                      article_content =
                            get_content_of(f"{path}/{date}/{category}/{article}")
73                      article_text = extract_text_from_html_article(article_content)
74                      article_extracted_date =
                            extract_exact_date_from_html_article(article_content)
75                      article_date, article_time =
                            parse_ntv_datetime(article_extracted_date)
76                      collection.append(
77                          {
78                              "date": article_date,
79                              "time": article_time,
80                              "category": category,
81                              "headline":
                                    convert_filename_to_article_headline(article),
82                              "filename": article,
83                              "text": article_text,
84                          }
85                      )
86      return collection
87
88
89  def main():
90      content = collect_article_contents_from("ntv-data")
91      with open("2-1-data.csv", "w", encoding="utf-8", newline="") as output_file:
92          fc = csv.DictWriter(output_file, fieldnames=content[0].keys())
93          fc.writeheader()
94          fc.writerows(content)
95
96
97  if __name__ == "__main__":
98      main()
```

3 Analysevorbereitung

Bevor die Daten auf einzelne Aspekte hin analysiert werden, müssen die Daten noch weiter aufbereitet und die Sentiment-Analyse und Stichwortgenerierung durchgeführt werden.

3.1 Vorbereitung der Upload-Analyse

Für die Upload-Analyse werden die CSV-Datei um die Spalten upload-hour (Stunde des Uploads), weekday (Wochentag) und length (Artikellänge) erweitert.

Listing 6: Vorbereitung zur Uploadanalyse

```
1  import pandas as pd
2
3
4  def main():
5      articles_data = pd.read_csv("2-1-data.csv")
6      articles = pd.DataFrame(articles_data)
7      articles = articles.drop("filename", axis=1)
8      articles.head()
9
10     articles["upload"] = pd.to_datetime(articles["time"] + " " +
           articles["date"])
11     articles = articles.dropna()
12     articles.head()
13
14     articles["upload-hour"] = articles["upload"].dt.hour
15     articles["upload-hour"] = articles["upload-hour"].astype(int)
16     articles.head()
17
18     articles["weekday"] = articles["upload"].dt.dayofweek
19     wochentag_map = {0: "Monday", 1: "Tuesday", 2: "Wednesday", 3: "Thursday",
           4: "Friday", 5: "Saturday", 6: "Sunday"}
20     articles["weekday"] = articles["weekday"].map(wochentag_map)
21     articles.head()
22
23     articles["length"] = articles["text"].apply(len)
24     articles.head()
25
26     articles.to_csv("2-2-data-extended.csv", index=False)
27
28
29 if __name__ == "__main__":
30     main()
```

3.2 Vorbereitung zur Themenanalyse

Für den ersten Ansatz der Themenanalyse müssen Stichwörter zu jedem Artikel generiert werden. Dies geschieht mit Hilfe des OpenAI-Sprachmodells *GPT-4o mini*, auf das über die OpenAI-REST-API zugegriffen wird und das zu jedem Artikel zehn Stichwörter generiert. Die Stichwörter werden anschließend der CSV-Datei hinzugefügt:

Listing 7: Vorbereitung zur Stichwortanalyse

```
import pandas as pd
from openai import OpenAI


def create_keywords_for_articles(openai_client, articles_df, start=0, end=5):
    print()
    for i in range(start, end):
        article_text = articles_df.iloc[i]["text"]
        completion = openai_client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[
                {
                    "role": "system",
                    "content": "Du fasst Nachrichtenartikel des
                        Nachrichtensenders NTV anhand von Stichpunkten
                        zusammenzufassen. Du gibst ausschliesslich die
                        Stichwoerter durch Kommata getrennt zurueck. Die
                        Stichwoerter sollen den Inhalt mit uebergeordneten
                        Kategorien wie Politik, Wirtschaft, Gesellschaft, Auto,
                        Sport, Unterhaltung und dergleichen beschreiben. Es
                        sollen exakt 10 Stichwoerter generiert werden.",
                },
                {
                    "role": "user",
                    "content": article_text,
                },
            ],
        )
        keywords = completion.choices[0].message.content
        articles_df.at[i, "keywords"] = keywords
        print("\r", end="")
        print(f"Generated Keywords for {i + 1} articles", end="")
    return articles_df


```

```
29 def main():
30     openai_client = OpenAI()
31     articles_data = pd.read_csv("2-2-data-extended.csv")
32     articles = pd.DataFrame(articles_data)
33     articles["keywords"] = None
34     articles = create_keywords_for_articles(openai_client, articles, 0, 10678)
35     articles.to_csv("2-3-data-extended-keywords.csv", index=False)
36
37
38 if __name__ == "__main__":
39     main()
```

3.3 Durchführung Sentiment-Analyse

Um die Sentiments der Artikel genauer untersuchen zu können, müssen diese zunächst identifiziert werden. Dazu wird die CSV-Datei folgende Spalten erweitert:

- sentiment_headline
- sentiment_text
- sentiment_prop_headline_positive
- sentiment_prop_headline_neutral
- sentiment_prop_headline_negative
- sentiment_prop_text_positive
- sentiment_prop_text_neutral
- sentiment_prop_text_negative

Listing 8: Vorbereitung zur Sentiment-Untersuchung

```
1  import pandas as pd
2  from germansentiment import SentimentModel
3
4  sentiment_model = SentimentModel()
5
6
7  def sentiment_on_texts(articles_df):
8      length = len(articles_df)
9      sentiments = []
10     for index, row in articles_df.iterrows():
11         print(f"\rRunning sentiment analysis on article {index + 1} of
```

```
            {length}", end="")
12          text_class, text_probabilities =
                sentiment_model.predict_sentiment([row["text"]],
                output_probabilities=True)
13          sentiments.append((text_class, text_probabilities))
14      return sentiments
15
16
17  def main():
18      articles_data = pd.read_csv("2-3-data-extended-keywords.csv")
19      articles = pd.DataFrame(articles_data)
20
21      headlines = articles["headline"]
22      headlines_classes, headlines_probabilities =
            sentiment_model.predict_sentiment(headlines, output_probabilities=True)
23
24      articles = articles.assign(sentiment_headline=headlines_classes)
25      articles = articles.assign(sentiment_prob_headline=headlines_probabilities)
26
27      text_sentiments = sentiment_on_texts(articles)
28      texts_classes = [x[0][0] for x in text_sentiments]
29      texts_probabilities = [x[1][0] for x in text_sentiments]
30      articles["sentiment_text"] = texts_classes
31      articles["sentiment_prob_text"] = texts_probabilities
32      articles["sentiment_prob_headline_positive"] = articles.apply(
33          lambda row: row["sentiment_prob_headline"][0][1], axis=1
34      )
35      articles["sentiment_prob_headline_negative"] = articles.apply(
36          lambda row: row["sentiment_prob_headline"][1][1], axis=1
37      )
38      articles["sentiment_prob_headline_neutral"] = articles.apply(
39          lambda row: row["sentiment_prob_headline"][2][1], axis=1
40      )
41      articles["sentiment_prob_text_positive"] = articles.apply(lambda row:
            row["sentiment_prob_text"][0][1], axis=1)
42      articles["sentiment_prob_text_negative"] = articles.apply(lambda row:
            row["sentiment_prob_text"][1][1], axis=1)
43      articles["sentiment_prob_text_neutral"] = articles.apply(lambda row:
            row["sentiment_prob_text"][2][1], axis=1)
44      articles = articles.drop("sentiment_prob_headline", axis=1)
45      articles = articles.drop("sentiment_prob_text", axis=1)
46      articles.to_csv("3-4-data-extended-keywords-sentiment.csv", sep=",",
            encoding="utf-8", index=False)
```

```
47
48
49 if __name__ == "__main__":
50     main()
```

4 Analyse der Daten

4.1 Analyse des Uploadverhaltens

Im nachfolgenden sind die Funktionen, die zur Erstellung der Diagramme verwendet wurden. Diese sollten in einzelnen Zellen in einem Jupyter-Notebook ausgeführt werden, um die Ergebnisse direkt sehen zu können.

Listing 9: Analyse der Uploadzeiten

```
1  import pandas as pd
2  import seaborn as sns
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
7  articles = pd.DataFrame(articles_data)
8
9  def create_line_chart_for_uploads_per_hour(articles):
10     uploads_per_hour = articles['upload-hour'].value_counts().sort_index()
11     plt.figure(figsize=(12, 6))
12     plt.plot(uploads_per_hour.index, uploads_per_hour.values, marker='o',
           linestyle='-', color='b')
13     plt.title('Uploads per hour')
14     plt.xlabel('Hour of day')
15     plt.ylabel('Uploads')
16     plt.xticks(range(24))
17     plt.grid(True)
18     plt.show()
19 create_line_chart_for_uploads_per_hour(articles)
20
21
22 def create_line_chart_for_uploads_per_weekday(articles):
23     uploads_per_weekday = articles['weekday'].value_counts().reindex(['Monday',
           'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
           fill_value=0)
24     plt.figure(figsize=(12, 6))
```

```
25      plt.plot(uploads_per_weekday.index, uploads_per_weekday.values, marker='o',
            linestyle='-', color='b')
26      plt.title('Anzahl der Uploads pro Wochentag')
27      plt.xlabel('Wochentag')
28      plt.ylabel('Anzahl der Uploads')
29      plt.grid(True)
30      plt.show()
31  create_line_chart_for_uploads_per_weekday(articles)
32
33
34  def create_heatmap_of_hour_per_weekday(articles):
35      heatmap_data = articles.groupby(['weekday', 'upload-hour'],
            observed=False).size().unstack(fill_value=0).reindex([
36          'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
                'Sunday'
37      ], fill_value=0)
38      plt.figure(figsize=(12, 5))
39      sns.heatmap(heatmap_data, annot=True, fmt='d', cbar=False, rasterized=True,
            cmap="crest")
40      plt.title('Uploads per weekday and hour')
41      plt.xlabel('Hour')
42      plt.ylabel('Weekday')
43      plt.show()
44  create_heatmap_of_hour_per_weekday(articles)
45
46  def create_bar_chart_for_median_by_hour_and_by_weekday(articles):
47      weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
            'Saturday', 'Sunday']
48      articles['weekday'] = pd.Categorical(articles['weekday'],
            categories=weekday_order, ordered=True)
49      median_by_hour = articles.groupby('upload-hour')['length'].median()
50      median_by_weekday = articles.groupby('weekday',
            observed=False)['length'].median()
51      median_by_weekday_hour = articles.groupby(['weekday', 'upload-hour'],
            observed=False)['length'].median().unstack()
52      fig, axs = plt.subplots(2, 1, figsize=(10, 8))
53      axs[0].bar(median_by_hour.index, median_by_hour.values)
54      axs[0].set_title('Median length by hour of upload')
55      axs[0].set_xlabel('Upload hour')
56      axs[0].set_ylabel('Median length')
57      axs[1].bar(median_by_weekday.index, median_by_weekday.values)
58      axs[1].set_title('Median length by weekday')
59      axs[1].set_xlabel('Weekday')
```

```
60      axs[1].set_ylabel('Median length')
61      axs[1].set_xticks(range(len(weekday_order)))
62      axs[1].set_xticklabels(weekday_order)
63      plt.tight_layout()
64      plt.show()
65  create_bar_chart_for_median_by_hour_and_by_weekday(articles)
66
67  def create_heatmap_of_article_length_by_hour_and_weekday(articles):
68      median_data = articles.groupby(['upload-hour', 'weekday'],
            observed=False)['length'].median().reset_index()
69      pivot_data = median_data.pivot_table(index='weekday',
            columns='upload-hour', values='length', aggfunc='median', observed=False)
70      plt.figure(figsize=(22, 10))
71      sns.heatmap(pivot_data, annot=True, rasterized=True, cmap="crest",
            fmt='.1f')
72      plt.title('Median of length after upload hour and weekday')
73      plt.xlabel('Hour')
74      plt.ylabel('Weekday')
75      plt.show()
76  create_heatmap_of_article_length_by_hour_and_weekday(articles)
77
78  def create_heatmap_plots_for_hours_per_category(articles):
79      upload_counts = articles.groupby(['category',
            'upload-hour']).size().unstack(fill_value=0)
80      for category in upload_counts.index:
81          plt.figure(figsize=(24, 1))
82          sns.heatmap(upload_counts.loc[[category]], annot=True, fmt='d',
                cbar=False, rasterized=True, cmap="crest")
83          plt.title(f'Uploads per hour for category: {category}')
84          plt.xlabel('hour')
85          plt.yticks([])
86          plt.show()
87  create_heatmap_plots_for_hours_per_category(articles)
88
89  def create_heatmap_plots_for_articles_per_hour_unified(articles):
90      heatmap_data = articles.groupby(['category',
            'upload-hour']).size().unstack(fill_value=0)
91      scaled_heatmap_data = heatmap_data.div(heatmap_data.sum(axis=1), axis=0) *
            100
92      plt.figure(figsize=(10, 6))
93      sns.heatmap(scaled_heatmap_data, annot=False, cbar=False, rasterized=True,
            cmap="crest")
94      plt.title('Uploads by category and hour')
```

```
95      plt.xlabel('Hour')
96      plt.ylabel('Category')
97      plt.show()
98  create_heatmap_plots_for_articles_per_hour_unified(articles)
99
100 def create_bar_plots_for_articles_per_weekday(articles):
101     upload_weekday_counts = articles.groupby(['category', 'weekday'],
            observed=False).size().unstack(fill_value=0)
102     upload_weekday_counts = upload_weekday_counts.reindex(columns=['Monday',
            'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
103     for category in upload_weekday_counts.index:
104         plt.figure(figsize=(10, 2))
105         sns.barplot(x=upload_weekday_counts.columns,
                y=upload_weekday_counts.loc[category].values)
106         plt.title(f'Upload behavior category: {category}')
107         plt.xlabel('Wochentag')
108         plt.ylabel('Anzahl der Artikel')
109         plt.xticks(rotation=45)
110         plt.show()
111 create_bar_plots_for_articles_per_weekday(articles)
112
113 def create_heatmap_plots_for_articles_per_weekday_unified(articles):
114     heatmap_data = articles.groupby(['category', 'weekday'],
            observed=False).size().unstack(fill_value=0)
115     scaled_heatmap_data = heatmap_data.div(heatmap_data.sum(axis=1), axis=0) *
            100
116     plt.figure(figsize=(10, 6))
117     sns.heatmap(scaled_heatmap_data, annot=False, cbar=False, rasterized=True,
            cmap="crest")
118     plt.title('Uploads per category and hour')
119     plt.xlabel('Weekday')
120     plt.ylabel('Category')
121     plt.show()
122 create_heatmap_plots_for_articles_per_weekday_unified(articles)
123
124 def create_heatmap_plots_for_articles_per_hour_per_weekday(articles):
125     categories = articles['category'].unique()
126     weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
            'Saturday', 'Sunday']
127     hours = np.arange(0, 24)
128     index = pd.MultiIndex.from_product([categories, weekdays, hours],
            names=['Category', 'Weekday', 'Hour'])
```

```
129     upload_category_weekday_hour_counts = articles.groupby(['category',
            'weekday', 'upload-hour'], observed=False).size().reindex(index,
            fill_value=0).unstack(level='Hour')
130     upload_category_weekday_hour_counts =
            upload_category_weekday_hour_counts.reindex(weekdays, level='Weekday')
131     for category in categories:
132         plt.figure(figsize=(14, 4))
133         sns.heatmap(upload_category_weekday_hour_counts.loc[category],
                annot=True, fmt='d', cbar=False, rasterized=True, cmap="crest")
134         plt.title(f'Upload behavior for category: {category}')
135         plt.xlabel('hour of day')
136         plt.ylabel('weekday')
137         plt.show()
138 create_heatmap_plots_for_articles_per_hour_per_weekday(articles)
139
140 def create_bar_chart_for_length_per_category(articles):
141     median_by_category =
            articles.groupby('category')['length'].median().reset_index()
142     plt.figure(figsize=(18, 6))
143     sns.barplot(x='category', y='length', data=median_by_category)
144     plt.title('Median length of category')
145     plt.xlabel('Category')
146     plt.ylabel('Median')
147     plt.show()
148 create_bar_chart_for_length_per_category(articles)
149
150 def create_heatmap_for_length_per_category_by_hour_and_weekday(articles):
151     categories = articles['category'].unique()
152     for category in categories:
153         category_data = articles[articles['category'] == category]
154         median_data = category_data.groupby(['upload-hour', 'weekday'],
                observed=False)['length'].median().reset_index()
155         pivot_data = median_data.pivot_table(index='weekday',
                columns='upload-hour', values='length', aggfunc='median',
                observed=False)
156         plt.figure(figsize=(23, 5))
157         sns.heatmap(pivot_data, annot=True, rasterized=True, cmap="crest",
                fmt='.0f')
158         plt.title(f'Median of length for category "{category}" by upload hour
                and weekday')
159         plt.xlabel('Hour')
160         plt.ylabel('Weekday')
161         plt.show()
```

```
create_heatmap_for_length_per_category_by_hour_and_weekday(articles)

def create_bar_charts_for_length_of_hour_per_weekday_per_category(articles):
    categories = articles['category'].unique()
    for category in categories:
        category_data = articles[articles['category'] == category]
        median_by_hour = category_data.groupby('upload-hour',
            observed=False)['length'].median().reset_index()
        plt.figure(figsize=(10, 4))
        sns.barplot(x='upload-hour', y='length', data=median_by_hour)
        plt.title(f'Median of length per hour and category {category}')
        plt.xlabel('upload hour')
        plt.ylabel('Median of length')
        plt.xticks(rotation=45)
        plt.show()
create_bar_charts_for_length_of_hour_per_weekday_per_category(articles)
```

4.2 Analyse der Sentiments

Der im Anschluss aufgeführte Code enthält die Funktionen zur Generierung von Visualisierungen zu den Sentiments. Auch diese sollten vorzugsweise in einem Jupyter Notebook in einzelnen Zellen ausgeführt werden, um die Diagramme sehen zu können.

Listing 10: Analyse der Sentiments

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
articles = pd.DataFrame(articles_data)
articles.head()

def create_sentiment_result_plot(df, column):
    sentiment_count = []
    for sentiment_class, count in df[column].value_counts().items():
        sentiment_count.append( (sentiment_class, count) )
    labels, values = zip(*sentiment_count)
    plt.xlabel('Sentiment class')
    plt.ylabel('Count')
    plt.title(f'Sentiment analyse of {column}')
    plt.bar(labels, values)
```

```
18      for i, value in enumerate(values):
19          print(i, value)
20          plt.text(i, value + 0.1, str(value), ha='center', va='bottom')
21      plt.show()
22  create_sentiment_result_plot(articles, "sentiment_headline")
23
24  create_sentiment_result_plot(articles, "sentiment_text")
25
26  def create_category_result_plot(df, column, relative=False):
27      categories = []
28      category_count = []
29      negatives = []
30      neutrals = []
31      positives = []
32      for category, count in articles['category'].value_counts().items():
33          negatives.append(articles[column][(articles.sentiment_text ==
                'negative') & (articles.category == category)].count())
34          neutrals.append(articles[column][(articles.sentiment_text == 'neutral')
                & (articles.category == category)].count())
35          positives.append(articles[column][(articles.sentiment_text ==
                'positive') & (articles.category == category)].count())
36          categories.append(category)
37      if relative:
38          for i, category in enumerate(categories):
39              count = articles[column][(articles.category == category)].count()
40              negatives[i] = negatives[i] / count
41              positives[i] = positives[i] / count
42              neutrals[i] = neutrals[i] / count
43      ind = np.arange(len(categories))
44      fig, ax = plt.subplots()
45      p1 = ax.bar(ind, positives, label='Positiv', color='seagreen')
46      p2 = ax.bar(ind, neutrals, bottom=positives, label='Neutral',
            color='cornflowerblue')
47      p3 = ax.bar(ind, negatives, bottom=[i+j for i,j in zip(positives,
            neutrals)], label='Negativ', color='lightcoral')
48      ax.set_xlabel('Kategorien')
49      ax.set_ylabel('Anteile')
50      ax.set_xticks(ind)
51      ax.set_xticklabels(categories)
52      plt.xticks(rotation=90)
53      ax.legend()
54      plt.tight_layout()
55      plt.show()
```

```
create_category_result_plot(articles, 'sentiment_headline', relative=True)

create_category_result_plot(articles, 'sentiment_text', relative = True)

def create_bar_chart_for_count_of_negative_articles_per_weekday(articles):
    negative_articles = articles[articles["sentiment_text"] == "negative"]
    negative_count_by_category = negative_articles.groupby("category").size()
    negative_count_by_category.plot(kind="bar", color="red", alpha=0.7)
    plt.title('Count of negative articles per category')
    plt.xlabel('category')
    plt.ylabel('count')
    plt.xticks(rotation=45)
    plt.grid(axis='y')
    plt.tight_layout()
    plt.show()
create_bar_chart_for_count_of_negative_articles_per_weekday(articles)

def create_bar_chart_for_percentage_of_negative_articles_per_weekday(articles):
    grouped =
        articles.groupby("weekday")["sentiment_text"].value_counts(normalize=True).unstack()
    grouped["negative_share"] = grouped["negative"] * 100
    grouped["negative_share"].plot(kind="bar", color="red", alpha=0.6)
    plt.title('Percentage of negative articles per weekday')
    plt.xlabel('weekday')
    plt.ylabel('percentage (%)')
    plt.xticks(rotation=45)
    plt.ylim(0, 100)
    plt.grid(axis='y')
    plt.tight_layout()
    plt.show()
create_bar_chart_for_percentage_of_negative_articles_per_weekday(articles)

def create_bar_chart_for_percentage_of_neutral_articles_per_weekday(articles):
    grouped =
        articles.groupby("weekday")["sentiment_text"].value_counts(normalize=True).unstack()
    grouped["neutral_share"] = grouped["neutral"] * 100
    grouped["neutral_share"].plot(kind="bar", color="blue", alpha=0.6)
    plt.title('Percentage of neutral articles per weekday')
    plt.xlabel('weekday')
    plt.ylabel('percentage (%)')
    plt.xticks(rotation=45)
    plt.ylim(0, 100)
    plt.grid(axis='y')
```

```
97      plt.tight_layout()
98      plt.show()
99  create_bar_chart_for_percentage_of_neutral_articles_per_weekday(articles)
100
101 def create_bar_chart_for_negative_articles_per_weekday_for_politics(articles):
102     negative_politik_articles = articles[(articles["category"] == "politik") &
            (articles["sentiment_text"] == "negative")]
103     negative_count_by_weekday = negative_politik_articles.groupby("weekday",
            observed=False).size()
104     ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
105     negative_count_by_weekday = negative_count_by_weekday.reindex(ordered_days)
106     negative_count_by_weekday.plot(kind="bar", color="red", alpha=0.6)
107     plt.title('Anzahl der negativen Artikel pro Wochentag (Kategorie
            "politik")')
108     plt.xlabel('Wochentag')
109     plt.ylabel('Anzahl der negativen Artikel')
110     plt.xticks(rotation=45)
111     plt.grid(axis='y')
112     plt.tight_layout()
113     plt.show()
114 create_bar_chart_for_negative_articles_per_weekday_for_politics(articles)
115
116 def
        create_bar_chart_for_count_of_negative_articles_per_hour_of_politics(articles):
117     negative_politik_articles = articles[(articles["category"] == "politik") &
            (articles["sentiment_text"] == "negative")]
118     negative_count_by_hour = negative_politik_articles.groupby("upload-hour",
            observed=False).size()
119     all_hours = range(24)
120     negative_count_by_hour = negative_count_by_hour.reindex(all_hours,
            fill_value=0)
121     negative_count_by_hour.plot(kind="bar", color="red", alpha=0.7)
122     plt.title('Count of negative articles per hour of category "politik"')
123     plt.xlabel('hour')
124     plt.ylabel('count')
125     plt.xticks(rotation=0)
126     plt.grid(axis='y')
127     plt.tight_layout()
128     plt.show()
129 create_bar_chart_for_count_of_negative_articles_per_hour_of_politics(articles)
130
131 def
        create_bar_chart_for_relative_count_of_negative_articles_per_hour_of_politics(articles):
```

```
    politik_articles = articles[articles["category"] == "politik"]
    total_count_by_hour = politik_articles.groupby("upload-hour",
        observed=False).size()
    negative_count_by_hour =
        politik_articles[politik_articles["sentiment_text"] ==
        "negative"].groupby("upload-hour").size()
    all_hours = range(24)
    total_count_by_hour = total_count_by_hour.reindex(all_hours, fill_value=0)
    negative_count_by_hour = negative_count_by_hour.reindex(all_hours,
        fill_value=0)
    negative_share_by_hour = (negative_count_by_hour /
        total_count_by_hour).fillna(0) * 100
    negative_share_by_hour.plot(kind="bar", color="red", alpha=0.6)
    plt.title('Anteil der negativen Artikel pro Stunde (Kategorie "politik")')
    plt.xlabel('Stunde')
    plt.ylabel('Anteil der negativen Artikel (%)')
    plt.xticks(rotation=0)
    plt.ylim(0, 100)
    plt.grid(axis='y')
    plt.tight_layout()
    plt.show()
create_bar_chart_for_relative_count_of_negative_articles_per_hour_of_politics(articles)
```

4.3 Topic-Analyse über Stichwörter

Ziel der Analyse über die Stichwörter ist die Generierung einer Karte, die die Topics als Netzwerk zeigt. Dazu wird eine Graphdatei aus den Daten erstellt, die anschließend in Gephi importiert werden kann. Die Gephi-Datei ist ebenso dem Upload angefügt. Die Stichwörter werden dazu zunächst auseinandergenommen und in einem neuen Dataframe gespeichert. Anschließend wird über die Zeilen iteriert und Informationen über Häufigkeite der einzelnen Stichwörter und Häufigkeit über Kombinationen von Stichwörtern gesammelt. Diese Informationen werden anschließend *networkx* übergeben, der daraus ein GraphML-Datei erzeugt, die in Gephi importiert werden kann.

Listing 11: Erstellung der Graph-Datei

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from collections import defaultdict

articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
```

```
articles = pd.DataFrame(articles_data)
articles.head()

keywords_df = articles['keywords'].str.split(',', expand=True)
keywords_df = keywords_df.apply(lambda x: x.str.strip())
keywords_df.head()

invalid_rows = keywords_df[keywords_df[10].notna()]

keywords_df = keywords_df.iloc[:, :10]
keywords_df.head()

G = nx.Graph()

def create_graphml_file_for_gephi(filename: str):
    word_count = {}
    for index, row in keywords_df.iterrows():
        words = row.dropna().values
        for word in words:
            if word not in word_count:
                word_count[word] = 0
            word_count[word] += 1
    for index, row in keywords_df.iterrows():
        words = row.dropna().values
        for i, word1 in enumerate(words):
            if word_count[word1] > 10:
                for j in range(i + 1, len(words)):
                    word2 = words[j]
                    if word1 != word2 and word_count[word2] > 10:
                        edge = tuple(sorted([word1, word2]))
                        if not G.has_edge(word1, word2):
                            G.add_edge(word1, word2, weight=1)
                        else:
                            G.edges[edge]['weight'] += 1
    nodes_to_remove = [node for node in G.nodes if word_count[node] <= 10]
    G.remove_nodes_from(nodes_to_remove)
    nx.write_graphml(G, f"{filename}")

create_graphml_file_for_gephi("ntv-topic-graph.graphml")
```

4.4 Topic-Analyse über Topic Modeling

Als zweiter Ansatz zur Untersuchung der Themen wurden *BERTopic* verwendet. Bertopic bietet neben der Extrahierung der Themen auch zahlreiche Funktionen zur Visualisierung. Wie in den anderen Abschnitten sollte auch der Code zur Visualisierung in einem Jupyter Notebook in einzelnen Zellen aufgerufen werden. So können die Ergebnisse direkt betrachtet und interaktiv verwendet werden.

Listing 12: Erstellung der Graph-Datei

```
import pandas as pd
import nltk
import matplotlib.pyplot as plt
from bertopic import BERTopic
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer

articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
articles = pd.DataFrame(articles_data)
articles.head()

# Bevor Texte mit Algorithmen wie ‘BERTopic‘ analysiert werden, sollten sie
# vorverarbeitet werden. Dazu zaehlt beispielsweise das Entfernen von
# Stoppwoertern, da diese ansonsten eventuell eigene Kategorien erhalten
# koennen. Dafuer wird die Bibliothek ‘nltk‘ verwendet, die entsprechende
# Datensaetze zur Verfuegung stellt. Ausserdem werden Woerter auf ihren
# Wortstamm reduziert, was die Textmenge reduziert und dabei hilft gleiche
# Woerter unabhaengig ihrer Konjunktion auch als solche zu erkennen.
stemmer = SnowballStemmer('german')
stop_words = set(stopwords.words('german'))

def preprocess_text_nltk(text):
    text = text.lower()
    tokens = word_tokenize(text, language='german')
    tokens = [word for word in tokens if word.isalnum() and word not in
        stop_words]
    tokens = [stemmer.stem(word) for word in tokens]
    return ' '.join(tokens)

articles['processed_text'] = articles['text'].apply(preprocess_text_nltk)
articles.head()

```

```
# Nun kann das Topic Modeling mit ‘BERTopic‘ durchgefuehrt werden. Dazu wird ein
# neues ‘topic_model‘ erstellt und diesem ueber die ‘fit_transform‘-Funktion die
# Dokumente bzw. die Artikeltexte uebergeben. Fuer jeden Artikel wird
# anschliessend im DataFrame die von Bertopic ermittelte Kategorie angefuegt.
documents = articles['processed_text'].tolist()
topic_model = BERTopic(language="multilingual")
topics, probs = topic_model.fit_transform(documents)
articles['topic'] = topics
articles.head()

# Zur Veranschaulich der Verteilung der Themen wird eine Funktion definiert:
def visualize_topic_distribution(articles, topic_column_name):
    topic_counts = articles[topic_column_name].value_counts()
    plt.figure(figsize=(30, 6))
    topic_counts.plot(kind='bar', color='skyblue')
    plt.xlabel('Topic')
    plt.ylabel('Count of articles')
    plt.title('Distribution of topics')
    plt.xticks(ticks=range(len(topic_counts)), labels=[f'{i}' for i in
        topic_counts.index], rotation=45)
    plt.show()

# Der nachfolgende Graph zeigt die Verteilung der Topics. Wie man sehen kann,
# faellt ein Grossteil der Artikel, ungefaehrt die Haelfe, unter die Kategorie
# ‘-1‘. Dieser Kategorie ordnet ‘BERTopic‘ alle ‘Outlier‘ zu, also Ausreisser
# zu, die keiner Kategorie zugeordnet werden konnten.
visualize_topic_distribution(articles, "topic")

# Dieses Problem zeigt sich ebenfalls in der von ‘BERTopic‘ generierten
# Darstellung. Alle grauen Punkte stellen die Ausreisser dar.
topic_model.visualize_documents(documents, hide_document_hover=True,
    hide_annotations=True)

# Ein weiteres Problem stellt die Menge an gefunden Topics mit 100+
# verschiedenen Themen dar. Fuer beide Probleme bietet ‘BERTopic‘ jedoch
# Loesungen. Die Menge an Kategorien kann jedoch ueber entsprechende
# Funktionsargumente selbststaendig angepasst werden. Im Folgenden wird die
# Anzahl an Topics auf 70 gesetzt, um die Menge an Themen leicht einzugrenzen
# und Topics, die nur aeusserst selten auftreten nicht zu beruecksichtigen.
documents = articles['processed_text'].tolist()
topic_model = BERTopic(language="multilingual", nr_topics=70)
topics, probs = topic_model.fit_transform(documents)
articles['topic'] = topics
```

```
75 articles.head()
76
77 visualize_topic_distribution(articles, "topic")
78
79 topic_model.visualize_documents(documents, hide_document_hover=True,
       hide_annotations=True)
80
81 # Als naechstes sollen die Ausreisser noch eingegrenzt werden. ‘BERTopic‘ stellt
82 # dafuer mehrere Strategien zur Verfuegung. Hier verwendet wird die automatische
83 # Reduzierung durch ‘BERTopic‘ ueber die Funktion
       ‘topic_model.reduce_outliers(documents, topics)‘.
84 # Diese werden in der neuen Spalte ‘new_topics‘ gespeichert.
85 articles['new_topic'] = topic_model.reduce_outliers(documents, topics)
86
87 visualize_topic_distribution(articles, "new_topic")
88 # Wie zu sehen ist, konnte eine Vielzahl an Ausreissern andern Topics zugeordnet
89 # werden. Die erweiterten Kategorien koennen ebenso im Cluster-Diagram
       angezeigt werden.
90 topic_model.visualize_documents(documents, hide_document_hover=True,
       hide_annotations=True)
91 # Auch diese Uebersicht stellt eine gute Visualisierung zu den
       Themenschwerpunkten von ‘n-tv‘ dar.
92 # Eine weitere von ‘BERTopic‘ zur Verfuegung gestellte Visualiserung stellt
       ebenfalls gut die Themen dar:
93 topic_model.visualize_topics()
```

5 Fazit

Dank moderner Algorithmen und zahlreicher Python-Packages lassen sich solche Analyseprojekte heutzutage auch mit relativ wenigen Vorkenntnissen im Themenbereich gut umsetzen. Gerade das Package *Matplotlib* stellt bei der Erstellung Diagrammen bzw. der Visualisierung von Ergebnissen eine große Hilfe dar. Dank der modernen Entwicklung im Bereich KI lassen sich auch nachträglich zu fremden Texten Stichwörter generieren, was bei der Erforschung der Themen und der Erstellung des Netzwerks stark geholfen hat. *BERTopic* hat sich ebenfalls als gute Wahl zur Erforschung der Themen erwiesen, da es gleichzeitig zahlreiche Visualisierungsmöglichkeiten bietet.