Übung: LLM-gestützte Textanalyse

Am Beispiel der Tagesschau-Daten

Prof. Dr. Nicolas Meseth

Die regelbasierte Textanalyse hat eine entscheidende Schwäche: Ihr müsst im Voraus wissen, wonach ihr sucht. Wörterbücher müssen manuell erstellt werden, Verneinungen bleiben unsichtbar, und kontextuelles Verstehen ist kaum möglich. Große Sprachmodelle (LLMs) versprechen, genau diese Grenzen zu überwinden: Statt Muster zu suchen, lesen und interpretieren sie Text. In diesem Experiment lernt ihr, wie ihr LLMs systematisch für Textanalyseaufgaben einsetzen könnt – und wo ihre eigenen Grenzen liegen. Das Experiment baut direkt auf dem vorherigen Experiment zur regelbasierten Textanalyse auf und stellt denselben Tagesschau-Datensatz in einen neuen methodischen Kontext.

Schritt 1: LLMs einrichten und testen

Bevor ihr mit der eigentlichen Analyse beginnt, richtet ihr zwei Zugangswege ein: ein lokal laufendes Modell für datenschutzsensitive und kostenfreie Experimente sowie einen Cloud-Zugang für leistungsstärkere Modelle.

1. Ladet LM Studio herunter und installiert es. Öffnet LM Studio und ladet über die Suchfunktion ein deutschsprachig-kompetentes Modell, zum Beispiel das kleine *Gemma 4 E2B*. Öffnet nach dem Laden die Chat-Oberfläche und stellt dem Modell eine einfache Frage auf Deutsch: „Was ist die Tagesschau?"

Beschreibt in zwei Sätzen: Wie kompetent wirkt das Modell? Gibt es Fehler oder Auffälligkeiten in der Antwort?

2. Startet den lokalen Server in LM Studio über *Local Server → Start Server*. Installiert anschließend in eurem Python-Projekt die notwendigen Pakete:

```
pip install openai pandas matplotlib seaborn tqdm scikit-learn
```

Importiert alle Bibliotheken und definiert zwei Variablen: `MODEL_LOCAL` (Modellname aus

LM Studio, z.B. `"gemma-3-4b-it"` – den genauen Namen findet ihr unter *My Models*) und `MODEL_OPENAI` (z.B. `"gpt-4o-mini"`).

3. Erstellt zwei Clients: einen für LM Studio und einen für die OpenAI API. LM Studio stellt eine OpenAI-kompatible API unter `http://localhost:1234/v1` bereit – ihr könnt das `openai`-Paket verwenden und lediglich `base_url` anpassen. Testet beide Verbindungen mit einer einfachen Anfrage: „Nenne mir in einem Satz, worum es bei der Tagesschau geht." Gebt die Antworten beider Modelle aus.

Hinweis: Speichert den OpenAI-API-Schlüssel als Umgebungsvariable `OPENAI_API_KEY`, anstatt ihn direkt im Code einzubetten.

4. Vergleicht lokale und Cloud-basierte LLMs anhand von vier Dimensionen. Füllt die folgende Tabelle aus und begründet eure Einschätzungen kurz:

Dimension	Lokal (LM Studio)	Cloud (ChatGPT)
Kosten		
Datenschutz		
Leistungsfähigkeit		
Geschwindigkeit		

Wann würdet ihr welche Variante bevorzugen?

Schritt 2: Analysecorpus aufbauen

LLM-Analysen sind pro API-Aufruf kostenpflichtig (Cloud) oder zeitintensiv (lokal). Statt den gesamten Tagesschau-Datensatz mit über 60.000 Artikeln zu analysieren, erstellt ihr zunächst einen repräsentativen Teilcorpus.

5. Ladet den Tagesschau-Datensatz in R und bereitet den Analysecorpus vor:

- Lest `tagesschau.zip` ein und ergänzt eine Spalte `year` (Erscheinungsjahr aus `date_time`).
- Filtert auf die sechs häufigsten Ressorts: `ausland`, `wirtschaft`, `inland`, `wissen`, `investigativ`, `faktenfinder`.
- Entfernt Zeilen, in denen `title` oder `text` fehlen.

Wie viele Artikel bleiben nach dem Filtern übrig?

6. Zieht aus dem gefilterten Datensatz eine **stratifizierte Zufallsstichprobe**: 50 Artikel pro Jahr (Stratifizierung nach `year`). Verwendet `set.seed(42)` für Reproduzierbarkeit. Behaltet die Spalten `date_time`, `year`, `title`, `shorttext`, `text`, `ressort` und `url`. Exportiert den Korpus als `data/tagesschau_sample.csv`, damit ihr ihn anschließend in Python laden könnt.

Wie viele Zeilen hat der exportierte Datensatz insgesamt? Welche Jahre sind enthalten?

Schritt 3: Einfache NLP-Aufgaben mit LLMs

Mit dem Korpus könnt ihr jetzt konkrete Analyseaufgaben durchführen. Ihr beginnt mit einfacheren Aufgaben, die ihr aus der regelbasierten Analyse kennt – um zu sehen, was LLMs besser, schlechter oder anders machen.

7. Ladet den Korpus in Python und schreibt zwei Hilfsfunktionen:

- `llm_text(prompt, client, model)`: Schickt einen Prompt, gibt die Textantwort zurück.
- `llm_json(prompt, client, model)`: Wie `llm_text`, aber mit `response_format={"type": "json_object"}` – gibt ein Python-Dictionary zurück.

Beide Funktionen sollen `temperature=0` verwenden, damit die Ergebnisse möglichst reproduzierbar sind. Testet beide Funktionen mit einem kurzen Beispielaufruf.

8. Wählt fünf zufällige Artikel aus dem Korpus und lasst das lokale Modell und ChatGPT jeweils eine **Zusammenfassung in zwei deutschen Sätzen** erstellen. Nutzt den `text`-Wert als Eingabe, begrenzt ihn aber auf maximal 600 Zeichen (`text[:600]`), um die Promptlänge zu kontrollieren.

Gebt Titel, Zusammenfassung des lokalen Modells und Zusammenfassung von ChatGPT nebeneinander aus. Wo gibt es inhaltliche Unterschiede oder Fehler?

9. Wie reißerisch ist ein Nachrichtentitel? Schreibt eine Funktion `rate_sensational(title, client, model)`, die einen Artikel-Titel in eine von vier **ordinalen Kategorien** einordnet und den Kategorienamen als String zurückgibt:

- `"neutral"` – sachliche Meldung, keine besondere Wertung
- `"pointed"` – zugespitzte oder deutlich pointierte Formulierung
- `"alarming"` – alarmierender oder aufwühlender Ton
- `"sensational"` – reißerisch, provozierend, Clickbait-Charakter

Wendet die Funktion auf 20 zufällige Artikel an. Gebt alle Titel der Kategorien `"alarming"` und `"sensational"` aus. Sind die Einstufungen plausibel?

Außerdem: Warum könnten ordinale Kategorien für LLMs zuverlässiger sein als eine numerische Skala von 1 bis 5?

10. Schreibt eine Funktion `rate_sentiment(title, text, client, model)`, die bewertet, ob ein Artikel eine **gute oder schlechte Nachricht** enthält. Verwendet eine viergliedrige Skala ohne neutrale Mittelkategorie, um das Modell zu einer Entscheidung zu zwingen:

- `-2` – sehr schlechte Nachricht (Katastrophe, Tod, schwere Krise)
- `-1` – eher schlechte Nachricht (Konflikt, Problem, Rückschlag)
- `+1` – eher gute Nachricht (positive Entwicklung, Fortschritt)
- `+2` – sehr gute Nachricht (Durchbruch, gelöstes Problem, Erfolg)

Wendet die Funktion auf 30 zufällige Artikel an. Berechnet den Mittelwert pro Ressort. Welches Ressort berichtet am meisten über schlechte Nachrichten?

Schritt 4: Ressort-Klassifikation und Evaluation

In der regelbasierten Textanalyse habt ihr Artikel über Wörterbücher klassifiziert. Jetzt fragt ihr ein LLM: Kann es das besser – und wie viel besser?

11. Schreibt eine Funktion `predict_section(title, shorttext, client, model)`, die das Ressort eines Artikels vorhersagt. Das Modell soll genau eines der sechs Ressorts zurückgeben: `ausland`, `wirtschaft`, `inland`, `wissen`, `investigativ`, `faktenfinder`. Nennt im Prompt die vollständige Liste und fordert eine exakte Antwort ohne weitere Erklärung.

Testet die Funktion an drei Beispielen aus dem Korpus und vergleicht Vorhersage und wahres Label.

12. Wendet die Klassifikation auf einen stratifizierten Test-Datensatz von 60 Artikeln an (10 pro Ressort). Führt die Klassifikation mit beiden Modellen durch (lokal und ChatGPT) und speichert die Ergebnisse in den Spalten `pred_local` und `pred_openai`. Berechnet anschließend die **Genauigkeit** (Anteil korrekt klassifizierter Artikel) für beide Modelle.

Welches Modell schneidet besser ab? Ist der Unterschied größer als erwartet?

13. Visualisiert die Klassifikationsergebnisse als **Konfusionsmatrix** – je eine Heatmap für das lokale Modell und für ChatGPT. Auf der x-Achse stehen die vorhergesagten Ressorts, auf der y-Achse die wahren Ressorts.

Welche Ressorts werden am häufigsten verwechselt? Lässt sich ein inhaltliches Muster erkennen?

14. Berechnet für jedes Ressort **Precision**, **Recall** und **F1-Score** mithilfe von `sklearn.metrics.classification_report`. Gebt den Report für beide Modelle aus.

Welches Ressort ist am schwersten zu klassifizieren? Diskutiert, warum das so ist.

Schritt 5: Sentiment-Analyse im Zeitverlauf

Mit den Grundlagen aus Schritt 3 könnt ihr jetzt eine umfangreichere Analyse aufsetzen: Wie verändert sich die Grundstimmung der Tagesschau-Berichterstattung über Ressorts und Jahre hinweg?

15. Wendet die Funktion `rate_sentiment()` aus Aufgabe 10 auf den **gesamten Stichprobencorpus** an. Nutzt `tqdm.pandas()` und `progress_apply()`, um den Fortschritt zu verfolgen. Speichert die Ergebnisse als neue Spalte `sentiment` im DataFrame.

Wie viele Werte konnten nicht geparst werden (d.h. ergaben `None`)? Was könnte der Grund dafür sein?

16. Berechnet den Durchschnittssentiment pro **Ressort** und **Jahr** für Artikel ab 2010. Speichert das Ergebnis als Pivot-Tabelle (Zeilen = Ressorts, Spalten = Jahre). Gebt die Tabelle aus.

17. Visualisiert die Pivot-Tabelle als **Heatmap** (x-Achse: Jahr, y-Achse: Ressort, Farbe: Durchschnittssentiment). Verwendet eine divergierende Farbskala, die negative Werte (rot) von positiven (grün) unterscheidet, mit 0 als Mittelpunkt.

Was fällt auf? In welchen Jahren und Ressorts überwiegt negatives Sentiment? Lassen sich gesellschaftliche Ereignisse (z.B. Wirtschaftskrisen, Pandemie) in der Heatmap ablesen?

Schritt 6: Wer spricht über wen?

Named-Entity-Recognition (NER) identifiziert Personen, Orte und Organisationen in Texten. Im Unterschied zur regelbasierten Erkennung mit Namenslisten versteht ein LLM den Kontext, erkennt verschiedene Schreibweisen und unterscheidet Personen mit gleichem Nachnamen.

18. Schreibt eine Funktion `extract_persons(text, client, model)`, die alle im Text genannten Personen als Python-Liste zurückgibt. Der Prompt soll das Modell anweisen, ausschließlich Personennamen (Vor- und Nachname) zurückzugeben – keine Orte oder Organisationen. Das Ergebnis soll ein JSON-Dictionary der Form `{"personen": ["Name1", "Name2"]}` sein.

Testet die Funktion an fünf Artikeln und beurteilt die Qualität der Extraktion.

19. Wendet `extract_persons()` auf alle Artikel im Korpus an und bringt das Ergebnis mit `explode()` in Langform: eine Zeile pro genannter Person. Behaltet `year`, `ressort` und `url` als Kontextspalten.

Erstellt eine Rangliste der 20 meistgenannten Personen im gesamten Datensatz. Wer steht an der Spitze, und entspricht das euren Erwartungen?

20. Wählt fünf bekannte Politiker aus dem Datensatz – zum Beispiel *Merkel*, *Scholz*, *Merz*, *Baerbock* und *Habeck*. Schreibt eine Funktion `person_sentiment(text, person, client, model)`, die bewertet, wie die genannte Person in einem Artikel dargestellt wird (Skala -2 bis +2). Gibt `None` zurück, wenn die Person nicht erwähnt wird.

Filtert den persons_df auf Artikel, in denen mindestens einer der fünf Politiker vorkommt, und wendet die Funktion an.

21. Aggregiert das Sentiment pro Politiker und Jahr (ab 2010) und visualisiert es als **Heatmap** (x-Achse: Jahr, y-Achse: Politiker, Farbe: Durchschnittssentiment). Verwendet dieselbe divergierende Farbskala wie in Aufgabe 17.

Welche Muster sind erkennbar? Gibt es Jahre, in denen ein Politiker besonders negativ oder positiv dargestellt wurde?

Schritt 7: Framing-Analyse mit LLMs

Framing beschreibt, welchen Deutungsrahmen ein Artikel einem Thema gibt. Derselbe Migrationsbericht kann als humanitäre Krise, als Sicherheitsbedrohung oder als politischer Streit gerahmt sein. LLMs können solche impliziten Rahmungen erkennen – etwas, das regelbasierte Verfahren kaum leisten können.

22. Filtert den Korpus auf Artikel zum Thema **Migration**, indem ihr prüft, ob der Titel eines der Wörter „Migration", „Flüchtling", „Asyl", „Migranten" oder „Einwanderer" enthält. Wie viele solche Artikel gibt es im Stichprobencorpus?

Schreibt anschließend eine Funktion `analyze_framing(title, text, client, model)`, die das dominante Framing eines Artikels klassifiziert. Verwendet mindestens fünf Kategorien: `humanitaer`, `sicherheitspolitisch`, `rechtlich`, `wirtschaftlich`, `politischer_streit`.

23. Wendet `analyze_framing()` auf alle gefundenen Migrationsartikel an. Erstellt zwei Visualisierungen:

- Ein horizontales Balkendiagramm der Gesamtverteilung aller Framing-Kategorien.
- Ein gestapeltes Balkendiagramm der Framing-Verteilung nach Jahr (falls Artikel aus mehr als zwei Jahren vorliegen).

Welches Framing dominiert in der Tagesschau-Berichterstattung über Migration? Hat sich das im Zeitverlauf verändert?

Schritt 8: Reflexion

24. Vergleicht die LLM-basierte Analyse mit der regelbasierten Analyse aus dem vorherigen Experiment entlang von vier Dimensionen. Füllt die folgende Tabelle aus und begründet eure Einschätzungen:

Dimension	Regelbasiert	LLM-basiert
Transparenz		
Reproduzierbarkeit		
Skalierbarkeit		
Qualität		

Für welche Aufgaben eignet sich welcher Ansatz besser?

25. Diskutiert die folgenden ethischen und methodischen Fragen der LLM-basierten Inhaltsanalyse:

- Können LLMs politisch voreingenommen sein? Wie würdet ihr das methodisch testen?
- Welche Risiken entstehen, wenn LLM-basierte Analysen in journalistische oder wissenschaftliche Entscheidungen einfließen?
- Wie lässt sich die Qualität einer LLM-Analyse valide messen – und was bedeutet das für eure Ergebnisse aus diesem Experiment?