Universität Trier

Semester: Sommersemester 2021

Modul: Computerlinguistische Programmierung

Dozierende: Herr Dr. Sven Naumann

# Computerlinguistik-Projekt

# Topic Modeling mit deutschen Tweets

Julia Karst, Tobias Niedballa, Sarah Ondraszek, Julia Weyer

---

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Planung

## 1.1 Projekt-Repository

Das Projekt und sämtliche Skripte seien auf GitHub zu finden unter `https://github.com/sarahondraszek/comp_ling_LDA`.

## 1.2 Terminplanung

27.04.: Terminplan + Thema ankündigen

04.05. - 11.05.: Bereitstellung eines Tweetskorpus

11.05.: 1. Zwischensitzung: Update Stand Korpus, weiteres Vorgehen

15.05.: Fertigstellung des Preprocessing (Tokenisierung, Lemmatisierung usw.)

20.05.: Beginn des Topic Modelings mit LDA in Python

08.06.: 2. Zwischensitzung: Ergebnisse des LDA

15.06.: Finale Zusammensetzung der Komponenten

22.06.: Bereitstellen eines Prototypen

29.06.: 1. Version der Dokumentation

14.07.: Präsentation

## 1.3 Vorgehen

1. Für ein Korpus entscheiden
2. Hypothese aufstellen: Wie werden die Topics aussehen?, z.B. „Die User haben in der ersten Mai-Woche vor allem über XY getwittert“.
3. Textkorpus aufbereiten
4. Für eine oder mehrere Bibliotheken entscheiden: für die Textverarbeitung und natürlich für das Topic Modeling selbst und evtl. für die Visualisierung (ggf. ist die Visualisierung schon enthalten), LDA-Bibiliothek: Gensim/LDA-Mallet
5. Implementierung + Zusammenführen der Module, GUI

6. Intertextueller Verweis (nur eine Möglichkeit, aber wahrscheinlich zu wenig Zeit)
7. Auswertung der Ergebnisse + Benutzerhandbuch

### 1.4 Rollen

**Übergreifende Rollen**:

1. Recherche
2. Dokumentation des eigenen Arbeitsbereiches
3. Interpretation der Topics

**Projektler*innen und ihre Rollen:**

1. Julia Karst: Überprüfung der Parameter, Visualisierung, Programmierung der GUI, Benutzerhandbuch
2. Tobias Niedballa: Programmierung des Preprocessing-Skripts
3. Sarah Ondraszek: Tweet-Beschaffung, Hilfe beim Preprocessing-Skript, Modulverwaltung, Dokumentverwaltung und -erstellung, LDA-Modell-Skript
4. Julia Weyer: Programmierung des Preprocessing-Skripts

## 2 Einleitung

Ziel dieses Projektes ist Topic Modeling auf kurzen Texten, in diesem Fall sollen aus deutschen Tweets der ersten Maiwoche Topics gezogen und visualisiert werden, um so auf bestimmte Ereignisse zurückschließen zu können. Die Hypothese, die dabei im Vordergrund steht, ist, dass wegen der momentanen politischen und pandemischen Lage in der ersten Maiwoche 2021 vor allem Themen wie Corona, Kanzlerwahl, Lockdown usw. im Fokus stehen werden. Die maschinell erzeugten Topics sollen bestenfalls diese These stützen.
Um Topic Modeling mit LDA durchführen zu können, muss natürlich zunächst das Korpus erstellt werden. Folgend muss das Korpus passend aufbereitet werden. Nach der Anwedung der LDA auf das Korpus müssen die Topics interpretiert werden. Jeder dieser Schritte wird im folgenden Text näher erläutert.

# 3 Tweets sammeln

Zuständigkeit: S.O.

Um überhaupt Topic Modeling machen zu können, muss ein passendes Korpus erstellt werden. Im Netz finden sich ein paar englisch-sprachige Twitter-Korpora, wie zum Beispiel in der *TweetsCOV19*-Knowledge Base, das vor allem auf Covid-spezifisiche Tweets ausgerichtet ist (Dimitrov et al. 2020). Durch die Twitter API kann man sich jedoch relativ schnell und flexibel ein eigenes Korpus aufbauen. Dafür muss der Benutzer sich zunächst für einen *Twitter Developer Account* registrieren und sein Projektziel beschreiben. Sobald die Bewerbung genehmigt ist, kann man, je nach *Hintergrund des Projekts*, auf verschiedene Methoden zum Tweet Retrieven stürzen. Mit akademischem Projekthintergrund stehen dafür unter anderem die *Sieben-Tage-Zurück-* und die *Archiv*-Methode bereit. Wie die Namen es bereits andeuten, kann man entweder Tweets aus dem Zeitraum der vergangenen Woche (also vom Scrape-Datum genau 168 Stunden zurück), oder Tweets aus dem gesamten Twitter-Archiv, ziehen. Für dieses Projekt bietet sich der Zeitraum einer Woche an, da unsere Hypothese Tweets einer Woche behandelt. Twitter bietet für registrierte Developer die Bibliothek *searchtweets*[1] an, die im Folgenden verwendet wird, um Zugriff auf die Tweets zu bekommen (Twitter 2021, Analyze past conversations zu finden im Abschnitt Tutorials). Mit Hilfe von `pip install searchtweets` und `import searchtweets` kann die Bibliothek geladen und importiert werden. Um auf die Tweets zugreifen zu können, muss man auf die Twitter API mit seinen *user-keys* zugreifen, die man bei der Anmeldung erhalten hat. Diese kann man mit `load_credentials` laden und für seine Anfrage verwenden. Um via `searchtweets` eine Anfrage an die API zu stellen, muss man zunächst eine *query* erstellen und definieren. Dazu nutzt man `searchtweets.gen_request_parameters()`; in dieser Methode kann sowohl die Anfrage (z.B. „lang:de“, „snow lang:en“) als auch die Start- und Endzeit der Anfrage und die Ergebnisse pro HTTP-Zugriff festgelegt werden. Die Anfrage muss einer bestimmten Form entsprechen, die Twitter in seinen Docs gelistet hat. Gewählt werden können beispielsweise die Sprache, die Geo-Location oder der User[2]. In einer eigens implementierten Methode

```
collect_tweets_in_files(path, query, credentials)
```

können die vorher gespeicherten Zugangsdaten, die Query und der Pfad, in dem die

[1] `https://pypi.org/project/searchtweets/`

[2] `https://developer.twitter.com/en/docs/twitter-api/tweets/search/quick-start/recent-search`

Tweet-Dateien gespeichert werden sollen, übergeben werden. Die Methode nutzt intern eine weitere Methode `check_files(path)`, die überprüft, ob auf gegebenem Speicher-Pfad schon eine Datei mit gleichem Namen vorhanden ist - dies sollte verhindert werden, um gedoppelte Tweet-Sammlungen zu vermeiden. Wenn dies gelungen ist und noch keine Datei mit dem Namen vorliegt, wird ein

```
rs = ResultStream(request_parameters, max_results, max_pages, **credentials
)
```

eingesetzt, um Tweets aus gewünschtem Zeitraum zu laden. Dafür müssen außerdem die Parameter `max_results, max_pages, max_tweets` gesetzt werden, in dem Fall auf 100 maximale Ergebnisse pro Fetch, 100 maximale Seiten pro Zugriff und insgesamt 10000 Tweets als Ergebnis im ResultStream. Manuell muss die Anzahl an gewünschten Tweets mit `rs.max_tweets` noch auf das gewählte `max_tweets` gesetzt werden. Aus dem ResultStream kann man die Tweets via `rs.stream()` laden und als Liste speichern. Der Output besteht dann aus den Tweet-IDs, dem Text, der newest- und oldest_id, dem result_count und dem next_token. Benötigt wird davon später nur der Text. Das Listen-Ergebnis wird in einem `pandas.DataFrame()` gespeichert und als *.csv* exportiert.

# 4 Preprocessing

Zuständigkeit: T.N., S.O., J.W.

## 4.1 Text aus .csv filtern

Da der Output lokal gespeichert wurde und wir ebenfalls wissen, wo die Dateien liegen, können wir auf den Dateipfad zugreifen. Ebenfalls wissen wir, dass der Output als .csv Dateien vorliegt, weswegen wir durch `'./data/corpus/*.csv'` auf diese Dateien zugreifen und in einer Listenform in der Variable docs für die Weiterverarbeitung speichern können. NLTK erlaubt uns weiterhin mit der Funktion

```
nltk.corpus.stopwords.words("german")
```

eine Liste an Stoppwörtern der Deutschen Sprache anzulegen und in der Variable `stopwords_ger` zu speichern. Auf diese greifen wir an späterer Stelle zurück.

## 4.2 Überflüssige Zeichen und englische Tweets entfernen

Nachdem wir uns den Aufbau der .csv Dateien genauer angeschaut haben, haben wir uns dazu entschieden, die ersten Schritte des Preprocessings in einer eigens dafür angelegten Methode `preprocessData()` anzulegen. Diese Methode verhält sich wie folgt: Es wird eine neue Listenvariable `temp_docs = []` angelegt. Darauf folgt eine Schleife `for file in input_file_list`, die nacheinander alle .csv Dateien in dem festgelegten Verzeichnis durchläuft, und diese Dateien jeweils erst öffnet und dabei nur die Spalte text ausliest und in der String Variable datastring temporär speichert. Um sämtliche irrelevanten Informationen aus diesem String zu entfernen verwenden wir Reguläre Ausdrücke in den folgenden sechs Schritten: Da Nutzernamen in Twitter durch das vorangehende @ Symbol indentifizierbar sind, können mit `re.sub(r"@[S]*|#", "", datastring)` leicht sämtliche Nutzernamen aus dem datastring entfernt werden. Ähnlich verhält es sich mit den in den Tweets enthaltenen Hashtags, bei denen der Inhalt allerdings erhalten bleiben soll, da diese meist Informationen bezüglich des Themas des Tweets enthalten, weshalb nur das Rautezeichen entfernt wird. Gleichermaßen lassen sich Links durch https oder www identifizieren und ebenfalls leicht durch `re.sub(r"https:[S]*|www[S]*", "", datastring)` entfernen. Daraufhin werden Zahlen mit dem RA `re.sub(r"[0—9]*", "", datastring)` entfernt. Desweiteren liegen in Tweets klassiche Zeilenumbruchmarkierungen vor, die ebenfalls durch einen Regulären Ausdruck entfernt werden können: `re.sub(r"\n", "", datastring)`. Durch `re.sub(r"W", "", datastring)` lassen sich Sonderzeichen entfernen. Ein letzter RA wurde verwendet, um irrelevante, twitterspezifische Marker zu entfernen:

```
re.sub("RT|NaN|via", "", datastring)
```

Dabei handelt es sich um RT (retweets), NaN (Inhaltslose Tweets), sowie via, welches Ausschluß darüber gibt, auf welchem Gerät der Tweet verfasst wurde.

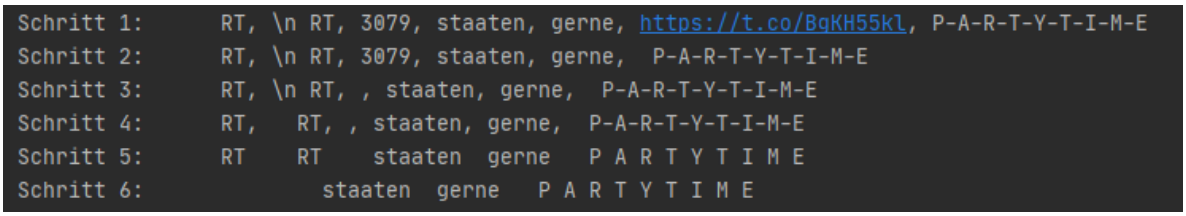


Abbildung 1: Darstellung der oben genannten regulären Ausdrücke anhand eines Beispiel-Strings. Im Verlauf des Preprocessings entstehen sowohl vereinzelte Buchstaben sowie überschüssige white space Zeichen. Diese werden jedoch im Verlauf der Tokenisierung eliminiert.

## 4.3 Tokenisierung

Um den Text weiterverarbeiten zu können, muss er zunächst in Einheiten der Wortebene segmentiert werden. Die Tokenisierung findet ebenfalls in der For-Schleife der Methode `preprocessData()` statt. Hierbei wird `nltk.wordpunct_tokenize()` auf den datastring ,in diesem Fall `datasting.lower()` angewendet, wodurch wir eine Liste an Token erhalten, die zudem ausschließlich kleingeschrieben sind. Nun greifen wir auf die Liste der deutschen Stoppwörter (`stopwords_ger`) zurück, um diese mit unserer Liste abzugleichen und die Stoppwörter zu entfernen. Aus dieser Liste werden danach alle Token mit der Länge von 2 Zeichen oder weniger entfernt und die bereinigte Liste in der Variable text gespeichert. Im letzten Schritt der Methode werden alle Token aus text an eine temporäre Liste angehangen, welche dann wiederum in unserer Variable `temp_docs` ebenfalls per `.append()` angehangen wird. Sobald die For-Schleife alle .csv Dateien abgelaufen ist, wird `temp_docs`, eine Liste von Dokumenten, die jeweils als Listen von Token dargestellt werden, zurückgegeben.

## 4.4 Lemmatisierung

Für die Lemmatisierung wird spacy verwendet, wofür einmalig die Pipeline `de_core_news_md` heruntergeladen werden muss. Diese wird der Variable nlp durch `spacy.load()` zugewiesen und die Länge auf maximal 2 Millionen begrenzt. In der Methode `lemmatizer()` wird zunächst eine leere Liste unter dem Namen `lemmatized_words` angelegt. Daraufhin verwenden wir eine geschachtelte For-Schleife. Zunächst wird die, der Methode übergebene Liste, `input_docs` durchlaufen. Nebenbei wird eine temporäre, leere Liste `temp_list` sowie ein temporäres Dokument `temp_doc` erzeugt. Dieses enthält ein nlp-Objekt, welches

aus einen String, der aus unserer vorherigen Liste erstellt wurde besteht. In der zweiten For-Schleife werden die einzelnen Token des temporären Dokuments durchlaufen und durch spacy's `word.lemma_` Funktion in lemmatisierter Form der `temp_list` angehangen. Außerhalb der zweiten For-Schleife wird dann die `temp_list` wiederum der `lemmatized_words` angehangen, wodurch wir im Endeffekt eine Liste an Listen von lemmatisierten Token erhalten.

## 4.5 N-Gramme

In der Methode `ngrams()` wir die `Phrases()` Funktion von Gensim verwendet, die Bigramme in der Variable bigram speichert. Dabei werden Paare von Token, die ein Bigramm bilden durch einen Unterstrich verbunden. Desweiteren lässt sich bei `Phrases()` die Mindestanzahl an Vorkommen der Bigramme definieren. Hierbei experimentierten wir mit verschiedenen Werten und entschieden uns für den Mindestwert 20. In einer geschachtelten For-Schleife werden die Bigramme anhand einer If-Abfrage, die überprüft ob es sich um ein Bigramm handelt, an unsere Liste an Token angehangen. Auf zusätzliches Einbinden von Trigrammen wurde verzichtet, da diese kein besseres Ergebnis als Bigramme zeigen und zusätzlich eine höhere Berechnungszeit benötigen.

## 4.6 Sprachfilter

Ebenfalls verzichtet wurde auf eine zusätzliche Methode zur Sprachfilterung, die nur deutschsprachige Token übrig lassen würde. Stattdessen wurden besonders seltene und häufige Wörter anhand ihrer Dokumenthäufigkeit gefiltert und entfernt (vgl. Rehurek 2021). Dadurch konnten bereits z.B. koreanische und japanische Schriftzeichen herausgenommen werden. Die zumeist englischen Token, die nach dem Filtern übrig bleiben, lassen sich in die Interpretation der Topics miteinbeziehen, da Twitter eine Plattform ist, in der auch im deutschsprachigen Raum viele Elemente aus der englischen Sprache genutzt werden.

## 4.7 Bag-Of-Words

Im letzten Schritt, vor der Modellierung eines entsprechenden LDA-Modells, muss das Korpus noch in die Bag-Of-Words-Form gebracht werden. Dies dient der Feature-Generierung und verwirft die Ordnung von Wörtern in Sätzen (z.B. grammatikalische Ordnung). Somit werden Wörter, ähnlich wie in einer Tasche, „randomisiert“ und aus

ihrer Position gelöst. Für jedes Dokument wird eine solche *Bag-Of-Words* erstellt, dabei wird zusätzlich ein Mapping zwischen den Wörtern und ihrer Frequenz im Dokument vollzogen. Gensim stellt bereits eine Methode für die Konvertierung zur Verfügung, genutzt wird also `doc2bow` für jedes Dokument (vgl. Rehurek 2021). Die BOW-Darstellung für jedes Dokument wird wieder in einer Liste von Listen gesammelt, wobei jede BOW-Darstellung eine Liste ist und die „große“ Liste drumherum alle Dokumente zusammenfasst. Das Ergebnis wird als Dokument gespeichert, um im späteren Teil einfach geladen werden zu können.

# 5 LDA

Zuständigkeit: S.O.
Topic Modeling kann durch verschiedene Algorithmen realisiert werden. Die Hauptsache ist, dass das Ergebnis dabei eine Liste, eine Menge von Topics der Dokumente ergibt. Das populärste und wohl bekannteste Verfahren ist LDA, die *Latent Dirichlet Allocation* (Blei et al. 2003, S. 1). Für diese Projektarbeit sei vorausgesetzt, dass ein Grundwissen über Topic Modeling besteht. Es gibt natürlich weitere Algorithmen, wie zum Beispiel die erweiterte Version der LDA, das *Embedded Topic Modeling* oder Topic Modeling mit *Word2Vec*-Verfahren. Im kleineren Projektrahmen ist LDA jedoch zu präferieren, da lediglich ein kleines Korpus vorliegt und nur simuliert werden soll, ob die anfänglich gestellte Hypothese verifiziert werden kann (Hong und Davison 2010, S. 81–82). Zahlreiche Bibliotheken, die in Python genutzt werden können, werden für LDA angeboten und ebenfalls in diesem Projekt zu Rate gezogen. Das Skript richtet sich nach Rehurek, der in seinem Artikel zur Dokumentation der Nutzung von Gensim und LDA die Funktionen erläutert (Rehurek 2021).
Für das Erstellen eines LDA-Modells werden zuerst, im entsprechenden Skript, die aus dem Preprocessing stammenden Dateien geladen (tweet_dictionary, docs, bow_corpus). Danach werden die verschiedenen Parameter gesetzt, dazu gehören:

- num_topics
- chunk_size
- passes
- iterations

- eval_every

Die Anzahl der Topics, bestimmt durch num_topics, richtet sich nach eigenen Präferenzen aus. Da kein besonders großes Korpus vorliegt, wird eine Anzahl von fünf Topics gewählt. Im ersten Durchlauf wurden zunächst zehn Topics gewählt, jedoch wiederholen sich die Wörtern innerhalb der Topics immer wieder. Da, wie zuvor angemerkt, kein großes Korpus vorliegt, kann das gesamte Korpus als Chunk (chunk_size) übergeben werden. Die Durchläufe (passes) und Iterationen (iterations) bleiben auf dem Standardwert 25 und 60 (vgl. Rehurek 2021). Die Model Perplexity (eval_every) wird nicht berechnet, da das hohe Computing-Kosten verursacht. Die Parameter werden dem Lda-Modell Objekt übergeben und somit das Modell erzeugt. Danach wird das Modell lokal gespeichert und später in der Visualisierung wieder geladen.

1. Erster Durchlauf:
   (a) Bisherige Probleme: Topics sind schwierig interpretierbar; Ganz normales Problem - mögliche Lösungen sind mehr Tweets sammeln, no_above und no_below anpassen, Parameter anpassen, Modell wechseln
2. Zweiter Durchlauf:
   (a) Anpassung der no_above und no_below Werte. Mit höheren Werten für für no_above gibt es etwas bessere Ergebnisse. Topics immer noch sehr zerstreut. Ergebnisse sind eher von geringer Qualität, Topics lassen sich interpretieren, aber nur mit viel Fantasie.
3. Dritter Durchlauf:
   (a) Hinzufügen von Bigrammen im Preprocessing-Schritt verbesserte die Ergebnisse immens, fremdsprachige Tweets im Ergebnis.

## 5.1 Parameteranpassung

Zuständigkeit: J.K.
Um bessere Ergebnisse und somit sinnvollere Topics zu erhalten, wurde getestet, welche Modellparameter die besten Ergebnisse erzielen. Es stellte sich heraus, dass die Anzahl der Topics auf etwas niedriger als die ursprünglichen geplanten zehn gesetzt werden sollte, da sich sonst viele Begriffe in mehreren Topics wiederholen. Wir entschieden uns daher, insgesamt fünf Topics ausgeben zu lassen.

Ebenso wurde überprüft, wie das Dictionary am besten zu filtern ist. Ein Wert von 2 oder 3 für no_below scheint dabei gut geeignet zu sein und für no_above bietet sich ein höherer Wert als der Standard von 0.5 an (z.B. 0.7 bis 0.9, abhängig von den restlichen Parametern). Letzteres liegt u.a. darin begründet, dass einige Begriffe wie z.B. 'Corona' so häufig vorkommen, dass sie bei einem zu niedrigen Wert von no_above herausgefiltert werden.

# 6 Visualisierung der Ergebnisse mit LDAvis

Zuständigkeit: J.K., S.O.

Die Ergebnisse aus dem Durchlauf des Topic Modelings können mit pyLDAvis[3] visualisiert werden. Dazu lädt man das entsprechende Package in seine Entwicklungsumgebung und nutzt die gegebene Methode *prepare*, die das Modell, das BOW-Korpus und das Tweet Dictionary lädt. Das Ergebnis der Visualisierung kann man sich entweder direkt anzeigen lassen oder als HTML zwischenspeichern.

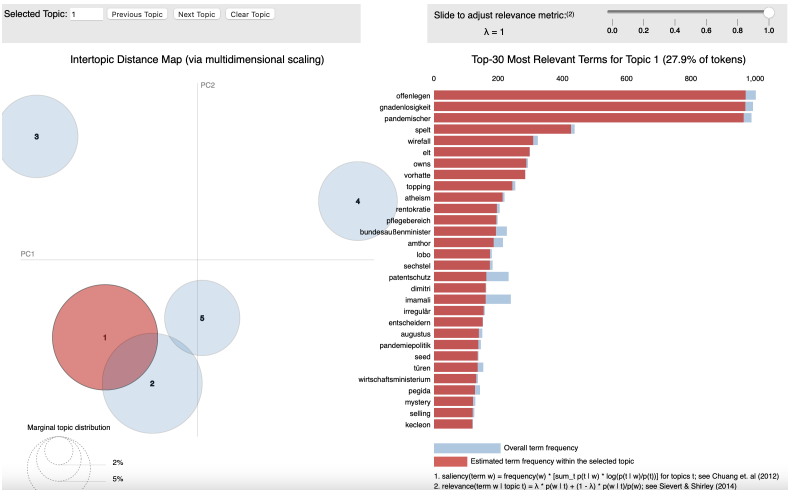


Abbildung 2: Visualisierung mit pyLDAvis

Das Programm *visualization.py* nutzt die aktuelle Version von pyLDAvis und speichert die Ergebnisse als HTML. Jedoch traten bei der Verwendung der neuesten Version von

[3] `https://pyldavis.readthedocs.io/en/latest/modules/API.html`

pyLDAvis Probleme auf, z.B. konnte die Visualisierung nicht im Browser geöffnet werden. Versuchte man den Befehl `pyLDAvis.show()` zu verwenden, erhielt man die Fehlermeldung „No such file or directory: https://cdn.jsdelivr.net/gh/bmabey/pyLDAvis@3.3.1/pyLDAvis/js/ldavis.v1.0.0.css“. So wurde zusätzlich noch ein alternatives Programm *visualization_alternative.py* geschrieben, in welchem aufgrund der Nutzung einer älteren Version von pyLDAvis (Version 2.1.2) dieses Problem nicht auftritt.

# 7 Benutzeroberfläche

Zuständigkeit: J.K.

Die Benutzeroberfläche wurde mit Tkinter erstellt. Die einzelnen Seiten sind sich dabei grundsätzlich in ihrem Aufbau ähnlich. Es gibt jeweils einen hellblauen Hintergrund *background_frame*, einen Frame *upper_frame* mit einem Label für den Titel der Seite und einen *middle_frame* mit Labeln für die Beschreibung und evtl. Buttons oder Entries für spezielle Funktionen. Im *lower_frame* gibt es zusätzlich noch Buttons, um zwischen den Seiten zu wechseln. Sämtliche Frames, Labels, Entries und Buttons wurden mithilfe von `place` platziert. Beispielsweise wurde der Button, welcher das Preprocessing startet, wie folgt implementiert: `button.place(relx=0.1, rely =0.7, relwidth=0.8, relheight=0.2)`. Relativ zu dem Frame, in welchem der Button liegt, beziehen sich *relx* und *rely* auf die x- und y-Werte und *relwidth* und *relheight* auf die Größe des Buttons. Somit wird dieser Button am unteren Teil des Frames angezeigt und nimmt, mittig platziert, 80% der Breite ein. Die Nutzung von `place` ermöglicht es zudem, die einzelnen Elemente auch bei einer Verzerrung des GUI-Fensters anschaulich darzustellen.

Insgesamt gibt es sechs Seiten: Eine Startseite, jeweils eine Seite für Korpusauswahl, Preprocessing, Parameterauswahl und Visualisierung und einen Endbildschirm. Das Wechseln zwischen den einzelnen Seiten bereitete anfangs noch Probleme. Eine Lösung fand sich schließlich in der Nutzung der Klasse `OrganiseFrames(tk.Tk)`[4]. Durch Klicken von 'Go Back' bzw. 'Done' kann man vor und zurück durch die Seiten gehen. Die Buttons verwenden dabei den Befehl `lambda: controller.show_frame("...")`, wobei der Klassenname der entsprechenden nächsten Seite angegeben wird.

Um das LDA Topic Modeling durchzuführen, greift die GUI auf das Dokument 'docs' im Ordner 'data' zu (vgl. Abschnitt 4). Dies erlaubt es, nicht jedes mal Preprocessing durchführen zu müssen, da dies viel Zeit kostet. Insbesondere bedeutet dies, dass vor

[4] vgl. https://stackoverflow.com/questions/7546050/switch-between-two-frames-in-tkinter

der Nutzung eines neuen Korpus erst die Datei 'docs' wieder gelöscht werden muss. Geschieht dies nicht, so wird immer wieder die 'docs' Datei (basierend auf dem alten Korpus) verwendet, auch wenn man andere Dateien für den Korpus auswählt.
Auf 'Select Corpus' können durch Anklicken des Buttons 'Select Files' mehrere csv-Dateien ausgewählt werden. Hierbei wird die Methode `addFile(frame)` aufgerufen[5] und die ausgewählten csv-Dateien werden in der Datei 'save.txt' gespeichert. Anschließend kann Preprocessing durchgeführt werden. Hierbei setzt sich der Korpus aus den einzelnen Files zusammen, die in 'save.txt' aufgelistet sind und die Datei 'docs' wird erzeugt. Gibt es bereits die Datei 'docs', so können die Seiten 'Select Corpus' und 'Preprocessing' übersprungen werden.

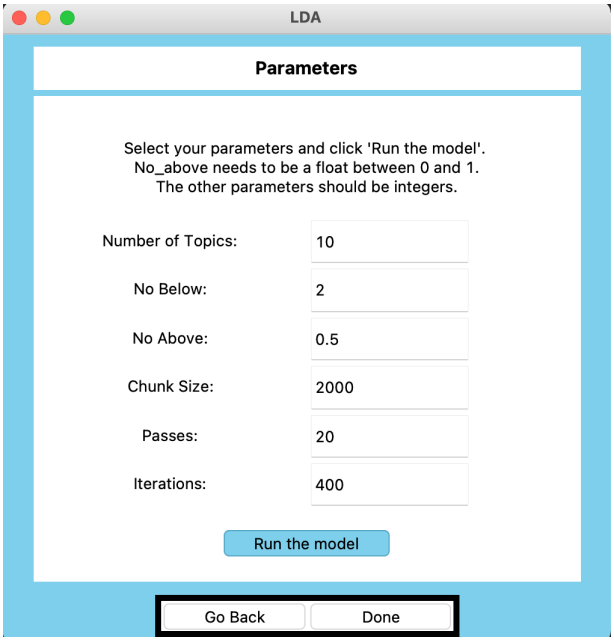


Abbildung 3: Parameterauswahl mit Default-Werten bei Verwendung der GUI

Anschließend kommt ein Expertenmenü, in dem man einige Modellparameter auswählen kann. Zur Auswahl steht dabei die Anzahl der Themen, die zwei Filter *no_below* und *no_above* und die Modellparameter *Chunk Size*, *Passes* und *Iterations*. Als Default-Werte für das Modell wurden hierbei die Parameter nach Rehurek übernommen. Als Default-Werte für die `Dictionary.filter_extremes`-Werte wurde für *no_below* 2 gewählt und für *no_above* der Standard von 0.5. Durch Klicken von 'Run the model'

[5]vgl. Methode 'addApp' in https://github.com/developedbyed/py-gui

wird das Modell laufen gelassen (wird dies nicht gedrückt, verwendet die Visualisierung im nächsten Schritt das Modell des letzten Durchlaufes). Die Ergebnisse werden zunächst auf der Konsole ausgegeben. Die entsprechende Methode `run_tm(topics,below, above,chunksize,passes,iterations)` richtet sich wie in Abschnitt 5 beschrieben nach Rehurek, mit dem Unterschied, dass die Parameter der User-Eingabe entnommen sind und zuvor noch mit der Methode `arevalid(topics,below,above,chunksize,passes, iterations)` auf korrekte Angaben überprüft werden. Anschließend kann man sich die Visualisierung der Topics unter Verwendung von pyLDAvis im Browser anzeigen lassen. Zu guter Letzt hat man auf dem Endbildschirm die Möglichkeit, die GUI zu schließen, zurück zur Parameterauswahl zu gehen oder die 'docs' Datei zu löschen und so einen neuen Korpus wählen zu können.

# 8 Interpretation der Ergebnisse

## 8.1 Bedeutungen der Topics

Zuständigkeit: J.K., J.W.
Wie zu erwarten, beziehen sich viele der Begriffe in den einzelnen Topics auf Themen aus dem Bereich Politik oder Gesellschaft, so gibt es beispielsweise viele Bezüge zur Coronapolitik, bekannten Persönlichkeiten oder Anfang Mai stattfindenden Ereignissen, wie z.B. den Demonstrationen am Tag der Arbeit. Oftmals erschienen die Begriffe zunächst noch willkürlich, nach einer genaueren Betrachtung des Korpus stellte sich allerdings heraus, dass diese Begriffe aus einzelnen Tweets stammen, die besonders häufig retweetet wurden. Um nur einige Beispiel zu nennen: Begriffe wie 'Luftfilter', 'pandemisch', 'Gnadenlosigkeit' und 'weni' tauchten gleich in mehreren Topics auf. Wie sich herausstellte, standen diese Begriffe im starken Zusammenhang mit dem Tweet "Kein Impfstoff für Kinder, keine Luftfilter für Schulen: Corona hat mit pandemischer Gnadenlosigkeit offengelegt, wie weni[g junge Menschen zählen in Deutschland.]"(bzw. ähnlichen Varianten). Das Zitat stammt aus der Kolumne "Die deutsche Rentokratie, jetzt auch mit Corona-Topping" von Sascha Lobo, welche am 05. Mai im Spiegel erschien. Ebenso fanden sich als Reaktion auf einen Beitrag des ZDF Magazin Royale zum Thema Österreich zahlreiche Tweets und Retweets, die sich auf René Benko bezogen. Daneben tauchten auch Begriffe auf, die man zunächst nicht erwartet hätte. Hauptsächlich bezogen sich diese auf asiatische Serien, Schauspieler oder Sänger.
Die Interpretation der einzelnen Topics gestaltet sich hingegen als schwieriger, da nicht

alle Begriffe untereinander zusammenpassten. So bezog sich beispielsweise ein Topic auf René Benko, aber gleichzeitig auch auf Beiträge des Auschwitz Memorial. Ebenso gab es in fast jedem Topic Bezüge zur Pandemiepolitik, als auch zu Demonstrationen anlässlich des 1. Mais. Insgesamt lassen sich wiederkehrende Themen erkennen und interpretieren, die allerdings auf die verschiedenen Topics verteilt und miteinander vermischt sind.

## 8.2 How-To: Topics interpretieren

Zuständigkeit: S.O.

Die Interpretation von Topics, die beim Topic Modelling entstehen, ist keine einfache Angelegenheit. Wie im vorigen Abschnitt schon erwähnt wurde, sind Topics oft keine homogene Masse und bilden ein - für Menschen schwer interpretierbares - Kontinuum ab. Eine Möglichkeit, den Topics trotzdem wissenschaftlich wertvolle Bedeutung zuzuschreiben, ist die konsequente und strukturierte Analyse auf Basis einer Hypothese. Im Kontext dieses Projektes liegt der Fokus vor allem auf politischen Ereignissen. Demnach werden die Topics nach Wörtern durchforstet, die diese Hypothese stützen, andere können „ignoriert“ werden.

> [...], we can look at the words that have the highest weights for each topic — that is, those that make up the greatest share of the topic. We can also use other metrics, like mutual information, to compare the words in each topic against all of the other topics, and pick out the words that are most distinctive (Kessel 2018).

So kann festgehalten werden, dass für alle Topics aus dem Modell global zu beobachten ist, dass die generelle Richtung politisch bestimmt ist. Viele Topics enthalten unterschiedliche Schlagwörter zu Ereignissen aus der Politik — und bilden somit einen Teil des gesamten Spektrums ab. Aus dem Topic Modelling kann man *keine* genauen Themen für bestimmte Dokumente ziehen. Es lassen sich ungefähre Grundzüge bzw. Oberthemen identifizieren.

## 8.3 Weitere Problematiken: Language Identification von Twitter

Zuständigkeit: S.O.

Die Language Identification von Twitter, die auch von der Twitter API genutzt wird, hat

keine 100-prozentige Akkuratheit. Das heißt, die Sprachfilterung in den Suchanfragen bringt einem auch Tweets aus anderen Sprachen ins Ergebnis. Bei diesem Projekt führt dies dazu, dass die Topics Wörter aus anderen Sprachen beinhalten, unter anderem z.B. aus dem Portugiesischen. Da Twitter eine Plattform ist, auf der oftmals in verschiedenen Sprachen gleichzeitig getweetet wird, ist dies aber nicht verwunderlich und auch nicht unbedingt ungünstig: Die Topics spiegeln nicht nur das Tweet-Verhalten wieder, sondern auch die Methodik, die sich hinter Twitters Language Classification versteckt. Mehr dazu kann in der verlinkten Literatur nachvollzogen werden (TwitterBlog 2015).

# 9 Fazit

Das Ergebnis dieses Projekts ist, trotz des zunächst unerwarteten Outputs, durchaus positiv zu bewerten. Mit Hilfe verschiedener Skripte zum Beschaffen von Tweets, zum Vorprozessieren und zum Modellieren eines geeigneten LDA-Modells konnten wir die Topics aus den Tweets der ersten Maiwoche analysieren. Unsere in der Einleitung formulierte Hypothese, dass vor allem über Ereignisse getwittert wurde, die in der ersten Maiwoche geschehen sind, wird durch die besagten Topics bekräftigt. Themen rund um die pandemische und politische Lage dominieren die Topics, die das Modell ausgegeben hat. Einige Ausreißer sind in den Ergebnissen zu finden - dies sei jedoch auf die genannten, twitter-spezifischen Probleme zurückzuführen.

Verbesserungen könnten natürlich vorgenommen werden, unter anderem bereits beim Auswählen der Tweets. Die Twitter-interne Sprachidentifikation könnte um eine eigene Filterung ergänzt werden. So würden Tweets, die nicht in der gewünschten Sprache verfasst wurden, aussortiert werden. Das Ergebnis könnte dadurch noch besser interpretierbar werden.

# Literatur


Blei, David M., Ng, Andrew Y. und Jordan, Michael I. "Latent Dirichlet Allocation". en. In: *January 2003* 3 (2003), S. 993–1022.

Dimitrov, Dimitar et al. "TweetsCOV19 - A Knowledge Base of Semantically Annotated Tweets about the COVID-19 Pandemic". In: *CoRR* abs/2006.14492 (2020). arXiv: 2006.14492. URL: https://arxiv.org/abs/2006.14492.

Hong, Liangjie und Davison, Brian D. "Empirical Study of Topic Modeling in Twitter". In: *Proceedings of the First Workshop on Social Media Analytics*. SOMA '10. Washington D.C., District of Columbia: Association for Computing Machinery, 2010, S. 80–88. ISBN: 9781450302173. DOI: 10.1145/1964858.1964870. URL: https://doi.org/10.1145/1964858.1964870.

Kessel, Patrick van. *An intro to topic models for text analysis*. 2018. URL: https://medium.com/pew-research-center-decoded/an-intro-to-topic-models-for-text-analysis-de5aa3e72bdb.

Rehurek, Radim. *LDA Model*. Zuletzt aufgerufen: 25. Mai 2021. 2021. URL: https://radimrehurek.com/gensim/auto_examples/tutorials/run_lda.html.

Twitter. *Twitter API Documentation*. Zuletzt aufgerufen: 21. Mai 2021. 2021. URL: https://developer.twitter.com/en/docs.

TwitterBlog. *Evaluating language identification performance*. 2015. URL: https://blog.twitter.com/engineering/en_us/a/2015/evaluating-language-identification-performance.