Lerneinheit: word2vec mit Gensim

Mareike Schumacher [1]

1. Universität Regensburg

forTEXT

Thema:	word2vec	DOI:	10.48694/fortext.3816
Jahrgang:	1	Ausgabe:	10
Erscheinungsdatum:	30-10-2024	Erstveröffentlichung:	2023-04-20 auf fortext.net
Lizenz:	CC BY SA		open access

Allgemeiner Hinweis: Rot dargestellte Begriffe werden im Glossar am Ende des Beitrags erläutert. Alle externen Links sind auch am Ende des Beitrags aufgeführt.

Eckdaten des Lehrmoduls

- Anwendungsbezug: Frauen- und Männerrollen in Goethes Erzähltexten und Dramen
- Methode: word2vec
- Angewendetes Tool: Gensim
- Lernziele: Trainieren eines word2vec-Modells, einfache Abfragen (vgl. Query) und Vektorarithmetik, erstellen von Visualisierungen zum gesamten Korpus und zu einzelnen semantischen Feldern
- Dauer der Lerneinheit: 60-90 Minuten
- Schwierigkeitsgrad des Tools: mittel

Bausteine

- Anwendungsbeispiel: In dieser Lerneinheit untersuchen Sie ein Textkorpus bestehend aus Prosawerken und Dramen Goethes.
- Vorarbeiten: Was müssen Sie tun, bevor es losgehen kann? Lernen Sie, wie man Gensim und weitere Python-Pakete installiert
- Funktionen: Welche Vorverarbeitungsschrite sind wichtig, damit Sie ein Modell trainieren können? Lernen Sie, wie Sie ein Korpus einlesen, welche Vorverarbeitungsschritte auszuführen sind und wie Sie ein Modell trainieren. Lernen Sie außerdem einige Methoden von Gensim kennen und wie Sie Visualisierungen erstellen können. Welche Funktionen bieten Ihnen Gensim? Lernen Sie die einzelnen Komponenten des Tools kennen und lösen Sie Beispielaufgaben.
- Lösungen zu den Beispielaufgaben: Hat die Lerngruppe die Beispielaufgaben richtig gelöst? Hier finden Sie Antworten.
- Nachweise und weiterführende Literatur

1. Anwendungsbeispiel

In dieser Lerneinheit untersuchen Sie ein Textkorpus (vgl. Korpus) bestehend aus Prosawerken und Dramen Goethes. Mithilfe der Beispielabfragen nähern wir uns der Thematik genderspezifischer Darstellungen in diesem Korpus. word2vec ist ein Verfahren, bei dem jedes Wort in einem Textkorpus anhand seiner Kontexte in einen Vektor umgerechnet wird. Dadurch entsteht ein vieldimensionales Feld, innerhalb dessen Wörter verortet werden. Nah beieinander stehende Wörter stehen in ähnlichen Kontexten (vgl. Kollokation; Methodenbeitrag zu word2vec Schumacher (2024)). In dieser Lerneinheit verwenden wir dafür Gensim (Řehůřek und Sojka 2011), ein Python-Package, das neben word2vec auch für Topic Modeling und ähnliche Distant Reading-Verfahren eingesetzt werden kann (Schumacher und Akazawa 2024). Gensim verfügt nicht über eine grafische Nutzeroberfläche (vgl. GUI) und ist darum kein ausgesprochen niedrigschwelliges Text-Analyse-Tool. Ergänzend zu dieser Lerneinheit stellen wir Ihnen ein Jupyter-Notebook (Kluyver u. a. 2016) zur Verfügung, mit dem Sie dennoch mit nur wenigen Klicks und ohne selbst CODE schreiben zu müssen, word2vec durchführen können.

2. Vorarbeiten

Wir empfehlen, diese Lerneinheit mithilfe eines Jupyter Notebooks auszuführen. Ein solches Notebook ermöglicht es Ihnen, von uns vorbereitete kurze Code-Einheiten per Klick auszuführen. Unabhängig davon, ob Sie das Jupyter Notebook nutzen möchten oder die Funktionen lieber über eine Kommandozeile (vgl. Commandline) ausführen, installieren Sie bitte vorab das Programmbündel Anaconda (Anaconda Software Distribution 2020). Haben Sie Anaconda installiert, so können Sie dieses Programm auch nutzen, um die Installation von Gensim

durchzuführen. Starten Sie dafür Anaconda und wechseln Sie links im Menü zu „Environments“. Geben Sie dann oben rechts in die Suchleiste „Gensim“ ein und wählen Sie im Drop-Down-Menü oben in der Mitte „All“. Als Suchergebnis erscheint nun eine Zeile für „Gensim – Topic modeling for humans“ vor der links ein leerer Kasten steht (vgl. Abb. 1).

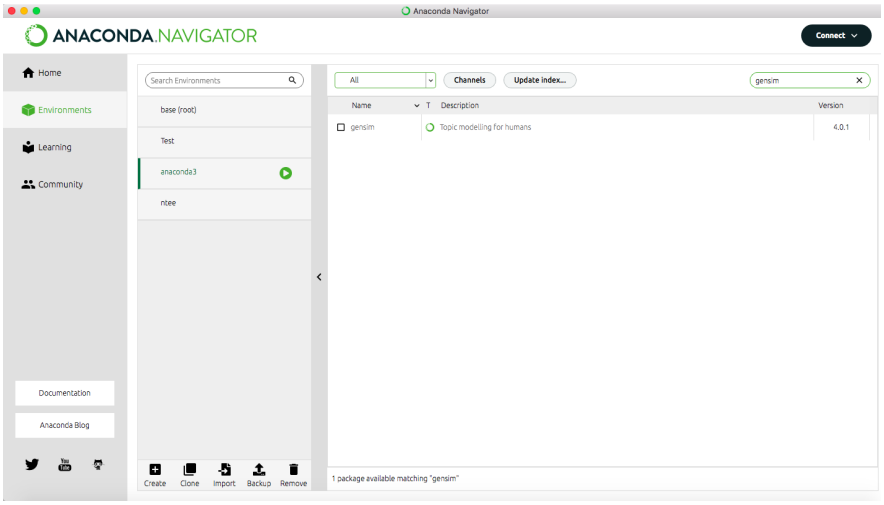

Abb. 1: Suche nach „Gensim“ innerhalb von Anaconda

Klicken Sie in diesen Kasten, sodass dort ein kleiner Pfeil erscheint und gehen Sie dann auf den Button „Apply“ unten rechts. Es kann einen Moment dauern, bis das Programm Ihnen anzeigt, was die Installation von Gensim bedeutet, bzw. welche anderen Packages zusätzlich installiert oder aktualisiert werden müssen. Sobald Ihnen die Liste angezeigt wird, können Sie mit „Apply“ die Installation bestätigen (vgl. Abb. 2).

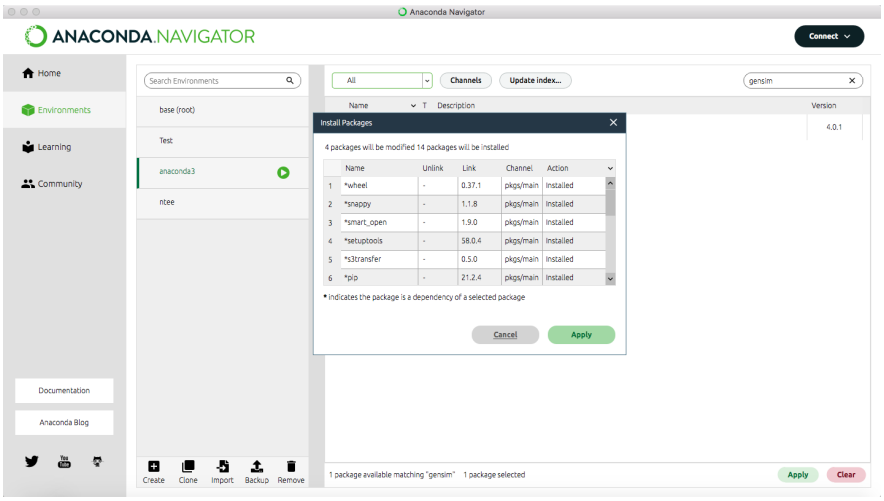

Abb. 2: Installation von Gensim innerhalb von Anaconda

Außer Gensim benötigen Sie für diese Lerneinheit noch die Python-Packages NumPy, SciPy, SKlearn, Matplotlib, NLTK und Plotly. Um zu prüfen, ob diese Pakete installiert werden müssen, klicken Sie oben rechts in der Suchleiste auf das Kreuz. Sie sehen nun eine lange Liste mit Packages. Suchen Sie nach den oben genannten und installieren Sie sie ebenso wie Gensim, wenn kein grüner Haken vorne in dem Kästchen davor steht (vgl. Abb. 3).

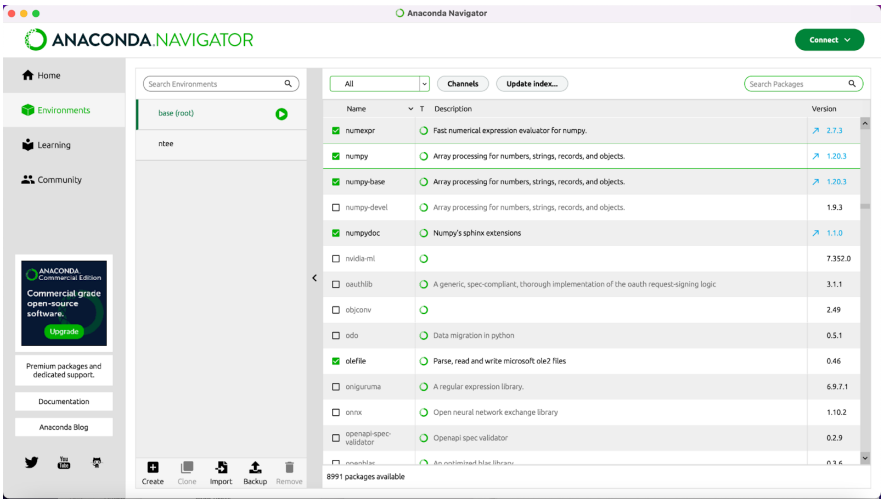

Abb. 3: Durchsuchen aller Packages in Anaconda

Neben technischen Vorarbeiten müssen Sie für word2vec auch das Textkorpus, das Sie untersuchen wollen, in eine Form bringen, mit der Gensim arbeiten kann. Das bedeutet, dass Sie alle Texte, die mit eingerechnet werden sollen, so aufbereiten müssen, dass alle Texte in einer Datei liegen und in jeder Zeile ein Satz steht. Diese Vorarbeit ist Teil des Jupyter Notebooks, das wir Ihnen zu dieser Lerneinheit bereit stellen. Für diese Lerneinheit können Sie ein von uns vorbereitetes Goethe-Korpus nutzen, das Sie über diesen Link herunterladen können. Es handelt sich dabei um das von Jan Horstmann zusammengestellte Korpus aus Prosatexten und Dramen Goethes. Für die Nutzung von word2vec kann es sinnvoll sein, weitere Vorverarbeitungsschritte (vgl. Preprocessing) am Textkorpus vorzunehmen. Zum Beispiel können Stoppwörter (vgl. Stoppwortliste), die nur wenig zum Inhalt von literarischen Texten beitragen (wie „der“, „die“, „das“), entfernt werden. Arbeiten Sie mit Dramen, so kann es sein, dass Sie kurze Sätze aus Ihrem Korpus ausschließen wollen, um Elemente wie „Akt 1“ oder „letzter Aufzug“, die das Modell als kurze Sätze interpretieren würde, nicht in die Berechnungen mit einzubeziehen, da es sich um gattungsspezigfische Gliederungselemente handelt. Solche Vorverarbeitungsschritte sind nicht Teil des Jupyter Notebooks zu dieser Lerneinheit. Wie Sie sie durchführen, erfahren Sie aber in unserer Lerneinheit *Preprocessing mit NLTK* (Schumacher und Vauth 2024). Natürlich würde eine solche Vorverarbeitung auch Sätze wie „Halt!“ aus einem Korpus eliminieren, kurze Sätze also, die tatsächlich inhaltlich zum Werk beitragen. Darum sind sämtliche Vorverarbeitungsschritte vorsichtig abzuschätzen. Je nach Fragestellung und betrachtetem Korpus kann es sinnvoll sein, mehrere word2vec-Modelle zu erstellen, die jeweils ein leicht anders vorbereitetes Korpus zur Basis haben. Auf diese Weise können Sie die Ergebnisse danach beurteilen, welche Phänomene bei der Analyse der unterschiedlich trainierten Modelle immer wieder auftauchen.
Wenn die Installation abgeschlossen und Sie das Korpus heruntergeladen haben, können Sie entweder das von uns vorbereitete Jupyter Notebook in unserem forTEXT-Github-Repository herunterladen oder die einzelnen Schritte über eine Kommandozeile durchführen. Die folgende Anleitung zeigt beide Wege. Wenn Sie mit der Kommandozeile Ihres Computers arbeiten wollen, können Sie nun direkt zum Punkt „3. Funktionen“ springen. Möchten Sie mit dem bereitgestellten Jupyter Notebook arbeiten, so wechseln Sie in Anaconda links im Menü von „Environments“ zu „Home“ und klicken dann in der Kachel „Jupyter Notebook“ auf den „Launch“-Button. Ihnen wird nun in Ihrem Browser Ihr Ordnersystem angezeigt. Gehen Sie zu dem Ordner, in dem Sie das Jupyter Notebook abgelegt haben, das Sie aus dem forTEXT-GitHub-Repository heruntergeladen haben. Klicken Sie auf den Dateinamen des Notebooks, so öffnet es sich in einem neuen Browser-Tab. Sie können nun den Anleitungen dort folgen. Die Antworten auf die dort gestellten Fragen finden Sie am Ende dieser Lerneinheit unter 4. Lösungen zu den Beispielaufgaben.

3. Funktionen

Wenn Sie das Jupyter Notebook nicht nutzen möchten, so öffnen Sie bitte in Anaconda eine Commandline. Gehen Sie dazu links im Menü zurück zu „Environments“ und klicken Sie auf den Play-Button hinter „Base (root)“. Gehen Sie dann auf „Open with Python“.

Basiseinstellungen laden und Modell trainieren

Als erstes laden Sie die Basiseinstellungen für Gensim. Mehr über Basiseinstellungen bei word2vec finden Sie in unserem *Methodeneintrag word2vec* (Schumacher 2024). Geben Sie dafür folgenden Code ein und klicken auf Enter:

```
import logging
logging.basicConfig(format='\%(asctime)s␣:␣\%(levelname)s␣:␣\%(message)s',
    ↪ level=logging.INFO)
import glob
import os
import nltk
```

Der Vorgang ist abgeschlossen, sobald „> > >“ unter dem eingegebenen Code erscheint. Als nächstes laden Sie Gensim. Geben Sie dazu unten stehenden Code ein und gehen Sie auf Enter.

```
from gensim.test.utils import datapath
from gensim import utils
```

Die nun folgenden vier Schritte dienen dazu, das Korpus in eine Form zu bringen, die für word2vec mit Gensim optimiert ist. Das heißt, dass Sie zunächst den Ordner definieren, in dem Ihr Korpus abgelegt ist, dann die Dateien daraus zu einer einzigen großen Datei zusammenfassen (merge), die zusammengefasste Datei so umformatieren, dass in jeder Zeile ein Satz steht und schließlich in den Sätzen jedes Wort als solches kennzeichnen (Tokenisierung (vgl. Type/Token) auf Satz- und Wortebene).

Schritt 1: Den Ordner definieren, in dem das Korpus abgelegt ist

Bitte ändern Sie nun für Schritt 1 in der Zeile

```
fileList = glob.glob("/Users/Testnutzer/Desktop/Goethe/\*.txt")
```

den Dateipfad und ersetzen ihn durch die Benennung in Ihrem eigenen Ordner-System. Haben Sie z.B. das von uns vorbereitete Goethe-Korpus in Ihrem Downloads-Ordner abgelegt, so wird Ihr Dateipfad wahrscheinlich so ähnlich lauten wie:

```
<div class="alert␣alert-block␣alert-warning">
<b>Beispiel:</b> corpus_path = datapath('/Users/Ihrname/Downloads/Goethe/')
</div>
```

Ist der Dateiname angepasst, so gehen Sie auf Enter. Um zu schauen, ob die Dateien korrekt in die Liste aufgenommen wurden, können Sie nun auf bekannte Weise unten stehenden Code ausführen. Dabei zeigt die 30 an, dass nur die ersten 30 Einträge der Liste angezeigt werden. Entfernen Sie die 30, so erhalten Sie eine Liste mit allen Einträgen.

```
fileList[:30]
```

Schritt 2: Die Dateien zu einer einzigen Textdatei zusammenfassen (merge)

Für Schritt 2 – das Zusammenfassen aller Dateien im Korpus zu einer einzigen Datei (merge) – führen Sie nun bitte folgenden Code aus.

```
Goethe_raw = []
for file_path in fileList:
    with open(file_path, encoding="utf8") as file:
        Goethe_raw.append(file.read())

Goethe_merged = '␣'.join(Goethe_raw)
```

Wenn Sie auch hier testen wollen, ob der Zwischenschritt erfolgreich war, können Sie dafür den folgenden Code nutzen. Die 300 in den eckigen Klammern definiert, dass nur die ersten 300 Zeichen angezeigt werden.

```
Goethe_merged[:300]
```

Schritt 3: Tokenisierung auf Satzebene

Um die Datei nun so aufzubereiten, dass in jeder Zeile nur ein einziger Satz steht, nutzen Sie den Tokenizer des Python-Packages NLTK, das viele Preprocessing-Operationen vereint (mehr dazu in der fortext-Lerneinheit *Preprocessing mit NLTK* (Schumacher und Vauth 2024)). Sie haben es ganz zu Anfang dieses Notebooks bereits geladen, sodass Sie nun ganz einfach den folgenden Code ausführen können.

```
Goethe_sentences = nltk.sent_tokenize(Goethe_merged, language='german')
```

Auch hier können Sie natürlich einen Test ausführen. Haben Sie den folgenden Code ausgeführt, werden die ersten fünf Sätze des Korpus angezeigt.

```
Goethe_sentences[:5]
```

Schritt 4: Tokenisierung auf Wortebene

Damit in Ihrem word2vec-Modell jedes Wort des Korpus berücksichtigt werden kann, müssen die Wörter zunächst im Text einzeln als solche markiert werden. Dazu nutzen Sie wieder den Tokenizer des NLTK-Packages, nur dass Sie dieses Mal auf Wortebene operieren.

```
Goethe_sentences_final = []
for sent in Goethe_sentences:
    Goethe_sentences_final.append(nltk.word_tokenize(sent, language='german'))
```

Auch hier können Sie mit der folgenden Zelle einen kleinen Test ausführen (erste 5 Sätze, in denen die Wörter jeweils mit einfachen Anführungsstrichen markiert sind).

```
Goethe_sentences_final[:5]
```

Word2vec-Modell trainieren

Nun können Sie ihr word2vec-Modell trainieren (vgl. Machine Learning). Geben Sie dazu in gleicher Weise nach und nach die folgenden Zeile ein und bestätigen Sie sie mit Enter:

```
import gensim.models
sentences = Goethe_sentences_final
model = gensim.models.Word2Vec(sentences=sentences)
```

Wenn Sie die letzte Zeile eingegeben und bestätigt haben, sehen Sie, wie Ihr Computer das Modell berechnet. Mit unserem Beispielkorpus geht das relativ schnell. Wenn Sie größere Datensätze nutzen, kann dieser Schritt ein wenig länger dauern.

Speichern des word2vec-Modells

Ihr Modell ist nun im Arbeitsspeicher abgelegt und Sie können Abfragen (vgl. Query) durchführen und Visualisierungen erstellen. Schließen Sie allerdings diese Sitzung, so wird das Modell nicht gespeichert. Um später noch einmal auf Ihr Modell zugreifen zu können oder um es mit anderen teilen zu können, speichern Sie es am besten ab. Mit unten stehender Codezeile wird das Modell in demselben Ordner abgespeichert, in dem sich auch das genutzte Korpus befindet.
Klicken Sie dazu nun in die unten stehende Box, sodass Ihr Cursor dort erscheint. Klicken Sie dann auf „Run“.

```
#import library
import joblib
filename = 'word2vec_model01_2202.sav'
joblib.dump(model, filename)
```

Bereits trainiertes, gespeichertes Modell laden

Möchten Sie (jetzt oder später) mit einem gespeicherten Modell weiterarbeiten, können Sie es mit unten stehender Codezeile aufrufen. Nicht vergessen: Derzeit arbeiten Sie bereits in dem Ordner, in dem auch Ihr Korpus und das Modell liegen. Möchten Sie das Modell in einer späteren Sitzung erneut laden, müssen Sie dazu evtl. die

Angabe ‚word2vec_model01_2202.sav' durch den kompletten Dateipfad ersetzen, unter dem sich Ihr Modell befindet.

```
loaded_model = joblib.load('word2vec_model01_2202.sav')

##Beispiel:
result = loaded_model.wv.most_similar (positive="Gretchen")
from IPython.display import display
display(result)
```

Abfragen erstellen

Sie haben nun Ihr word2vec-Modell trainiert und gespeichert. Sie können jetzt Abfragen erstellen und so herausfinden, welche Wörter in einem ähnlichen Kontext verwendet werden, wie ein Zielwort, für das Sie sich besonders interessieren. Die Ähnlichkeitsabfragen sind relativ, das heißt, es werden Ihnen zwar die ähnlichsten Wörter innerhalb des Korpus angezeigt, der Grad der Ähnlichkeit kann aber sehr unterschiedlich ausgeprägt sein. Wie hoch die Ähnlichkeit ist, können Sie anhand des Wertes ablesen, der hinter einem Wort angegeben ist. Diese Werte können sich im Bereich von +1 bis -1 befinden, einer Skala, die von sehr ähnlich (positiv, also +1) bis sehr unähnlich (negativ, also -1) erstreckt.
Im nächsten Schritt fragen Sie das Modell nach den Wörtern, die in ähnlichen Kontexten verwendet werden wie „Gretchen". Geben Sie dazu unten stehenden Code ein und gehe Sie auf Enter:

```
w1 = "Gretchen"
model.wv.most_similar (positive=w1)
```

Dasselbe können Sie auch mit dem Zielwort „Werther" machen:

```
w1 = "Werther"
model.wv.most_similar (positive=w1)
```

Aufgabe 1: Vergleichen Sie die Wörter, die dem Wort „Gretchen" am nächsten stehen und die Wörter, die in ihrer Verwendung „Werther" am meisten ähneln. Was fällt Ihnen auf?
Um noch ein konzeptuell ganz anderes Wort in den Fokus zu rücken, führen wir die gleiche Abfrage nun auch noch mit dem Zielwort „liebe" durch:

```
w1 = "Liebe"
model.wv.most_similar (positive=w1)
```

Aufgabe 2: Schauen Sie sich die Wörter an, die in ähnlichen Kontexten wie „Liebe" verwendet werden. Wie interpretieren Sie die Ergebnisse im Hinblick auf ein mögliches semantisches Feld der Liebe?

Vektorarithmetik

Durch die Umrechnung von Wörtern in Vektoren, also in Zahlenwerte, bietet word2vec die einzigartige Möglichkeit, mit Wörtern zu rechnen. Sie können z.B. den Vektor eines Wortes von dem eines anderen subtrahieren und dann Wörter anzeigen lassen, deren Vektoren dem Ergebnis sehr ähnlich sind. In unten stehendem Beispiel führen wir diese Vektorarithmetik mit den Wörtern Frau - Mann durch.
Geben Sie dazu unten stehenden Code ein und drücken Sie auf Enter.

```
w1 = ['Frau']
w2 = ['Mann']
model.wv.most_similar (positive=w1,negative=w2,topn=15)
```

Aufgabe 3: Schauen Sie sich die Ähnlichkeitswerte hinter den Wörtern an. Was fällt Ihnen – im Hinblick auf ein semantisches Feld des Weiblichen – auf?

Diese Vektorarithmetik funktioniert auch mit ganzen Wortgruppen. Indem eine ganze Gruppe von Wörtern zusammengefasst wird, ergibt sich ein semantisches Feld. Da für dieses semantische Feld mehr Daten im Modell enthalten sind, werden solche Abfragen häufig genauer als die Vektorarithmetik mit nur zwei Wörtern.
Um mehr über die Konzeptionierung des Weiblichen bei Goethe herauszufinden, stellen wir eine Wortgruppe zusammen, die mehrere Frauenrollen umfasst. Dann subtrahieren wir „Mann" davon und erhalten kontextähnliche Wörter, die ebenfalls zum semantischen Feld des Weiblichen beitragen.
Geben Sie nun diesen Code ein und bestätigen Sie mit Enter:

```
w1 = [”Frau”,'Maedchen','Mutter','Tante', 'Schwester']
w2 = ['Mann']
model.wv.most_similar (positive=w1,negative=w2,topn=15)
```

Aufgabe 4: Bewerten Sie das Ergebnis im Vergleich zu der oben stehenden Abfrage mit nur zwei Wörtern.

Auch die Gegenprobe kann interessant sein. Wenn Sie unten stehenden Code eingeben und mit Enter bestätigen, so wird eine Abfrage nach dem semantischen Feld des Männlichen durchgeführt:

```
w1 = [”Mann”,'Junge','Vater','Onkel','Bruder']
w2 = ['Frau']
model.wv.most_similar (positive=w1,negative=w2,topn=15)
```

Aufgabe 5: Schauen Sie sich das Ergebnis im Vergleich mit der Abfrage an, bei der die Wörter „Frau“, „Maedchen“, „Mutter“, „Tante“, „Schwester“ positiv und das Wort „Mann“ negativ mit einbezogen wurden. Was fällt Ihnen auf?
Ähnlichkeit zwischen zwei Wörtern bewerten

Mithilfe Ihres word2vec-Modells können Sie – wie Sie nun bereits wissen – den Grad der Ähnlichkeit von Wörtern messen. Mit der nächsten Abfrage bekommen Sie den Wert der Ähnlichkeit zwischen zwei Zielwörtern genannt. Geben Sie dazu unten stehenden Code ein und gehen Sie auf Enter.

```
model.wv.similarity(w1=”Gretchen”,w2=”Kind”)
```

Nun wissen Sie, welchen Ähnlichkeitswert Gretchen und Kind haben. Möchten Sie den Ähnlichkeitswert von Gretchen und Frau damit vergleichen, so führen Sie dieselbe Abfrage noch einmal wie folgt durch:

```
model.wv.similarity(w1=”Gretchen”,w2=”Frau”)
```

Natürlich darf auch die „Gretchenfrage“ nicht fehlen. Dem word2vec-Modell können Sie sie auf diese Weise stellen: Wie ähnlich sind die Wortkontexte von „Gretchen“ und „Religion“.

```
model.wv.similarity(w1=”Gretchen”,w2=”Religion”)
```

Aufgabe 6: „Gretchen“ und „Kind“, „Frau“ oder „Religion“, welche beiden Wörter werden am ähnlichsten verwendet?

Das Modell prüfen

Es gibt ein paar simple Wege, um zu prüfen, ob Ihr Modell tatsächlich Wörter zusammen gruppiert, die einem sinnvollen semantischen Feld zugeordnet werden können. Eine Methode dafür entspricht dem „eins von diesen Dingen gehört nicht zu den anderen“-Prinzip. Sie nennen dem Modell eine Reihe von Wörtern und zurück kommt eine Antwort auf die Frage, welches Wort in der Reihe nicht zu den anderen gehört:

```
model.wv.doesnt_match([”Mann”,”Krieger”,”Soldat”,”Kind”])
```

Findet das Modell das richtige Wort heraus, so zeigt dies, dass es etwas über die Konzepte der Wörter gelernt hat.

Das Modell visualisieren

Sie haben nun ein word2vec-Modell erstellt, einige Abfragen kennen gelernt und geprüft, ob das Modell sinnvolle semantische Felder ausmachen kann. Um Ihnen noch eine weitere Perspektive auf Ihr Modell zu zeigen, führen wir nun noch zwei grundlegende Formen der Visualisierung des word2vec-Modells durch. Bei beiden wird eine Reduktion der Dimensionen des Modells auf ein zweidimensionales Koordinatensystem durchgeführt. Dazu wird eine Methode namens t-SNE genutzt (Schumacher 2024). In diesem Koordinatensystem werden die Wörter im Korpus dargestellt. Nah beieinander stehende Wörter werden in ähnlichen Kontexten verwendet, weit voneinander entfernt stehende Wörter gehören nicht zu einem ähnlichen semantischen Feld. Wir werden nun zunächst das gesamte Modell visualisieren und dann nur einzelne semantische Felder.
Um das gesamte word2vec-Modell zu visualisieren, geben Sie nun folgenden Code ein und bestätigen Sie mit Enter.

```
from sklearn.decomposition import IncrementalPCA
from sklearn.manifold import TSNE
import numpy as np
```

```
def reduce_dimensions(model):
    num_dimensions = 2 # final num dimensions (2D, 3D, etc)
    vectors = np.asarray(model.wv.vectors)
    labels = np.asarray(model.wv.index_to_key)
    tsne = TSNE(n_components=num_dimensions, random_state=0)
    vectors = tsne.fit_transform(vectors)
    x_vals = [v[0] for v in vectors]
    y_vals = [v[1] for v in vectors]

    return x_vals, y_vals, labels

x_vals, y_vals, labels = reduce_dimensions(model)

def plot_with_plotly(x_vals, y_vals, labels, plot_in_notebook=True):
    from plotly.offline import init_notebook_mode, iplot, plot
    import plotly.graph_objs as go

    trace = go.Scatter(x=x_vals, y=y_vals, mode='text', text=labels)
    data = [trace]
    if plot_in_notebook:
        init_notebook_mode(connected=True)
        iplot(data, filename='word-embedding-plot')
    else:
        plot(data, filename='word-embedding-plot.html')

def plot_with_matplotlib(x_vals, y_vals, labels):
    import matplotlib.pyplot as plt
    import random
    random.seed(0)
    plt.figure(figsize=(12, 12))
    plt.scatter(x_vals, y_vals)
    indices = list(range(len(labels)))
    selected_indices = random.sample(indices, 25)
    for i in selected_indices:
        plt.annotate(labels[i], (x_vals[i], y_vals[i]))

    try:
        get_ipython()
    except Exception:
        plot_function = plot_with_matplotlib
    else:
        plot_function = plot_with_plotly

    plot_function(x_vals, y_vals, labels)
    plt.show()
```

Es müsste sich nun ein Fenster mit Ihrer Visualisierung öffnen. Ganz unten finden Sie interaktive Bedienelemente sowie die Möglichkeit, Ihre Visualisierung zu speichern. Evtl. müssen Sie die Visualisierung etwas verkleinern, um dieses Menü sehen zu können.

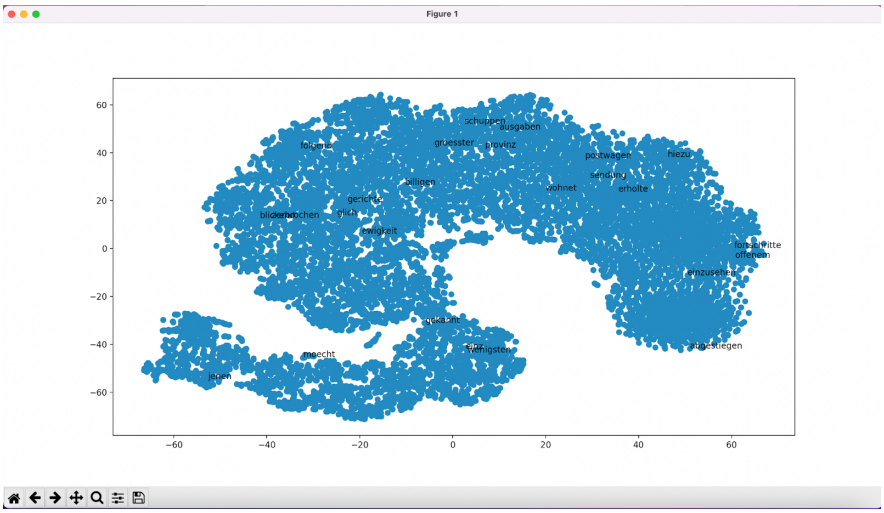

Abb. 4: Visualisierung des gesamten word2vec-Modells

Bei der nächsten Form der Visualisierung können Sie einzelne semantische Felder in unterschiedlichen Farben anzeigen lassen. Für unser Fallbeispiel betrachten wir die Begriffe „Frau“ und „Mann“. Die Graphik, die erstellt wird, hat den Titel „Genderdarstellungen bei Goethe“ und wird unter dem Namen „Gender_Goethe.png“ auf Ihrem Computer gespeichert.
Um die Grafik zu erstellen, geben Sie folgende Codes in drei Schritten ein:

```
##### 1. Schritt

embedding_clusters = []
word_clusters = []
for word in keys:
    embeddings = []
    words = []
    for similar_word, _ in model.wv.most_similar(word, topn=30):
        words.append(similar_word)
        embeddings.append(model.wv[similar_word])
    embedding_clusters.append(embeddings)
    word_clusters.append(words)


##### 2. Schritt

from sklearn.manifold import TSNE
import numpy as np

embedding_clusters = np.array(embedding_clusters)
n, m, k = embedding_clusters.shape
tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=3500,
    ↪  random_state=32)
embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embedding_clusters.
    ↪ reshape(n \* m, k))).reshape(n, m, 2)

##### 3. Schritt

import matplotlib.pyplot as plt
import matplotlib.cm as cm

def tsne_plot_similar_words(title, labels, embedding_clusters, word_clusters, a
    ↪ , filename=None):
    plt.figure(figsize=(16, 9))
    colors = cm.rainbow(np.linspace(0, 1, len(labels)))
    for label, embeddings, words, color in zip(labels, embedding_clusters,
        ↪ word_clusters, colors):
```

```
        x = embeddings[:, 0]
        y = embeddings[:, 1]
        plt.scatter(x, y, c=color, alpha=a, label=label)
        for i, word in enumerate(words):
            plt.annotate(word, alpha=0.5, xy=(x[i], y[i]), xytext=(5, 2),
                ↪ textcoords='offset␣points', ha='right', va='bottom', size=8)
            plt.legend(loc=4)
            plt.title(title)
            plt.grid(True)
        if filename:
            plt.savefig(filename, format='png', dpi=150, bbox_inches='tight')
        plt.show()

tsne_plot_similar_words('Genderdarstellungen␣bei␣Goethe', keys,
    ↪ embeddings_en_2d, word_clusters, 0.7,'Gender_Goethe1.png')
```

Auch hier öffnet sich wieder ein extra-Fenster mit der Visualisierung, die Sie interaktiv erkunden können. Ist das Fenster zu groß für Ihren Bildschirm, ziehen Sie es einfach etwas kleiner, um die Menüleiste unten zu sehen.

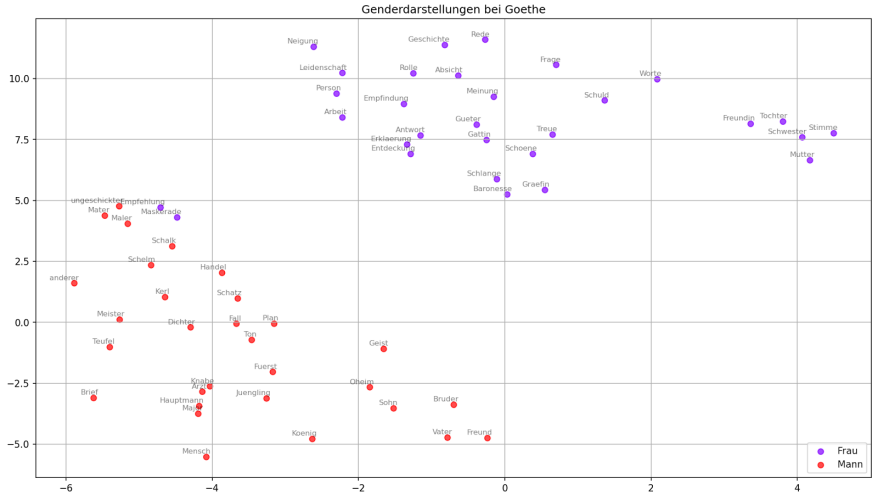

Abb. 5: Visualisierung von Wörtern aus dem Modell mit ähnlichen Kontexten wie „Frau“ und „Mann“

Aufgabe 7: Schauen Sie sich die Visualisierung der semantischen Felder der Wörter „Frau“ und „Mann“ an. Ist die Zuweisung, die im word2vec-Modell festgelegt ist, akkurat?

Sie können aber auch ganz andere Begriffe oder sogar eine andere Anzahl von Begriffen zum Ausgangspunkt Ihrer Visualisierung machen. Schauen Sie sich nun die Begriffe „Religion“, „Wissenschaft“, „Bildung“ und „Natur“ genauer an, indem Sie wie gehabt den folgenden (bereits entsprechend angepassten) Code in drei Schritten eingeben.

```
import gensim

keys = ['Religion','Wissenschaft','Bildung','Natur']
embedding_clusters = []
word_clusters = []

for word in keys:
    embeddings = []
    words = []
    for similar_word, _ in model.wv.most_similar(word, topn=30):
        words.append(similar_word)
        embeddings.append(model.wv[similar_word])
    embedding_clusters.append(embeddings)
    word_clusters.append(words)

from sklearn.manifold import TSNE
import numpy as np
```

```
embedding_clusters = np.array(embedding_clusters)
n, m, k = embedding_clusters.shape

tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=3500,
    ↪  random_state=32)

embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embedding_clusters.
    ↪ reshape(n \* m, k))).reshape(n, m, 2)

import matplotlib.pyplot as plt
import matplotlib.cm as cm

def tsne_plot_similar_words(title, labels, embedding_clusters, word_clusters, a
    ↪ , filename=None):
    plt.figure(figsize=(16, 9))
    colors = cm.rainbow(np.linspace(0, 1, len(labels)))
    for label, embeddings, words, color in zip(labels, embedding_clusters,
        ↪ word_clusters, colors):
        x = embeddings[:, 0]
        y = embeddings[:, 1]
        plt.scatter(x, y, c=color, alpha=a, label=label)
        for i, word in enumerate(words):
            plt.annotate(word, alpha=0.5, xy=(x[i], y[i]), xytext=(5, 2),
                ↪ textcoords='offset␣points', ha='right', va='bottom', size
                ↪ =8)

    plt.legend(loc=4)
    plt.title(title)
    plt.grid(True)

    if filename:
        plt.savefig(filename, format='png', dpi=150, bbox_inches='tight')

    plt.show()


tsne_plot_similar_words('Religion␣und␣Wissenschaft␣bei␣Goethe', keys,
    ↪ embeddings_en_2d, word_clusters, 0.7,
'Religion_Wissenschaft_Goethe.png')
```

Aufgabe 8: Schauen Sie sich nun die Visualisierung der semantischen Felder der abstrakteren Begriffe „Religion“, „Wissenschaft“, „Bildung“ und „Natur“ an. Was fällt Ihnen auf?

Aufgabe 9: Vergleichen Sie nun Ihre neuen mit den alten Ergebnissen auf Ihren Screenshots. Was fällt Ihnen auf?

4. Lösungen zu den Beispielaufgaben

Aufgabe 1: Vergleichen Sie die Ähnlichkeitswerte von „Gretchen“ und den ähnlichsten Wörtern und „Werther“ und den ähnlichsten Wörtern. Was fällt Ihnen auf?

Die Abfrage zu „Gretchen“ zeigt unter den am ähnlichsten verwendeten Wörtern hauptsächlich Figurenbezeichnungen. Das Wort „Gretchen“ wird also ähnlich verwendet wie andere Figurenreferenzen. Dabei fällt auf, dass es sich zumeist nicht um bekannte Hauptfiguren aus den Texten Goethes handelt, sondern meist um andere Nebenfiguren. Bei „Werther“ fällt hingegen auf, dass die Liste relativ viele Adjektive wie „Schoen“ oder „Still“ zeigt, deren Großschreibung darauf hindeutet, dass sie am Satzanfang stehen müssen. Es könnte also sein, dass das Modell hier darauf hindeutet, dass der Name „Werther“ vergleichsweise häufig genutzt wird, um einen Satz zu beginnen. Dagegen befindet sich auf der Liste mit „Amalia“ nur ein einziger anderer Figurennamen. Hier könnte sich ein Hinweis auf eine ähnlich gestaltete Figur finden.

Aufgabe 2: Schauen Sie sich die Wörter an, die in ähnlichen Kontexten wie „Liebe“ verwendet werden. Wie interpretieren Sie die Ergebnisse im Hinblick auf ein mögliches semantisches Feld der Liebe?

Es fällt auf, dass mit „Kraft“, „Freude“, „Treue“ und „Hoffnung“ vier Wörter auf der Liste stehen, die positive Emotionen oder Zustände ausdrücken. Mit „Not“, „Schuld“ und „Schmerzen“ stehen dem beinahe ebenso viele negativ konnotierte Ausdrücke gegenüber. Das semantische Feld der Liebe scheint hier also ambivalent ausgestaltet zu sein. Darüber hinaus fällt auf, dass mit „Tochter“ und „Freundin“ hier auch zwei weibliche Figurenreferenzen auftauchen.

Aufgabe 3: Schauen Sie sich die Ähnlichkeitswerte hinter den Wörtern an. Was fällt Ihnen – im Hinblick auf ein semantisches Feld des Weiblichen – auf?

Die Liste wird angeführt von „Mutter“ als einer besonders bedeutenden weiblichen Rolle. Weitere familiär geprägte weibliche Rollen, die sich auf der Liste finden, sind z.B. „Schwester“ und „Tochter“. Auch Beschreibungen implizierende Wörter wie „Schoene“ und „Empfindlichkeit“ sind enthalten, ebenso wie „Haende“ oder „Kleider“. Das semantische Feld des Weiblichen umfasst also familiäre Frauenrollen, ist eher implizit beschreibend und enthält Erwähnungen körperlicher Attribute wie „Haende“ oder „Stimme“ ebenso wie Hinweise auf die Kleidung.

Aufgabe 4: Bewerten Sie das Ergebnis im Vergleich zu der oben stehenden Abfrage mit nur zwei Wörtern!

Bei dieser Abfrage zeigen sich viel mehr Frauenrollen als bei der Abfrage mit nur zwei Wörtern. Enthalten sind z.B. „Freundin“, „Tochter“ und „Gattin“. Interessant ist hier, dass zur „Stimme“, die sich schon bei der Abfrage mit zwei Wörtern zeigte, nun auch die „Meinung“ hinzu kommt.

Aufgabe 5: Schauen Sie sich das Ergebnis im Vergleich mit der Abfrage an, bei der die Wörter „Frau“, „Maedchen“, „Mutter“, „Tante“, „Schwester“ positiv und „Mann“ negativ mit einbezogen wurden. Was fällt Ihnen auf?

Diese Liste zeigt so gut wie gar keine beschreibenden Wörter, sondern beinahe ausschließlich männliche Rollenbezeichnungen wie „Oheim“, „Sohn“ oder „Gatte“. Viele dieser Rollen zeigen eine familiäre Prägung. Andere wie „Schelm“, „Kerl“, „Schatz“ oder „Teufel“ gehen mit positiver oder negativer Bewertung einher.

Aufgabe 6: „Gretchen“ und „Kind“, „Frau“ oder „Religion“, welche beiden Wörter werden am ähnlichsten verwendet? Was ergibt der Vergleich der Ähnlichkeitswerte von „Gretchen“, „Frau“ und „Mann“ und „Faust“ mit „Religion“?

Der Ähnlichkeitswert der Worte „Gretchen“ und „Kind“ ist höher als der von „Gretchen“ und „Frau“. Am höchsten ist aber die Ähnlichkeit zwischen „Gretchen“ und „Religion“. Die „Gretchenfrage“ wird also auch vom word2vec-Modell des Goethe-Korpus als wichtig für die Konzeption der Gretchen-Figur ausgemacht. Der Ähnlichkeitswert zwischen „Frau“ und „Religion“ ist niedriger als der zwischen „Gretchen“ und „Religion“. Noch niedriger ist der Wert zwischen „Mann“ und „Religion“. Bewertet das Modell aber die Ähnlichkeit der Begriffskontexte von „Faust“ und „Religion“, so ist diese zwar geringer als zwischen „Gretchen“ und „Religion“, aber doch deutlich höher als zwischen „Mann“ und „Religion“. Das Modell könnte hier erfasst haben, dass die Religionsthematik für den gesamten Faust-Text und dessen Figuren von Bedeutung ist.

Aufgabe 7: Schauen Sie sich die Visualisierung der semantischen Felder der Wörter „Frau“ und „Mann“ an. Ist die Zuweisung, die im word2vec-Modell festgelegt ist, akkurat?

Die semantischen Felder von „Frau“ und „Mann“ werden als zwei relativ genau getrennte Felder im Vektorraum angezeigt. Es können aber Fehlzuweisungen, wie z.B. „Maedchen“ als Wort des semantischen Feldes von Mann auftreten. Auch gibt es ein paar Begriffe, die das Label „Frau“ aufweisen, die aber sehr nah am semantischen Feld von „Mann“ angezeigt werden. Dies können z.B. „Verordnungen“ oder „Schreibtafel“ sein.

Aufgabe 8: Schauen Sie sich nun die Visualisierung der semantischen Felder der abstrakteren Begriffe „Religion“, „Wissenschaft“, „Bildung“ und „Natur“ an. Was fällt Ihnen auf?

Die Felder der Begriffe „Religion“, „Wissenschaft“ und „Bildung“ überlagern sich stark. Das Feld des Wortes „Natur“ steht dagegen etwas abseits und weist weniger Überschneidungen auf. Eine mögliche Deutung wäre hier, dass die Konzepte von Religion, Bildung und Wissenschaft in Goethes Erzähltexten und Dramen ähnlich gezeichnet werden. Das würde auch erklären, warum der Ähnlichkeitswert von „Faust“ und „Religion“ vergleichsweise hoch ist. Hier könnte die Idee zum Tragen kommen, dass die Wissenschaft, an die er glaubt, der Religion konzeptuell ähnlich dargestellt wird.

Aufgabe 9: Vergleichen Sie nun Ihre neuen mit den alten Ergebnissen auf Ihren Screenshots. Was fällt Ihnen auf?

Alle Ergebnisse zeigen eine Variation der ersten Abfrage-Outputs. Zum Teil weisen die Listen mit den Ähnlichkeitswerten sogar andere Begriffe auf.
Das liegt daran, dass bei diesem Verfahren ein Faktor der Randomisierung mit einfließt. Das word2vec-Modell wird jedes Mal in einem erneuten Lernprozess erstellt, wenn das Notebook neu gestartet wird. Dabei werden leicht variierende Parameter genutzt. Ergebnisse von word2vec-Abfragen sind also nicht reproduzierbar. Ein word2vec-Modell stellt immer nur eine mögliche Perspektive (von vielen) auf ein Korpus dar. Am besten machen Sie also mehrere Durchläufe, wenn Sie das Verfahren nutzen, und versuchen genau darauf zu achten, welche Phänomene sich immer wieder zeigen, also stabil bleiben.

Externe und weiterführende Links

- Goethe-Korpus: https://web.archive.org/save/https://github.com/janhorstmannn/goethe-prose-drama (Letzter Zugriff: 06.10.2024)

Bibliographie

Anaconda Software Distribution. 2020. Anaconda Documentation. https://docs.anaconda.com.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, u. a. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, hg. von F. Loizides und B. Schmidt, 87–90. IOS Press.

Řehůřek, Radim und Petr Sojka. 2011. Gensim-python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3, Nr. 2.

Schumacher, Mareike. 2024. Methodenbeitrag: word2vec. Hg. von Evelyn Gius. *forTEXT* 1, Nr. 10. word2vec (30. Oktober). doi: 10.48694/fortext.3815, https://fortext.net/routinen/methoden/word2vec-1.

Schumacher, Mareike und Mari Akazawa. 2024. Toolbeitrag: Gensim. Hg. von Evelyn Gius. *forTEXT* 1, Nr. 10. word2vec (30. Oktober). doi: 10.48694/fortext.3817, https://fortext.net/tools/tools/gensim.

Schumacher, Mareike und Michael Vauth. 2024. Lerneinheit: Preprocessing mit NLTK. Hg. von Evelyn Gius. *forTEXT* 1, Nr. 2. Korpusbildung (12. Juni). doi: 10.48694/fortext.3809, https://fortext.net/routinen/lerneinheiten/preprocessing-mit-nltk.

Glossar

Annotation Annotation beschreibt die manuelle oder automatische Hinzufügung von Zusatzinformationen zu einem Text. Die manuelle Annotation wird händisch durchgeführt, während die (teil-)automatisierte Annotation durch Machine-Learning-Verfahren durchgeführt wird. Ein klassisches Beispiel ist das automatisierte PoS-Tagging (Part-of-Speech-Tagging), welches oftmals als Grundlage (Preprocessing) für weitere Analysen wie Named Entity Recognition (NER) nötig ist. Annotationen können zudem deskriptiv oder analytisch sein.

Browser Mit Browser ist in der Regel ein Webbrowser gemeint, also ein Computerprogramm, mit dem das Anschauen, Navigieren auf, und Interagieren mit Webseiten möglich wird. Am häufigsten genutzt werden dafür Chrome, Firefox, Safari oder der Internet Explorer.

Close Reading Close Reading bezeichnet die sorgfältige Lektüre und Interpretation eines einzelnen oder weniger Texte. Close Reading ist in der digitalen Literaturwissenschaft außerdem mit der manuellen Annotation textueller Phänomene verbunden (vgl. auch Distant Reading als Gegenbegriff).

CODE Der Code, oder auch Programmcode/ Maschinencode, bezieht sich auf eine Sammlung von Anweisungen, die durch verschiedene Programmiersprachen wie Java, Python oder C realisiert werden können. Für die Ausführung der Anweisungen wird der Code durch einen Compiler oder einen Interpreter in die Maschinensprache, einen Binärcode, des Computers übersetzt.

Commandline Die Commandline (engl. *command line interface* (CLI)), auch Kommandozeile, Konsole, Terminal oder Eingabeaufforderung genannt, ist die direkteste Methode zur Interaktion eines Menschen mit einem Computer. Programme ohne eine grafische Benutzeroberfläche (GUI) werden i. d. R. durch Texteingabe in die Commandline gesteuert. Um die Commandline zu öffnen, klicken Sie auf Ihrem Mac „cmd“ + „space“, geben „Terminal“ ein und doppelklicken auf das Suchergebnis. Bei Windows klicken Sie die Windowstaste + „R“, geben „cmd.exe“ ein und klicken Enter.

Distant Reading Distant Reading ist ein Ansatz aus den digitalen Literaturwissenschaften, bei dem computationelle Verfahren auf häufig große Mengen an Textdaten angewandt werden, ohne dass die Texte selber gelesen werden. Meist stehen hier quantitative Analysen im Vordergrund, es lassen sich jedoch auch qualitative Metadaten quantitativ vergleichen. Als Gegenbegriff zu *Close Reading* wurde der Begriff insbesondere von Franco Moretti (2000) geprägt.

GUI GUI steht für *Graphical User Interface* und bezeichnet eine grafische Benutzeroberfläche. Ein GUI ermöglicht es, Tools mithilfe von grafischen Schaltflächen zu bedienen, um somit beispielsweise den Umgang mit der Commandline zu umgehen.

HTML HTML steht für *Hypertext Markup Language* und ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente. HTML-Dokumente werden von Webbrowsern dargestellt und geben die Struktur und Online-Darstellung eines Textes vor. HTML-Dateien können außerdem zusätzliche Metainformationen enthalten, die auf einer Webseite selbst nicht ersichtlich sind.

Hyperparameter Hyperparameter beziehen sich auf externe, anpassbare Einstellungen, die genutzt werden um den Lernprozess zu kontrollieren und zu beeinflussen (zu modellinternen Parametern siehe Parameter).

Sie sind unabhängig vom Datensatz und beziehen sich beispielsweise auf Einstellungen wie Anzahl der Iterationen, Größe der Datensätze oder Kontextfenster.

Kollokation Als Kollokation bezeichnet man das häufige, gemeinsame Auftreten von Wörtern oder Wortpaaren in einem vordefinierten Textabschnitt.

Korpus Ein Textkorpus ist eine Sammlung von Texten. Korpora (Plural für „das Korpus“) sind typischerweise nach Textsorte, Epoche, Sprache oder Autor*in zusammengestellt.

LDA LDA steht für *Latent Dirichlet Allocation* und ist ein generatives, statistisches Wahrscheinlichkeitsmodell, welches zum Topic Modeling angewendet werden kann. Bei der LDA werden auf Grundlage eines Wahrscheinlichkeitsmodells Wortgruppen aus Textdokumenten erstellt. Dabei wird jedes Dokument als eine Mischung von verborgenen Themen betrachtet und jedes Wort einem Thema zugeordnet. Wortreihenfolgen und Satzzusammenhänge spielen dabei keine Rolle.

Lemmatisieren Die Lemmatisierung von Textdaten gehört zu den wichtigen Preprocessing-Schritten in der Textverarbeitung. Dabei werden alle Wörter (Token) eines Textes auf ihre Grundform zurückgeführt. So werden beispielsweise Flexionsformen wie „schneller“ und „schnelle“ dem Lemma „schnell“ zugeordnet.

Machine Learning Machine Learning, bzw. maschinelles Lernen im Deutschen, ist ein Teilbereich der künstlichen Intelligenz. Auf Grundlage möglichst vieler (Text-)Daten erkennt und erlernt ein Computer die häufig sehr komplexen Muster und Gesetzmäßigkeiten bestimmter Phänomene. Daraufhin können die aus den Daten gewonnen Erkenntnisse verallgemeinert werden und für neue Problemlösungen oder für die Analyse von bisher unbekannten Daten verwendet werden.

Markup Language Markup Language bezeichnet eine maschinenlesbare Auszeichnungssprache, wie z. B. HTML, zur Formatierung und Gliederung von Texten und anderen Daten. So werden beispielsweise auch Annotationen durch ihre Digitalisierung oder ihre digitale Erstellung zu Markup, indem sie den Inhalt eines Dokumentes strukturieren.

Metadaten Metadaten oder Metainformationen sind strukturierte Daten, die andere Daten beschreiben. Dabei kann zwischen administrativen (z. B. Zugriffsrechte, Lizenzierung), deskriptiven (z. B. Textsorte), strukturellen (z. B. Absätze oder Kapitel eines Textes) und technischen (z. B. digitale Auflösung, Material) Metadaten unterschieden werden. Auch Annotationen bzw. Markup sind Metadaten, da sie Daten/Informationen sind, die den eigentlichen Textdaten hinzugefügt werden und Informationen über die Merkmale der beschriebenen Daten liefern.

Named Entities Eine Named Entity (NE) ist eine Entität, oft ein Eigenname, die meist in Form einer Nominalphrase zu identifizieren ist. Named Entities können beispielsweise Personen wie „Nils Holgerson“, Organisationen wie „WHO“ oder Orte wie „New York“ sein. Named Entities können durch das Verfahren der Named Entity Recognition (NER) automatisiert ermittelt werden.

Parameter Im Kontext von Machine-Learning-Modellen handelt es sich bei (Modell-)Parametern um modellinterne Konfigurationsvariablen, die anhand des Trainingssatzes bestimmt werden (zu modellexternen Parametern siehe Hyperparameter). Als Parameter werden einerseits Aspekte benannt, die den Lernprozess bestimmen und andererseits solche, die dabei erlernt werden. Die Werte der Parameter ergeben sich aus dem Datensatz selbst. Werte solcher Parameter können beispielsweise die Gewichtungen in neuronalen Netzwerken sein, also welche Aspekte im Trainingsprozess besonders einflussreich sind (z. B. können Wörter im direkten Umfeld eines Zielwortes als wichtiger bewertet werden also solche, die weit von diesem entfernt stehen) oder etwa wie die Gewichtung (also die Reihenfolge) der einzelnen Wörter innerhalb der Topics beim Topic Modeling.

POS PoS steht für *Part of Speech*, oder „Wortart“ auf Deutsch. Das PoS- Tagging beschreibt die (automatische) Erfassung und Kennzeichnung von Wortarten in einem Text und ist of ein wichtiger Preprocessing-Schritt, beispielsweise für die Analyse von Named Entities.

Preprocessing Für viele digitale Methoden müssen die zu analysierenden Texte vorab „bereinigt“ oder „vorbereitet“ werden. Für statistische Zwecke werden Texte bspw. häufig in gleich große Segmente unterteilt (*chunking*), Großbuchstaben werden in Kleinbuchstaben verwandelt oder Wörter werden lemmatisiert.

Query *Query* bedeutet „Abfrage“ oder „Frage“ und bezeichnet eine computergestützte Abfrage zur Analyse eines Textes. Um Datenbestände zu durchsuchen, werden Abfragesprachen eingesetzt, die *Queries* (Anfragen) an den Datenbestand senden. So bilden alle möglichen Queries zusammen die *Query Language* eines Tools.

Stoppwortliste Stoppwörter sind hochfrequente Wörter, meist Funktionswörter, die, aufgrund ihrer grammatisch bedingten Häufigkeit, beispielsweise die Ergebnisse von inhaltlichen oder thematischen Analysen verzerren können. Deshalb werden diese Wörter, gesammelt in einer Stoppwortliste, bei digitalen Textanalysen meist nicht berücksichtigt.

Topic Modeling Das Topic Modeling ist ein statistisches, auf Wahrscheinlichkeitsrechnung basierendes, Verfahren zur thematischen Exploration größerer Textsammlungen. Das Verfahren erzeugt „Topics“ zur Abbildung häufig gemeinsam vorkommender Wörter in einem Text. Für die Durchführung können verschiedene Algorithmen und Modelle wie das LDA verwendet werden.

Type/Token Das Begriffspaar „Type/Token“ wird grundsätzlich zur Unterscheidung von einzelnen Vorkommnissen (Token) und Typen (Types) von Wörtern oder Äußerungen in Texten genutzt. Ein Token ist also ein konkretes Exemplar eines bestimmten Typs, während ein Typ eine im Prinzip unbegrenzte Menge von Exemplaren (Token) umfasst.
Es gibt allerdings etwas divergierende Definitionen zur Type-Token-Unterscheidung. Eine präzise Definition ist daher immer erstrebenswert. Der Satz „Ein Bär ist ein Bär.“ beinhaltet beispielsweise fünf Worttoken („Ein“, „Bär“, „ist“, „ein“, „Bär“) und drei Types, nämlich: „ein“, „Bär“, „ist“. Allerdings könnten auch vier Types, „Ein“, „ein“, „Bär“ und „ist“, als solche identifiziert werden, wenn Großbuchstaben beachtet werden.