



BORIS THOME

NATURAL LANGUAGE PROCESSING WORKSHOP

ÜBERSICHT

- ▶ Vorverarbeitung von Textdaten
- ▶ Anwendungsbeispiele
 - ▶ Keyword Extraction
 - ▶ Part-Of-Speech Tagging
 - ▶ Dependency Parsing
 - ▶ Named-Entity Recognition
 - ▶ Kosinus-Ähnlichkeit
 - ▶ Sentiment Analyse

ZIELSETZUNG

- ▶ Überblick über das Thema Sprachverarbeitung
- ▶ Grundlegende Techniken der NLP-Pipeline vermitteln
- ▶ Anwendungsgebiete des Natural Language Processing aufzeigen
- ▶ Relevanz von Preprocessing bei vielen NLP-Aufgaben verdeutlichen

VORVERARBEITUNG VON TEXTDATEN (PREPROCESSING)

- ▶ Preprocessing hängt von Aufgabe und Textart ab
 - ▶ Sprache: Deutsch, Englisch oder mehrsprachig?
 - ▶ z.B. Social Media Texte vs. Zeitungstexte
 - ▶ Extrahieren von Schlüsselwörtern oder Klassifikationsaufgabe?
- ▶ Trägt zur Verbesserung der Datenqualität bei
- ▶ Bessere Qualität und leichtere Interpretation der Auswertungen
- ▶ Datenreduktion kann zu Zeitersparnis führen

GROß- / KLEINSCHREIBUNG (UPPERCASING / LOWERCASING)

- ▶ Beispiel:
 - ▶ „Es macht Spaß zu studieren.“
 - ▶ „Das Studieren gefällt mir.“
- ▶ Daher: Vereinheitlichung von Textdaten mittels Lowercasing / Uppercasing
- ▶ Problem: Informationsverlust
 - ▶ z.B. Erkennung von Nomen in deutscher Sprache über Großschreibung

STOPPWÖRTER (STOP WORDS)

- ▶ Definition: **Stoppwörter** nennt man in der Informationsrückgewinnung bzw. im **Information Retrieval** Wörter, die bei einer **Volltextindexierung** nicht beachtet werden, da sie sehr häufig auftreten und gewöhnlich keine Relevanz für die Erfassung des **Dokumentinhalts** besitzen. (<https://de.wikipedia.org/wiki/Stopwort>)
- ▶ Frage:
 - ▶ Welche Stoppwörter gibt es in der deutschen Sprache?

ENTFERNEN VON STOPPWÖRTERN (STOP WORD REMOVAL)

- ▶ Mögliche Antworten:
 - ▶ Bestimmte Artikel (der, die, das)
 - ▶ Unbestimmte Artikel (einer, eine, ein)
 - ▶ Konjunktionen (und, oder, doch, weil)
- ▶ Stoppwörter werden oft im Preprocessing herausgefiltert
- ▶ Grund: Sie tragen nicht zur semantischen Bedeutung von Texten bei, treten aber sehr häufig auf!

ENTFERNEN VON STOPPWÖRTERN (STOP WORD REMOVAL)

- ▶ Implementierung mittels Wortlisten (z.B. Python NLTK)
- ▶ Iteration über alle Wörter im Text
- ▶ Vergleich mit vorgefertigter Wortliste
- ▶ Wird ein Stoppwort aus der Liste erkannt, so wird es aus dem Text entfernt
- ▶ Wortlisten können variieren
- ▶ Anpassung möglich:
 - ▶ Sehr häufig auftretende Wörter im Textkorpus können ebenfalls als Stoppwörter interpretiert werden

ENTFERNEN VON STOPPWÖRTERN (STOP WORD REMOVAL)

```
import nltk
from nltk.corpus import stopwords
all_stop_words = stopwords.words('german')
print(stopwords.words('german'))
```

[3] ✓ 0.4s Python

... ['aber', 'alle', 'allem', 'allen', 'aller', 'alles', 'als', 'also', 'am', 'an', 'ander', 'andere', 'anderem', 'anderen', 'anderer', 'anderes', 'andernd', 'andern', 'anderr', 'anders', 'auch', 'auf', 'aus', 'bei', 'bin', 'bis', 'bist', 'da', 'damit', 'dann', 'der', 'den', 'des', 'dem', 'die', 'das', 'dass', 'daß', 'derselbe', 'derselben', 'denselben', 'desselben', 'demselben', 'dieselbe', 'dieselben', 'dasselbe', 'dazu', 'dein', 'deine', 'deinem', 'deinen', 'deiner', 'deines', 'denn', 'derer', 'dessen', 'dich', 'dir', 'du', 'dies', 'diese', 'diesem', 'diesen', 'dieser', 'dieses', 'doch', 'dort', 'durch', 'ein', 'eine', 'einem', 'einen', 'einer', 'eines', 'einig', 'einige', 'einigem', 'einigen', 'einiger', 'einiges', 'einmal', 'er', 'ihn', 'ihm', 'es', 'etwas', 'euer', 'eure', 'eurem', 'euren', 'eurer', 'eures', 'für', 'gegen', 'gewesen', 'hab', 'habe', 'haben', 'hat', 'hatte', 'hatten', 'hier', 'hin', 'hinter', 'ich', 'mich', 'mir', 'ihr', 'ihre', 'ihrem', 'ihren', 'ihrer', 'ihres', 'euch', 'im', 'in', 'indem', 'ins', 'ist', 'jede', 'jedem', 'jeden', 'jeder', 'jedes', 'jene', 'jenem', 'jenen', 'jener', 'jenes', 'jetzt', 'kann', 'kein', 'keine', 'keinem', 'keinen', 'keiner', 'keines', 'können', 'könnte', 'machen', 'man', 'manche', 'manchem', 'manchen', 'mancher', 'manches', 'mein', 'meine', 'meinem', 'meinen', 'meiner', 'meines', 'mit', 'muss', 'musste', 'nach', 'nicht', 'nichts', 'noch', 'nun', 'nur', 'ob', 'oder', 'ohne', 'sehr', 'sein', 'seine', 'seinem', 'seinen', 'seiner', 'seines', 'selbst', 'sich', 'sie', 'ihnen', 'sind', 'so', 'solche', 'solchem', 'solchen', 'solcher', 'solches', 'soll', 'sollte', 'sondern', 'sonst', 'über', 'um', 'und', 'uns', 'unsere', 'unserem', 'unseren', 'unser', 'unseres', 'unter', 'viel', 'vom', 'von', 'vor', 'während', 'war', 'waren', 'warst', 'was', 'weg', 'weil', 'weiter', 'welche', 'welchem', 'welchen', 'welcher', 'welches', 'wenn', 'werde', 'werden', 'wie', 'wieder', 'will', 'wir', 'wird', 'wirst', 'wo', 'wollen', 'wollte', 'würde', 'würden', 'zu', 'zum', 'zur', 'zwar', 'zwischen']

```
print(len(all_stop_words))
```

[4] ✓ 0.5s Python

... 232

LEMMATISIERUNG (LEMMATIZATION)

- ▶ Reduktion der flektierten Wortform auf die Grundform des Wortes
- ▶ Führt zur Vereinheitlichung der Daten:
 - ▶ Beispiel: „Spielte“, „Spielt“, „Spielten“ => „Spielen“
- ▶ Lemmatisierung meist mithilfe von Wörterbüchern (Lookup-Based)
- ▶ Problem: Wörterbuch/Liste enthält in der Regel nicht alle Wörter
- ▶ Umgang mit unbekannten Wörtern?
 - ▶ Flektierte Form unverändert lassen
 - ▶ Zusätzliche Regeln definieren

LEMMATISIERUNG MIT SPACY

```
import spacy

text = "Heute präsentiere ich einen spannenden Vortrag zum Thema Sprachverarbeitung."
nlp = spacy.load('de_core_news_sm') # lade deutsches Sprachmodell
lemmatized = []
document = nlp(text)
for word in document:
    print(word.lemma_)
```

[5] ✓ 3.3s

... heute
präsentieren
ich
ein
spannend
Vortrag
zu
Thema
Sprachverarbeitung
—

STAMMFORMREDUKTION (WORD STEMMING)

- ▶ Verschiedene morphologische Formen eines Wortes werden auf Wortstamm zurückgeführt
 - ▶ „playing“ => „play“
- ▶ Basiert auf Porter-Algorithmus (regelbasiertes Stemming):
 - ▶ Führt eine Menge von Verkürzungsregeln aus
 - ▶ Regeln werden so lange angewandt, bis das Wort eine Minimalanzahl von Silben aufweist
- ▶ Ursprünglich für die englische Sprache entworfen, jedoch auch auf andere Sprachen anwendbar
- ▶ Mehr Flexibilität als Lemmatisierung, da kein Lookup durchgeführt wird

STAMMFORMEN [HTTPS://WWW.NLTK.ORG/HOWTO/STEM.HTML](https://www.nltk.org/howto/stem.html) LEDUKTION (NLTK-IMPLEMENTIERUNG)

```
from nltk.stem.porter import *

text = "Learning things about natural language processing is very interesting."
porter_stemmer = PorterStemmer()

document = nlp(text)
for word in document:
    print(porter_stemmer.stem(word.text))
```

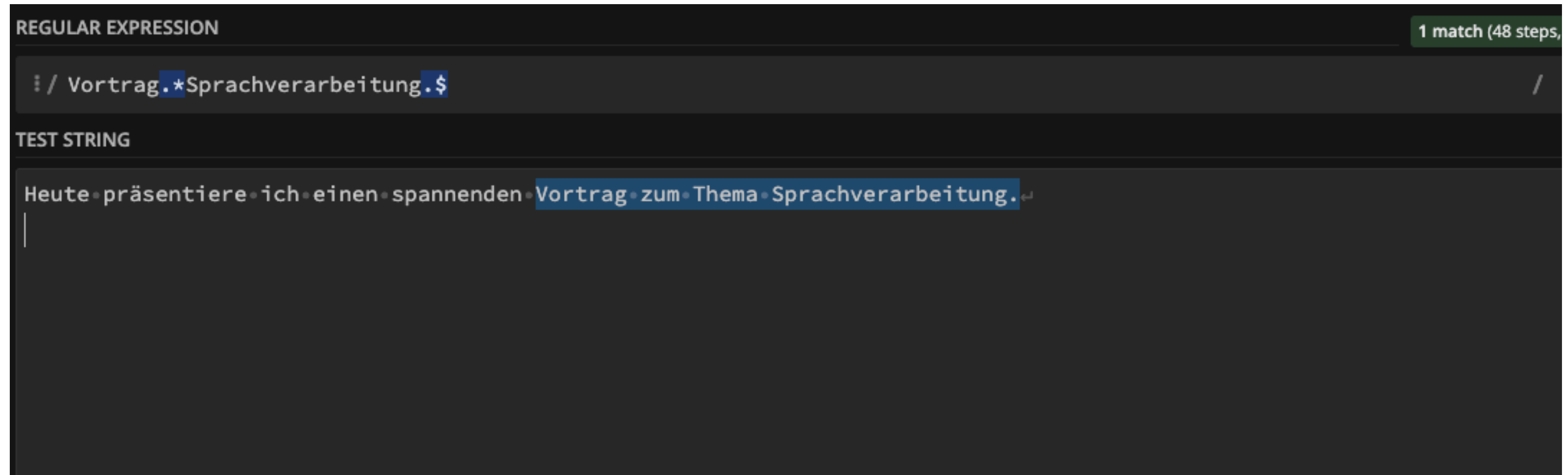
[14] ✓ 0.5s

... learn
thing
about
natur
languag
process
is
veri
interst
.

REGULAR EXPRESSIONS (REGEX)

- ▶ Beschreibung von Mengen von Zeichenketten mithilfe syntaktischer Regeln
- ▶ Konzept stammt aus der theoretischen Informatik
- ▶ Suche komplexer Patterns (Sequenzen) möglich
- ▶ Suchen und Ersetzen Funktion
- ▶ Spezifische Syntax notwendig
- ▶ In viele Programmiersprachen integrierbar
- ▶ Hilfsseiten zum Testen von Regex Strings (<https://regex101.com/>)

REGULAR EXPRESSIONS (BEISPIEL)



The screenshot shows a web-based regular expression testing interface. At the top, the label 'REGULAR EXPRESSION' is on the left, and '1 match (48 steps)' is on the right. Below this, the regular expression pattern is entered in a text field: `/Vortrag.*Sprachverarbeitung.$/`. The label 'TEST STRING' is on the left, and the test string is entered in a larger text area: `Heute präsentiere ich einen spannenden Vortrag zum Thema Sprachverarbeitung.`. The words 'Vortrag' and 'Sprachverarbeitung.' in the test string are highlighted with a blue background, indicating a successful match.

- ▶ Zeichenkette, die mit „Vortrag“ beginnt und mit „Sprachverarbeitung.“ Endet
- ▶ Zwischen den beiden Worten dürfen beliebige Zeichen stehen „.*“

RECHTSCHREIBKORREKTUR

- ▶ Verschiedene Bibliotheken mit Rechtschreibkorrektur-Funktion
- ▶ Kann zur Datenbereinigung beitragen
- ▶ Kann je nach Dokumentart sehr sinnvoll sein
- ▶ Basieren auf Wortlisten mit Lookup Funktion
- ▶ Auswahl passender Bibliothek spielt wichtige Rolle
- ▶ Bibliothek muss auf entsprechende Sprache eingestellt werden

KEYWORD EXTRACTION (SCHLÜSSELWORT-FINDUNG)

- ▶ Extrahieren von Schlüsselwörtern, Phrasen, Sätzen, Abschnitten etc.
- ▶ Maß für die „Relevanz“ solcher Terme
- ▶ Idee: Vorkommenshäufigkeit als Indikator für wichtige Terme
 - ▶ Je häufiger ein Term vorkommt, desto relevanter ist er für das Dokument
- ▶ Reicht dies bereits aus, um Schlüsselwörter zu finden?
 - ▶ => Python Notebook

KEYWORD EXTRACTION – VORKOMMENSHÄUFIGKEIT (BEISPIEL)

- ▶ Probleme:
 - ▶ Viele irrelevante Terme („ist“, „Das“)
 - ▶ Spezifische Terme, die selten vorkommen sind besonders relevant, stehen jedoch weit unten im Ranking
 - ▶ Gleicher Term kommt mehrfach vor („Die“, „die“)
- ▶ Lösung:
 - ▶ Preprocessing + Bewertungsschema überdenken

VORKOMMENSHÄUFIGKEIT – INVERSE DOKUMENTHÄUFIGKEIT (TF-IDF)

- ▶ Kombination aus Vorkommenshäufigkeit und Inverser Dokumenthäufigkeit
- ▶ $\text{idf}(t) = \log \frac{N}{\sum_{D:t \in D} 1}$ (Inverse Dokumenthäufigkeit)
 - ▶ (N : Anzahl aller Dokumente im Korpus, t : Term, D : Dokument)
- ▶ $\text{tf.idf}(t, D) = \text{tf}(t, D) \cdot \text{idf}(t)$ (Term frequency - Inverse Document Frequency)
 - ▶ Multiplikative Verknüpfung mit Vorkommenshäufigkeit
- ▶ => Python Notebook

PART-OF-SPEECH TAGGING (POS-TAGGING)

- ▶ Zuordnung von Wörtern und Satzzeichen eines Textes zu Wortarten
- ▶ Mögliche POS-Tags sind: „Nomen“, „Verb“ und „Adjektiv“
 - ▶ Feinere Unterteilungen sind jedoch möglich!
- ▶ POS-Tagging ist eine sprachabhängige Aufgabe
- ▶ Implementierung ebenfalls mit Bibliotheken wie spacy oder NLTK möglich
 - ▶ Basiert auf statistischen Merkmalen der jeweiligen Sprache:
 - ▶ z.B.: Nach bestimmten Artikeln „Der/Die/Das“ folgen oft Nomen

PART-OF-SPEECH TAGGING – SPACY

- ▶ spaCy's Liste enthält 19 POS-Tags
- ▶ Aufteilung in spezifischere Tags wie z.B. „interjection“
- ▶ Klasse „X“ für alle Tokens, die keiner anderen Klasse zugeordnet werden können

Spacy POS Tags List

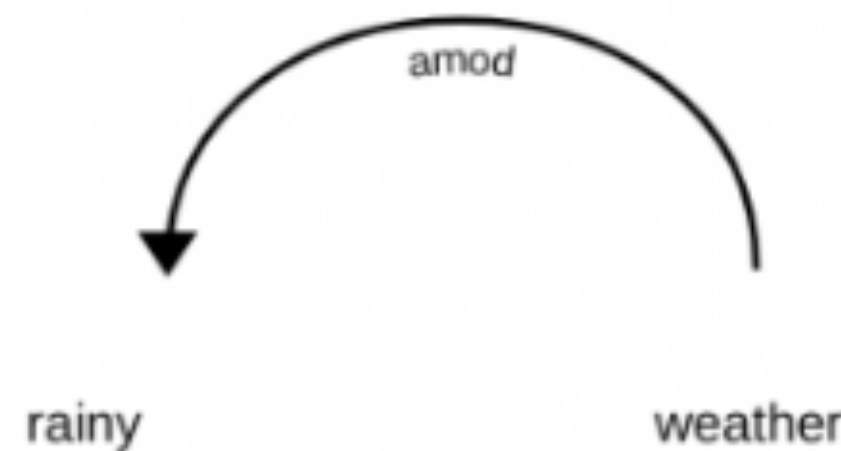
Every token is assigned a POS Tag in Spacy from the following list:

POS	DESCRIPTION	EXAMPLES
ADJ	adjective	*big, old, green, incomprehensible, first*
ADP	adposition	*in, to, during*
ADV	adverb	*very, tomorrow, down, where, there*
AUX	auxiliary	*is, has (done), will (do), should (do)*
CONJ	conjunction	*and, or, but*
CCONJ	coordinating conjunction	*and, or, but*
DET	determiner	*a, an, the*
INTJ	interjection	*psst, ouch, bravo, hello*
NOUN	noun	*girl, cat, tree, air, beauty*
NUM	numeral	*1, 2017, one, seventy-seven, IV, MMXIV*
PART	particle	*'s, not,*
PRON	pronoun	*I, you, he, she, myself, themselves, somebody*
PROPN	proper noun	*Mary, John, London, NATO, HBO*
PUNCT	punctuation	*., (,), ?*
SCONJ	subordinating conjunction	*if, while, that*
SYM	symbol	*\$, %, §, ©, +, -, ×, ÷, =, :), 😊*
VERB	verb	*run, runs, running, eat, ate, eating*
X	other	*sfpkdspxmsa*
SPACE	space	

DEPENDENZGRAMMATIK (DEPENDENCY PARSING)

- ▶ Beschreibt Abhängigkeiten verschiedener Worte eines Satzes zueinander
- ▶ Abhängigkeiten beschreiben semantische Relationen

- ▶ Beispiel:



- ▶ Im Beispiel modifiziert das Wort „rainy“ die Bedeutung des Wortes „weather“
 - ▶ Somit erhält es den sogenannten „amod“-Tag (adjectival modifier)
- ▶ => Python Notebook

NAMED-ENTITY RECOGNITION (NER)

- ▶ Automatische Identifikation und Klassifikation von Eigennamen
- ▶ Ein Eigenname ist dabei eine Folge von Wörtern, die eine real existierende Entität beschreibt (z.B. ein Firmenname, ein Ort, eine Zeitangabe, Personen etc.)
- ▶ Verschiedene Möglichkeiten:
 - ▶ Basierend auf Wortlisten
 - ▶ Regelbasierte Modelle (z.B. Groß-/Kleinschreibung etc. berücksichtigen)
 - ▶ Machine Learning Modelle
- ▶ => Python Notebook

KOSINUS ÄHNLICHKEIT (COSINE SIMILARITY)

- ▶ Maß für die Ähnlichkeit von zwei Vektoren
- ▶ Kosinus des Winkels zwischen den zwei Vektoren wird berechnet:

$$\text{Kosinus-Ähnlichkeit} = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{\sum_{i=1}^n a_i \cdot b_i}{\sqrt{\sum_{i=1}^n (a_i)^2} \cdot \sqrt{\sum_{i=1}^n (b_i)^2}}.$$

- ▶ Ist das Ergebnis 1, so sind die zwei Vektoren gleich
- ▶ Je geringer das Ergebnis, desto mehr unterscheiden sich die Dokumente voneinander
- ▶ => Python Notebook

SENTIMENT-ANALYSE / TONALITÄT

- ▶ Ziel: Geäußerte Haltung als positiv oder negativ zu erkennen
- ▶ Anwendungsfall:
 - ▶ Erkennung von Tonalität in Produktcommentaren
 - ▶ Analyse von Tweets zu bestimmten Themen mittels Twitter API
- ▶ Sentiment-Tools sollten auf Dokumentart und Sprache zugeschnitten sein

SENTIMENT-ANALYSE MIT VADER

- ▶ VADER (Valence Aware Dictionary and sEntiment Reasoner):
 - ▶ Lexikon- und regelbasiertes Tool zur Sentiment Analyse
 - ▶ Lexikon- und regelbasiertes Tool zur Sentiment Analyse
- ▶ VADER ist für die Untersuchung von Social Media Texten optimiert:
 - ▶ Berücksichtigt Satzzeichen, Umgangssprache, Smileys, Abkürzungen usw.
- ▶ Für die englische Sprache ausgelegt

SENTIMENT-ANALYSE MIT VADER

- ▶ Berechnung eines Compound Score anhand von Scores jeden Wortes im Dokument
- ▶ Berücksichtigung bestimmter Regeln (wie z.B. Negation etc.)
- ▶ Normalisierung: Compound zwischen -1 und +1
- ▶ Compound Score:
 - ▶ Positives Sentiment: Compound Score ≥ 0.05
 - ▶ Neutrales Sentiment: Compound Score > -0.05 und < 0.05
 - ▶ Negatives Sentiment: Compound Score ≤ -0.05

SENTIMENT-ANALYSE MIT VADER

NAMED ENTITY RECOGNITION

SENTIMENT ANALYSIS

I really love you! :)

NAMED ENTITY RECOGNITION

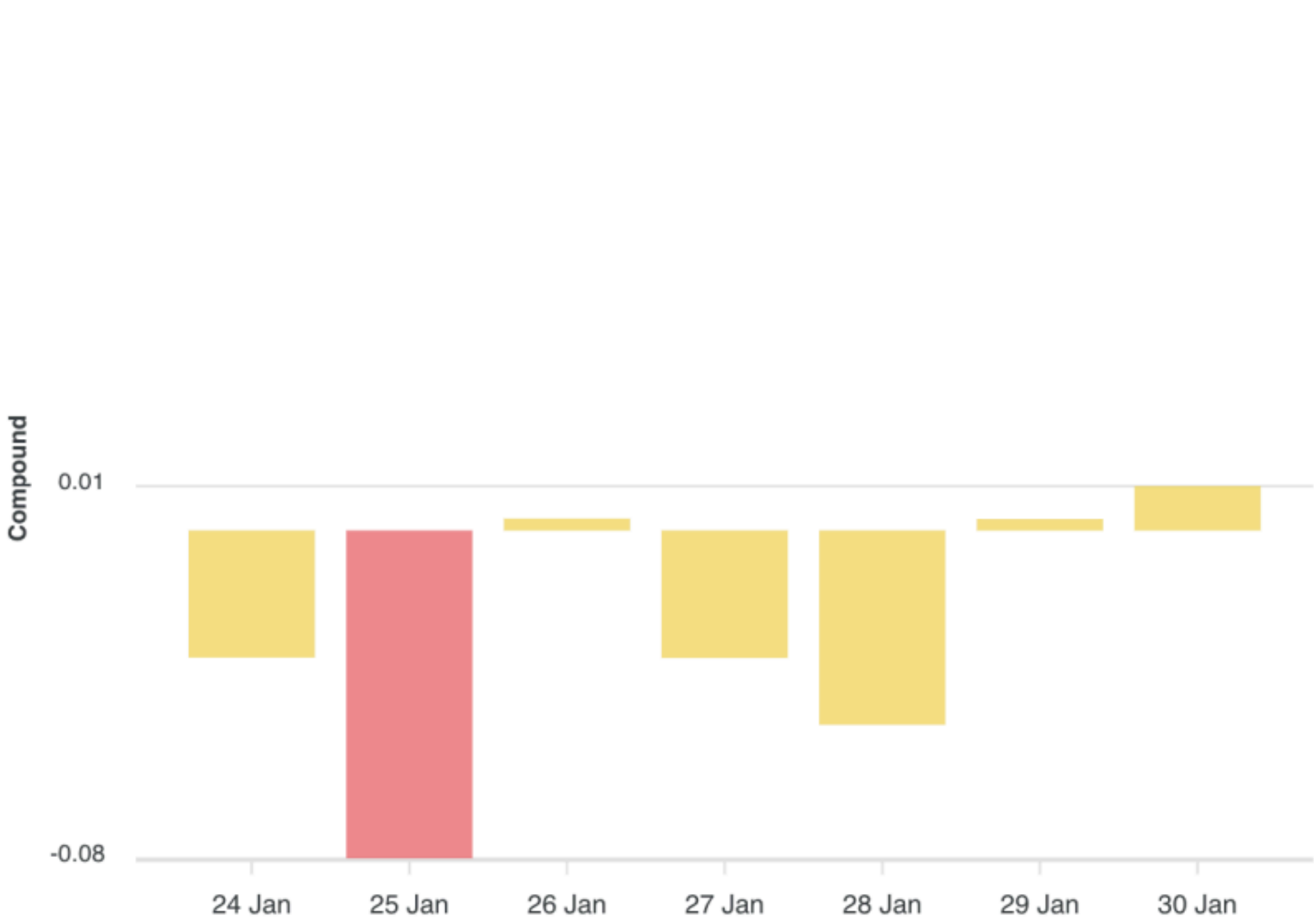
SENTIMENT ANALYSIS

:)

NAMED ENTITY RECOGNITION

SENTIMENT ANALYSIS

I hate you! :(





Q & A