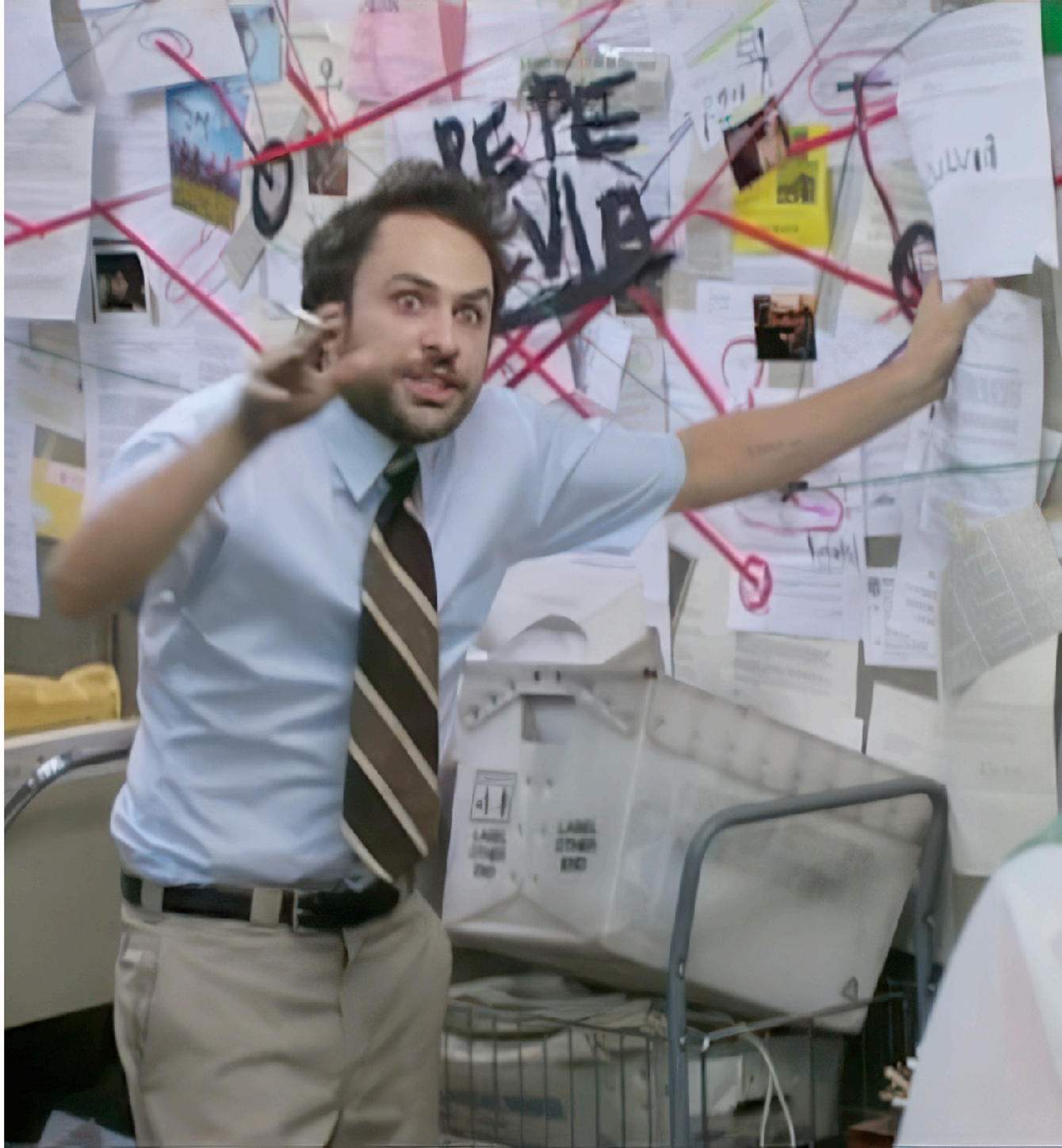
Text Analytics Projekt

Abschlusspräsentation

Colinho22 , 12.12.25

Agenda

1. Aufbau Projekt
2. Daten
3. Trainingsmethoden
4. Modell Fight Club 🤫



Mit den Skills aus dem Modul möchte ich eine eigne Content Classification App bauen. Dazu nehme ich Daten aus mehren Quellen und trainiere mit vier Methoden Modelle und teste deren Fähigkeit als Klassifizierungsmodell.

But why though?

Anwendungsbeispiele:

- **Content Moderation** - Automatische Kategorisierung von Posts
- **News Aggregation** - Artikel automatisch in Sektionen (Sport, Business, etc.) einordnen
- **Email Filtering** - Über Spam hinaus: Kategorisierung nach Dringlichkeit/ Abteilung

Problemstellungen:

- Manuelle Klassifikation kostet Zeit & skaliert schlecht
- Verschiedene Use Cases brauchen unterschiedliche Trade-offs (Geschwindigkeit vs Genauigkeit)

Projektstruktur

1. Datenset ✅
2. ngrams ✅
3. embeddings ✅
4. lstm ✅
5. Transformer ✅
6. GUI & Test ✅

https://github.com/Colinho22/text-analytics-project

```
text-analytics-project
|
|__data/
|    |_combined_dataset.csv
|    |_processed_datasets/..
|
|__notebooks/ (.ipynb analysis)
|
|__src/
|    |_ data pipeline
|    |_<method>_model.py
|    |_<method>_train.py
|
|__models/
|    |_ trained models
|
|__results/
|    |_output from notebooks (images, .txt, ...)
|
|__demo/
     |_app.py
```

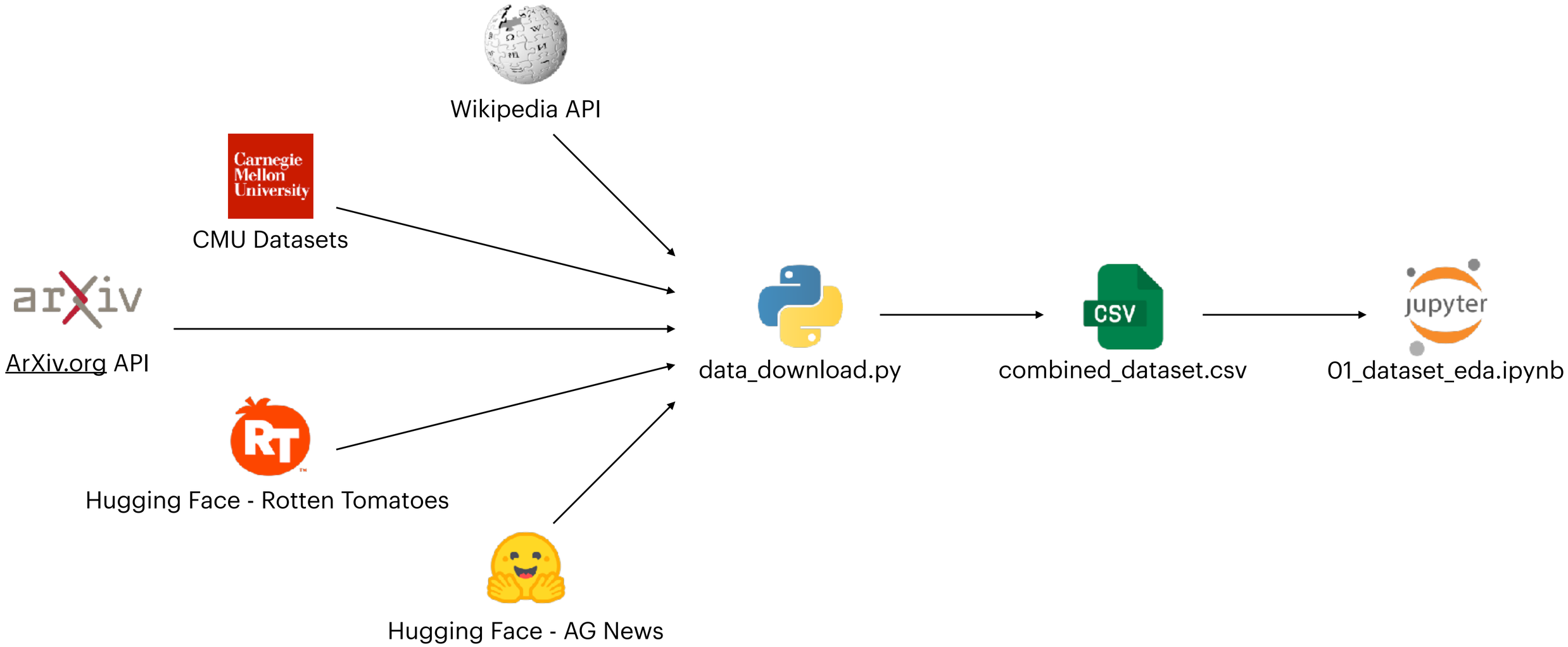
Anmerkungen zur Codebase

- Python 3.10.11
- Requirements.txt
- Apple MacBook Air, M3, 24GB RAM, MacOS 26.1
  - Transformer-Training optimiert für Apple M-Chip Architektur (CPU → GPU)
- Transformer *model.safetensors* (Training Weights) nicht in Codebase!
  - Option A: Transformer lokal neu trainieren (*transformer_train.py*)
  - Option B: *model.safetensors* separat von G-Drive herunterladen und lokal Einfügen

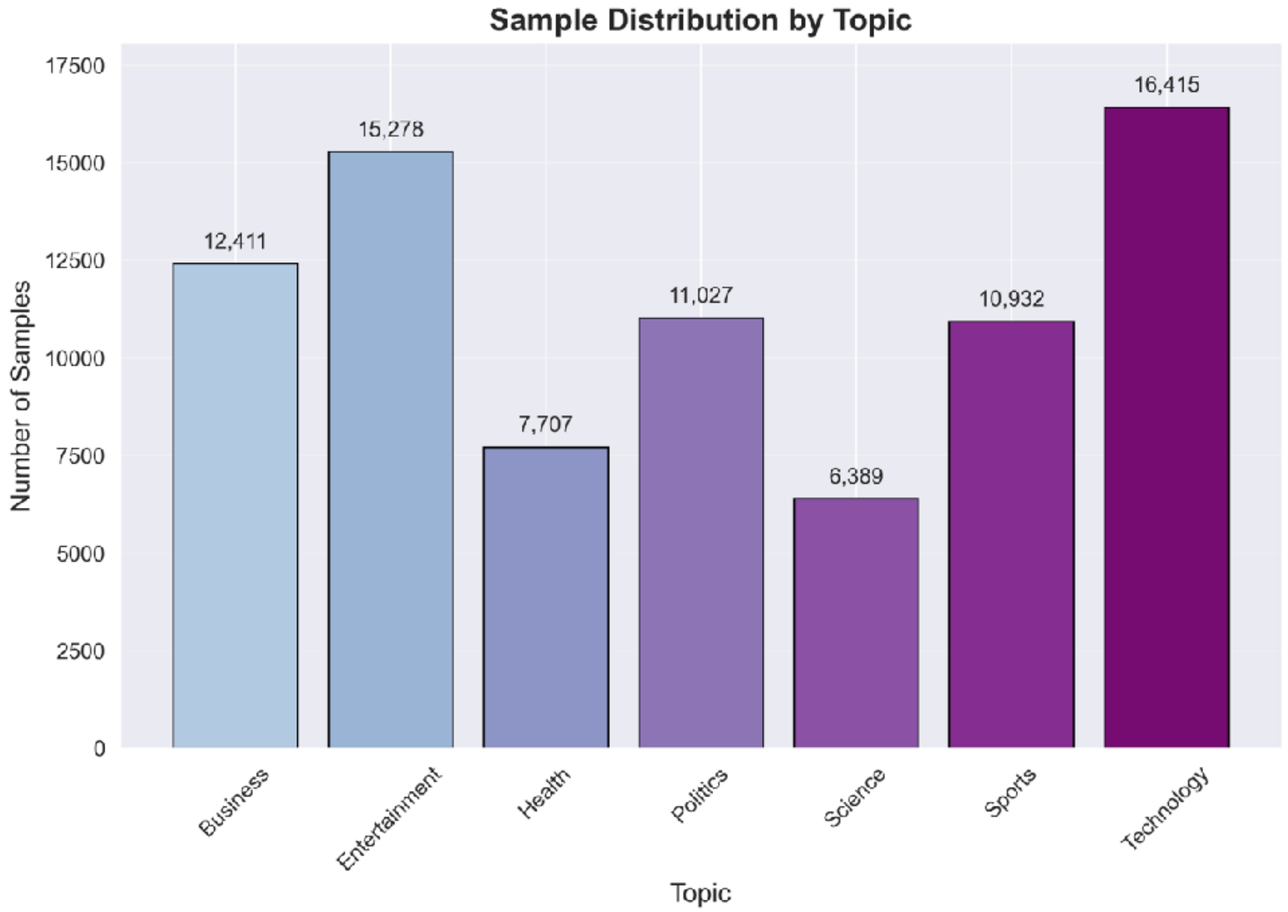
Daten

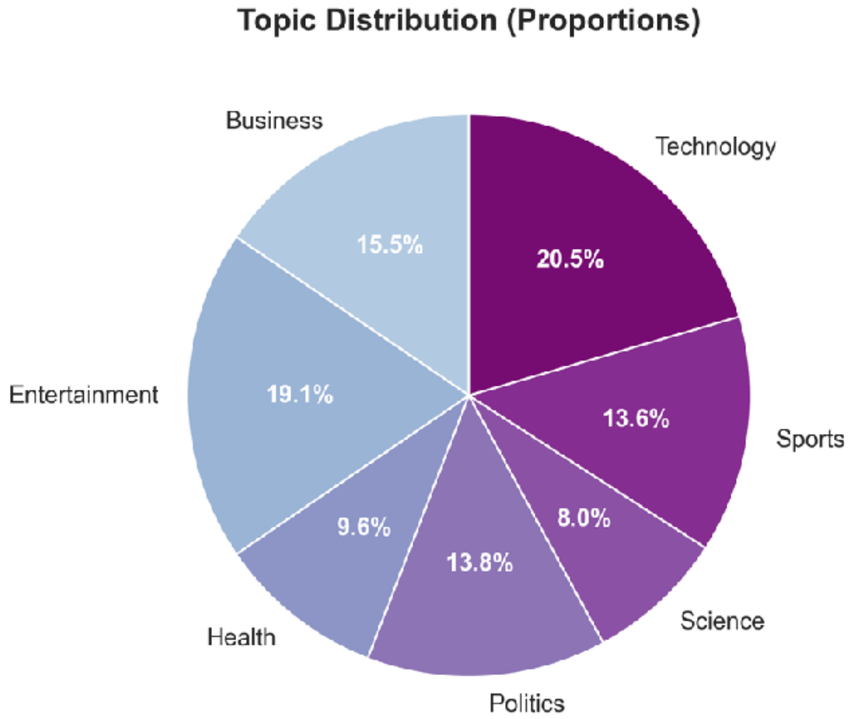
Aggregation & Analyse

Aggregation

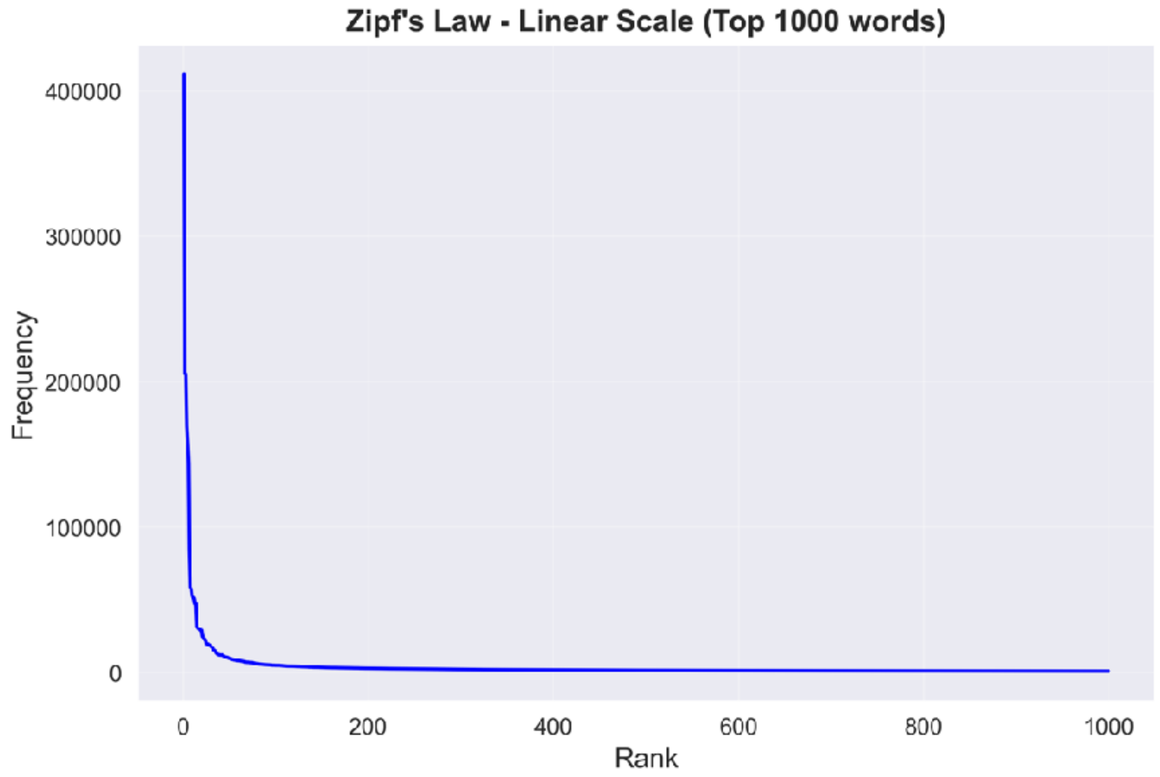


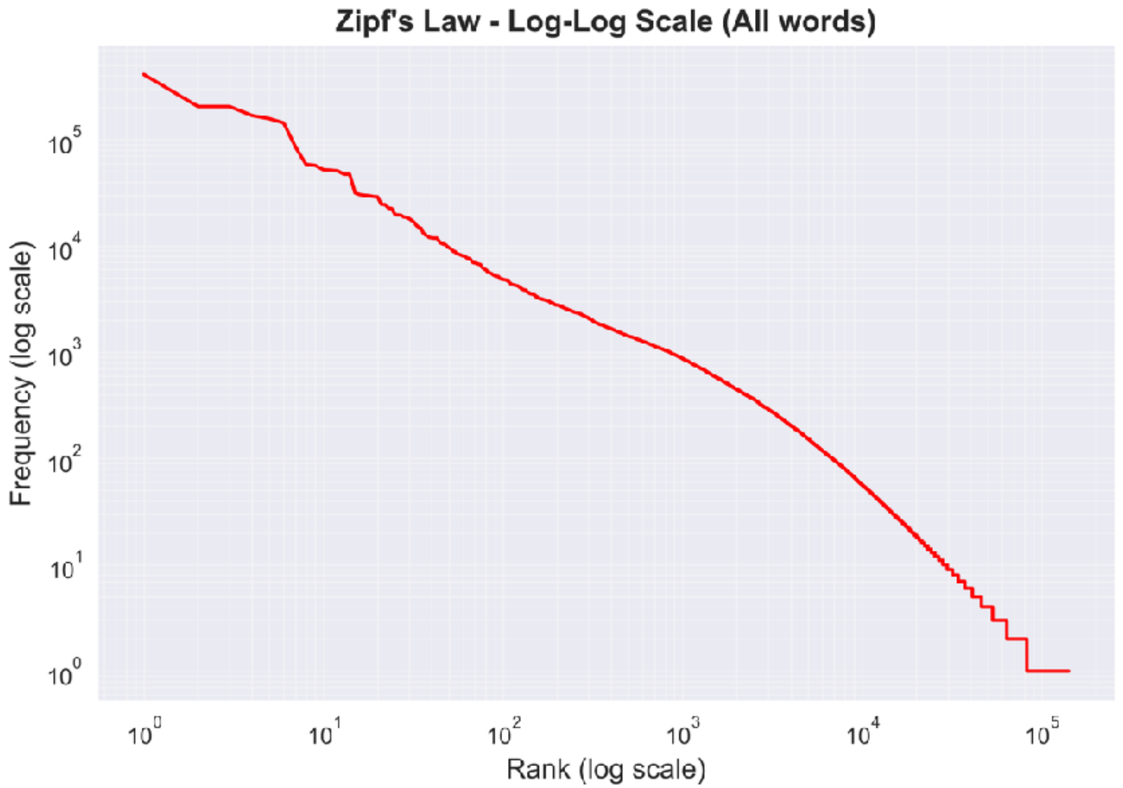
Analyse Datenset





Analyse Datenset





Datenaufbereitung

```
================================================================================
DATA SPLITS VERIFICATION REPORT
================================================================================

SPLIT SIZES
--------------------------------------------------------------------------------
Train:        56,111 samples ( 70.00%)
Validation:   12,024 samples ( 15.00%)
Test:         12,024 samples ( 15.00%)
Total:        80,159 samples (100.00%)

STRATIFICATION
--------------------------------------------------------------------------------
Number of classes: 7
Maximum distribution difference: 0.004%

DATA LEAKAGE CHECK
--------------------------------------------------------------------------------
Train ∩ Validation:       0 overlapping samples
Train ∩ Test:             0 overlapping samples
Validation ∩ Test:        0 overlapping samples

TEXT LENGTH ANALYSIS
--------------------------------------------------------------------------------
Train mean:        117.5 words
Validation mean:   115.6 words
Test mean:         117.8 words

================================================================================
VERIFICATION STATUS: ALL CHECKS PASSED ✅
================================================================================
```

Trainingsmethoden

N-grams, Embeddings, LSTM, Transformer

N-grams

Funktion:

- Erfasst aufeinanderfolgende Wortfolgen (Uni- / Bi- / Trigrams)
- TF-IDF Gewichtung
  - Term Frequency (in 1 doc)
  - Inverse Document Frequency (overall)

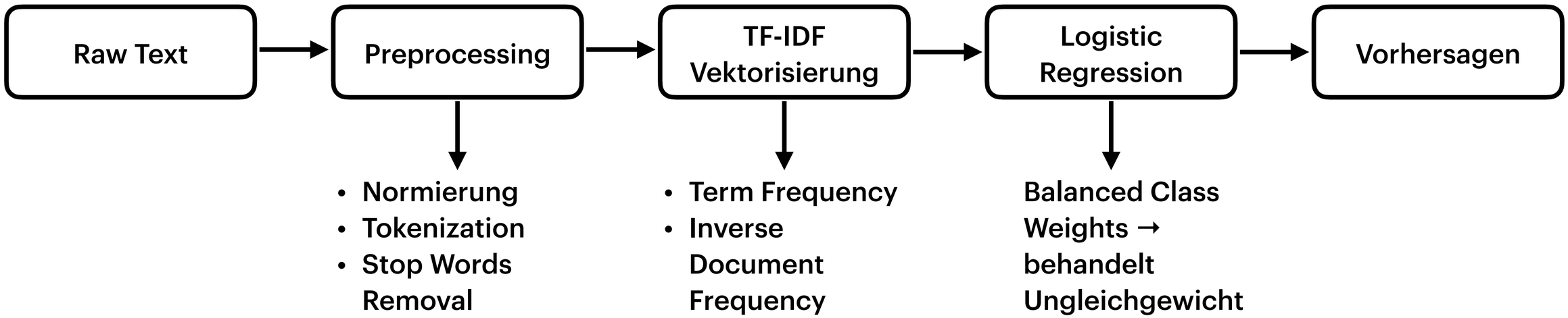
Vorteile:

- Schnelles & aufgabenspezifisches Training
- TF-IDF betont distinktives Vokabular
- Erfasst lokale Wortfolgen

Nachteile:

- Verliert langen Kontext
  → nur lokale Sequenzen

N-grams



$$\text{TF}(t, d) = \text{count}(t \text{ in } d) \qquad \text{IDF}(t) = \log\left(\frac{N}{\text{df}_{(t)}}\right) \qquad \text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

N-grams

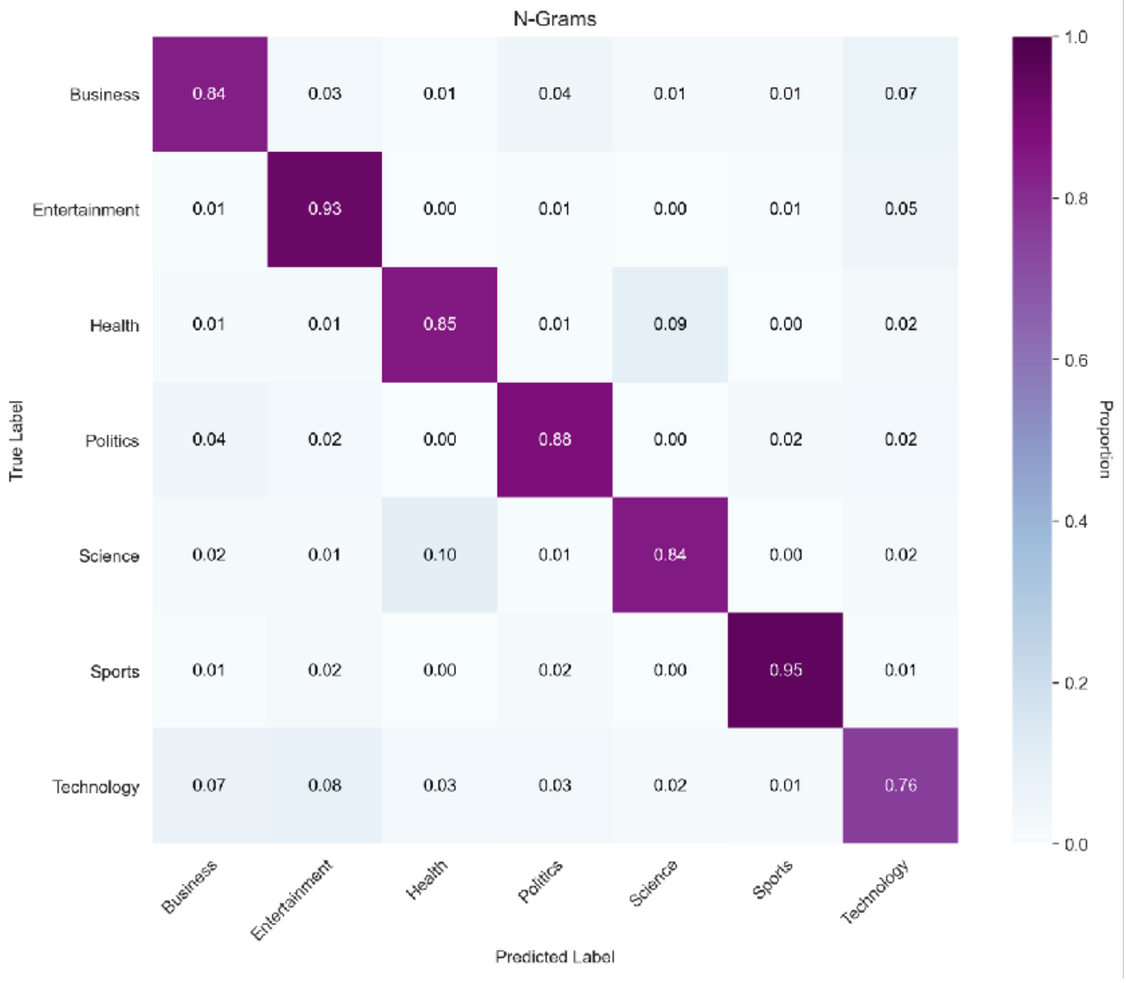
Modellparameter:

- N-gram range: (1, 3) - unigrams, bigrams, trigrams
- Max features: 10,000
- Min document frequency: 2
- Max document frequency: 0.95
- Classifier: Logistic Regression with balanced class weights

Trainingsresultat:

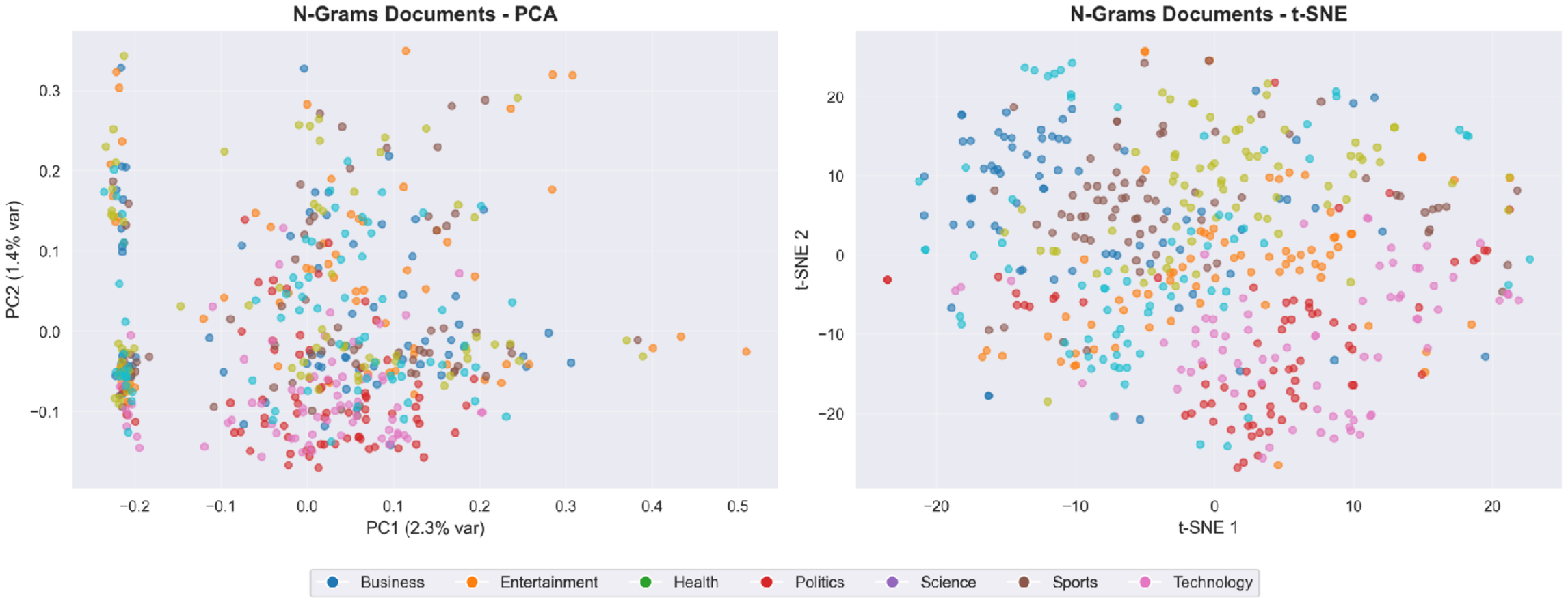
- 42.56 Sekunden
- Accuracy: 0.8641
- Macro F1: 0.8636

N-grams



- Business schwierigste Kategorie (Überlappung mit Technology & Science)
- Sport & Entertainment sehr gut unterscheidbar
- Starke Baseline: 86.4% in 42 Sekunden

N-grams


N-Grams Documents - PCA
PC2 (1.4% var)
PC1 (2.3% var)
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
N-Grams Documents - t-SNE
t-SNE 2
t-SNE 1
20
10
0
−10
−20
−20
−10
0
10
20
Business
Entertainment
Health
Politics
Science
Sports
Technology

Embeddings

Funktion:

- Lernt dichte Vektorrepräsentationen
- Jedes Wort → kontinuierlicher Vektor
- Voraussage durch Kontext
- Dokumentrepräsentation:
  Durchschnitt aller Wortvektoren

König - Mann + Frau ≈ Königin
→ semantische Arithmetik

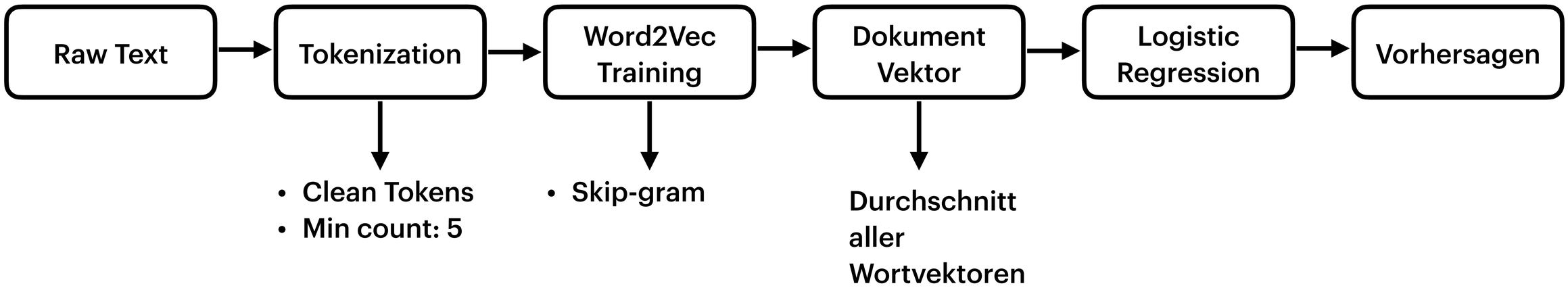
Vorteile:

- Schnelles Training
- Erfasst semantische Ähnlichkeiten
- Kompakte Repräsentation

Nachteile:

- Verliert Wortfolge-Information
- Nicht aufgabenspezifisch

Embeddings



Dokument-Vektor $\vec{d} = \frac{1}{n} \sum_{i=1}^{n} \vec{w_i}$ ← Problem: verliert Struktur

Embeddings

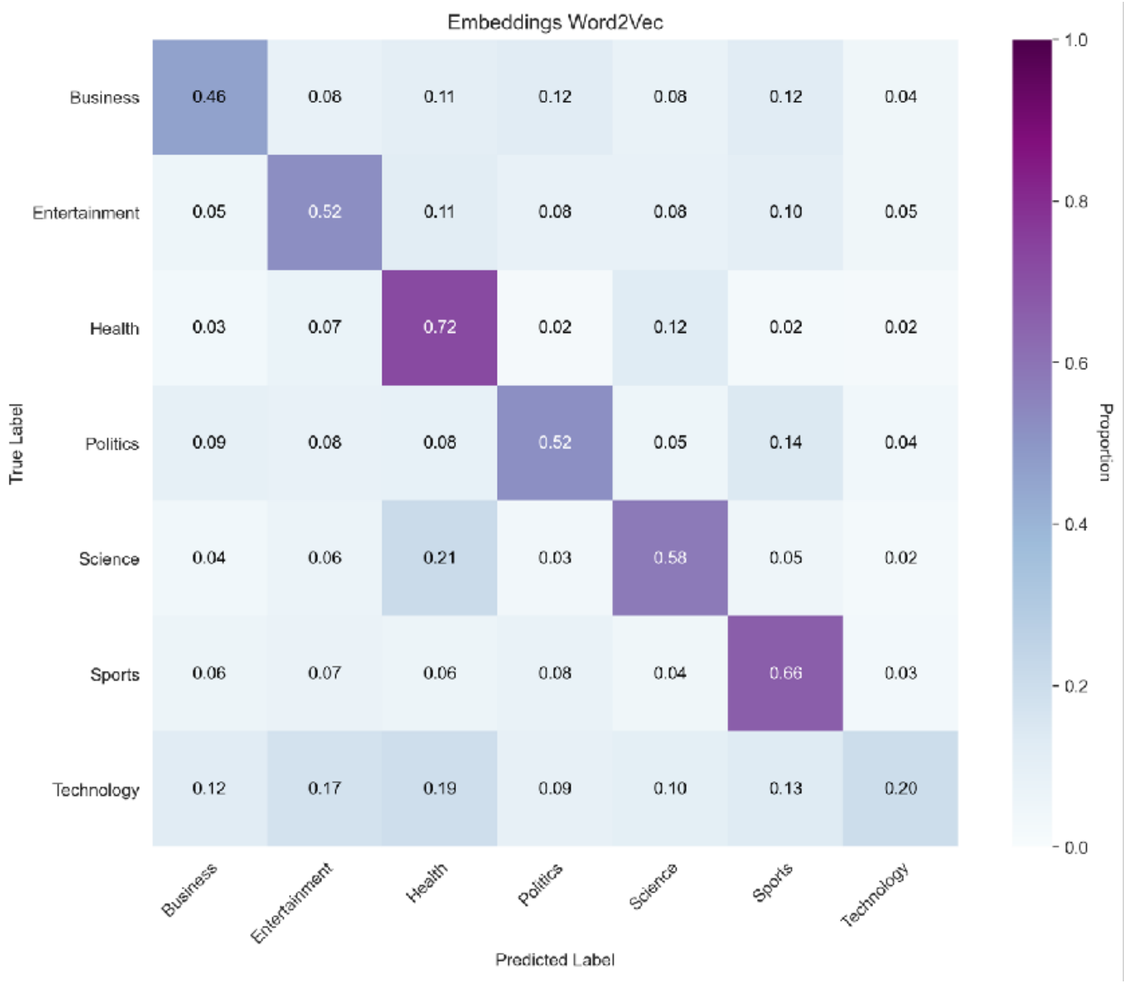
Modellparameter:

- Embedding dimensions: 200
- Vocabulary size: 45,693 words
- Window size: 5
- Min word count: 5
- Training epochs: 30
- Algorithm: Skip-gram
- Classifier: Logistic Regression

Trainingsresultat:

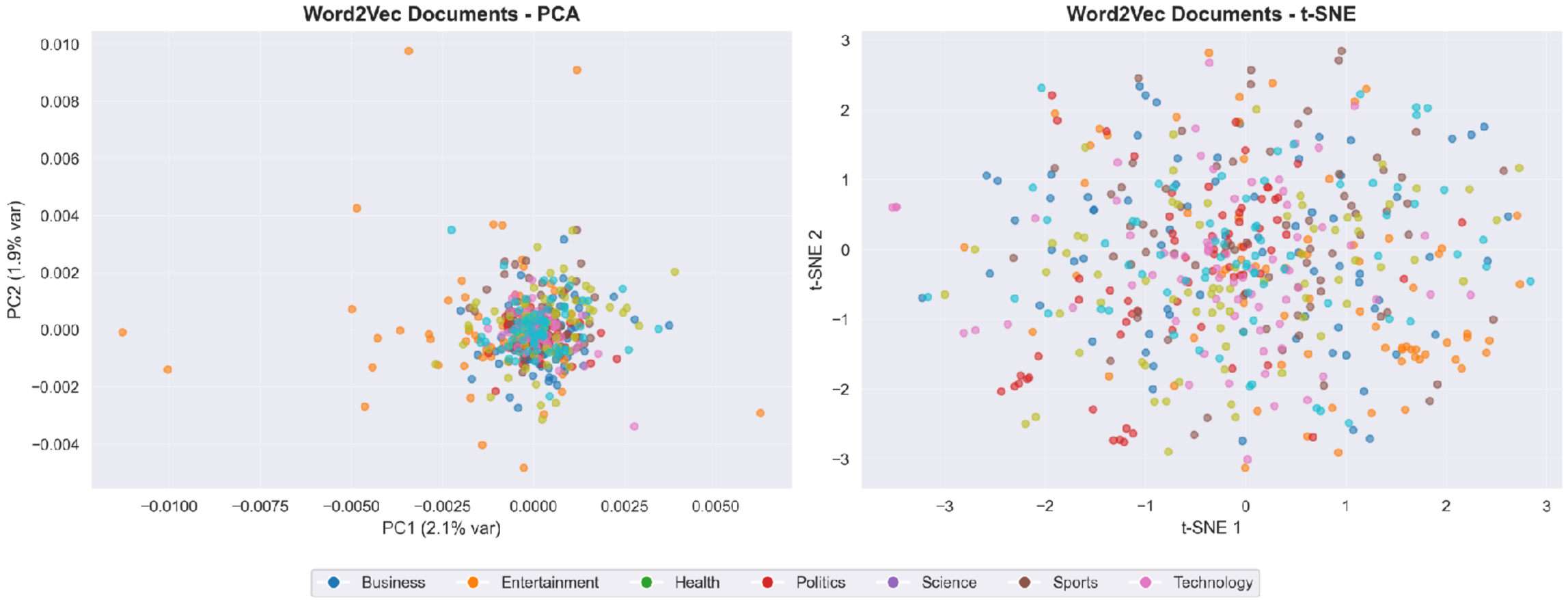
- 7.48 Sekunden
- Accuracy: 0.4885
- Macro F1: 0.4831

Embeddings



- Schwaches Resultat → evt. Programmierfehler?
- Für Klassifikationsaufgabe offenbar nicht geeignet

Embeddings


Word2Vec Documents - PCA
PC2 (1.9% var)
PC1 (2.1% var)
0.010
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
−0.0100
−0.0075
−0.0050
−0.0025
0.0000
0.0025
0.0050
Word2Vec Documents - t-SNE
t-SNE 2
t-SNE 1
3
2
1
0
−1
−2
−3
Business
Entertainment
Health
Politics
Science
Sports
Technology

Long Short-Term Memory (LSTM)

Funktion:

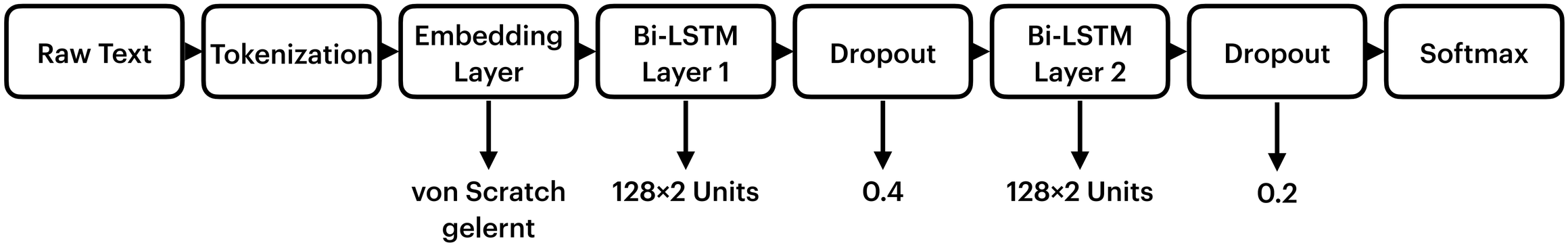
- Recurrent-Neural-Net sequentieller Muster
- Verarbeitet Text Wort für Wort bidirektional (vor- & rückwärts)
- Hidden States als Gedächtnis

Ich→hab→hunger | hunger→hab→Ich

Vorteile:

- Erfasst Wortfolge & Kontext
- Lernt aufgabenspezifische Embeddings
- Kann lange Abhängigkeiten verarbeiten
- Deterministisch und nachvollziehbar

LSTM



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Gates: $f_t, i_t, o_t = \sigma(W[h_{t-1}, x_t])$

LSTM

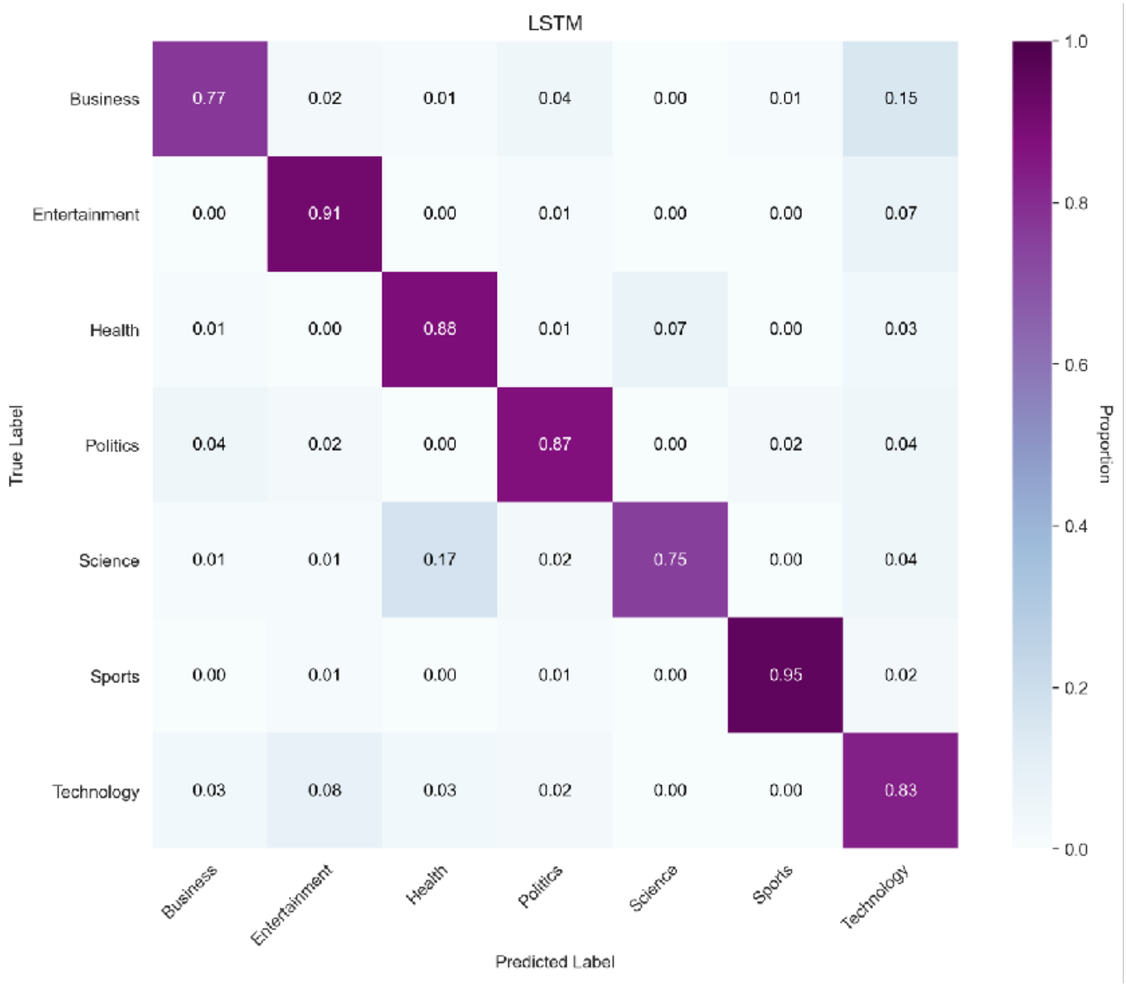
Modellparameter:

- Architecture: Bidirectional LSTM
- Number of layers: 2
- LSTM units per layer: 128 (256 Total)
- Embedding dimensions: 200 (learned from scratch)
- Vocabulary size: 20,000 words
- Max sequence length: 250 tokens
- Dropout = 0.4 & Recurrent dropout = 0.2

Trainingsresultat:

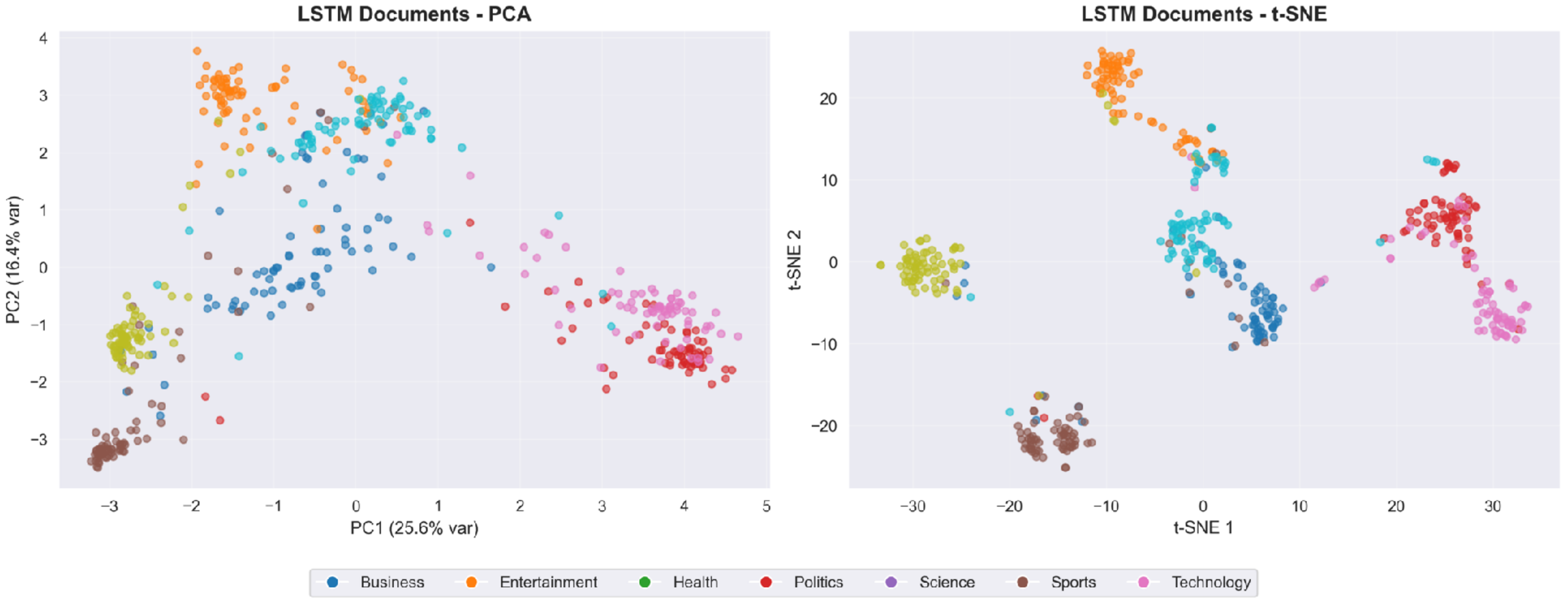
- 6.33 Stunden
- Accuracy: 0.8570
- Macro F1: 0.8561

LSTM



- Business als unerwartete Schwierigkeit
- Vergleichbares Resultat mit N-grams nach **535 Mal längerem Training** für diese Aufgabe

LSTM


LSTM Documents - PCA
PC2 (16.4% var)
PC1 (25.6% var)
LSTM Documents - t-SNE
t-SNE 2
t-SNE 1
Business
Entertainment
Health
Politics
Science
Sports
Technology

Transformer

Funktion:

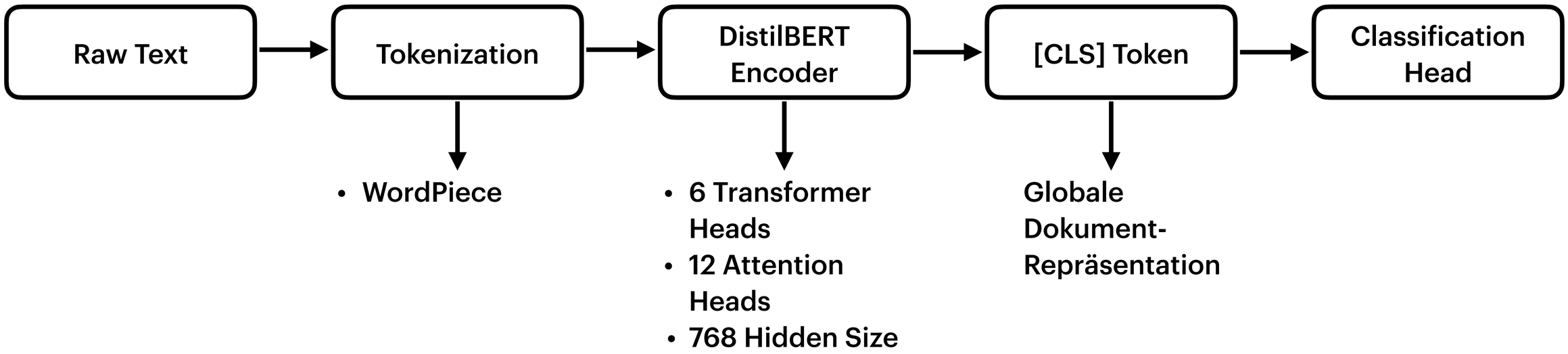
- Self-Attention-Mechanismus
- Parallele Verarbeitung aller Wörter
- Jedes Wort “beachtet” alle anderen
  → Kontextverständnis

Der Löwe schläft unter ~~dem Baum~~,
weil er müde ist.

DistilBERT:

- Vortrainiert auf grossen Textkorpora
- Kontextuelle Embeddings
  → “Bank” (Fluss) vs. “Bank” (Konto)
- Bidirektionaler Kontext
- Destilliert → 40% kleiner & 60% schneller bei 97% der BERT-Leistung

Transformer



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad Q = \text{Query}, \quad K = \text{Key}, \quad V = \text{Value}$$

Transformer

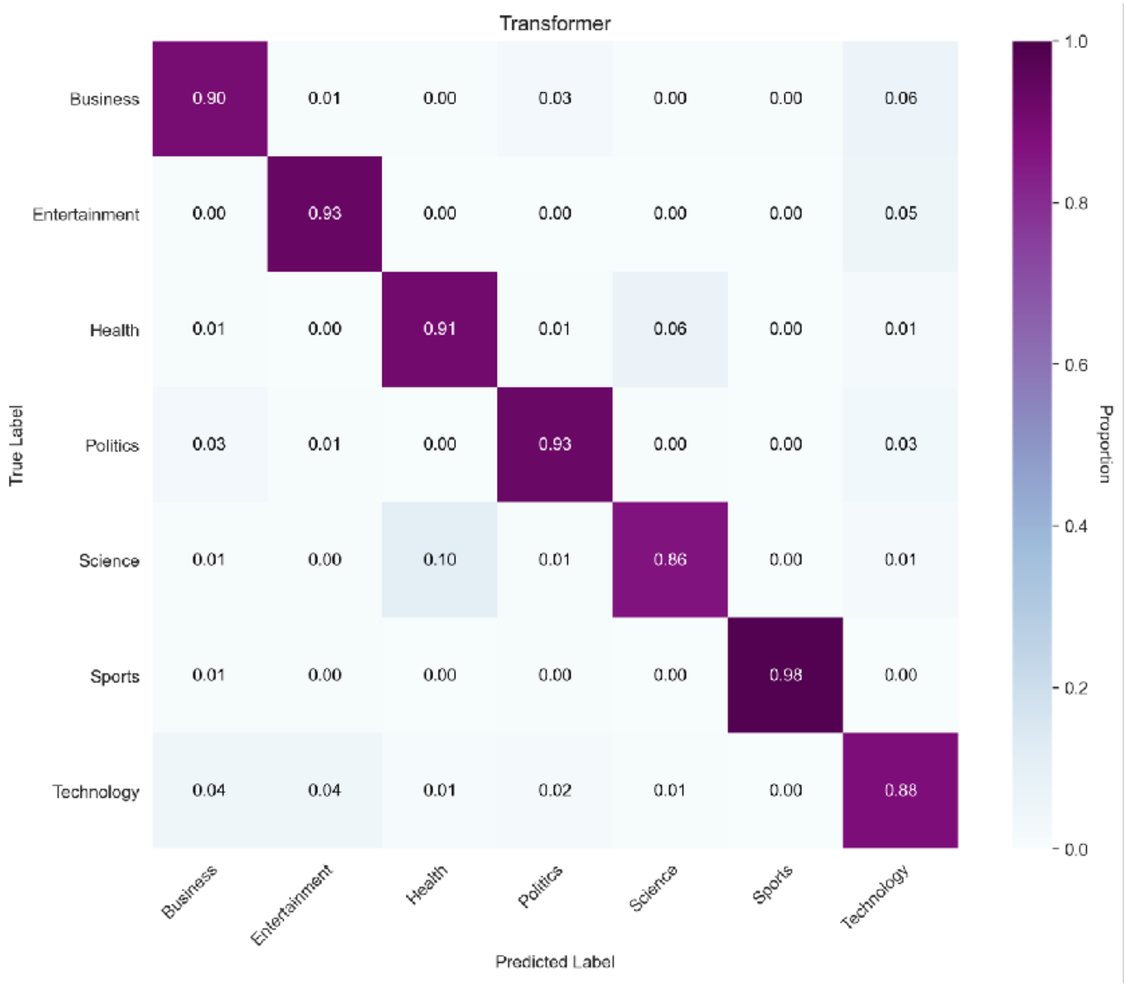
Modellparameter:

- distilbert-base-uncased
- ~66 mio. Parameter (pre-trained)
- 6 Transformer layers
- 12 attention heads
- 768 hidden size
- 512 token max sequence length
- 30’522 WordPiece tokens

Trainingsresultat:

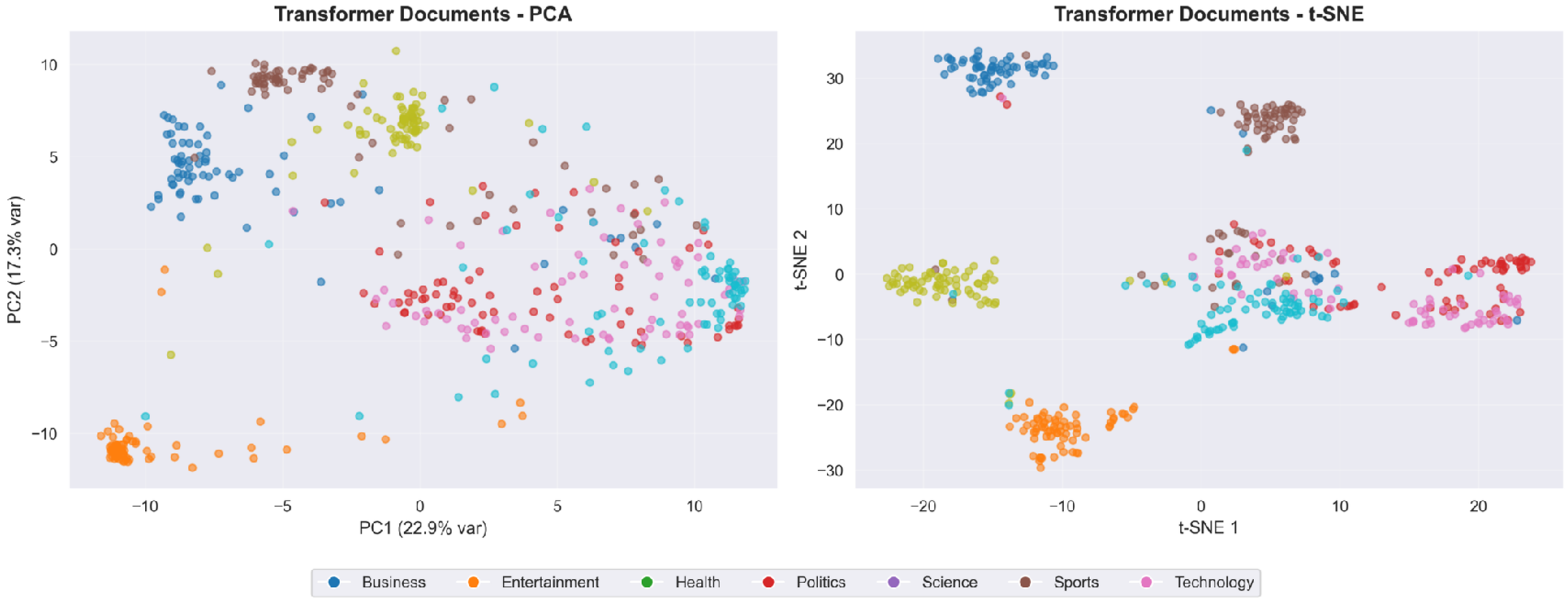
- 4 Epochen
- 8.92 Stunden
- Accuracy: 0.9153
- Macro F1: 0.9137

Transformer



- 98% Genauigkeit für Sport
- Health ↔ Sciences besser als bei anderen Methoden

Transformer



Training Comparison

```
================================================================================
MODEL COMPARISON
================================================================================

      Model    Accuracy    Macro F1    Training Time                  Library
    N-Grams      0.8641      0.8636          42.56s             scikit-learn
 Embeddings      0.4885      0.4831          🚀7.48s         gensim + sklearn
       LSTM      0.8570      0.8561           6.33h         TensorFlow/Keras
Transformer     🏆0.9153      0.9137           8.92h     PyTorch/Transformers

================================================================================
```

Lessons Learned

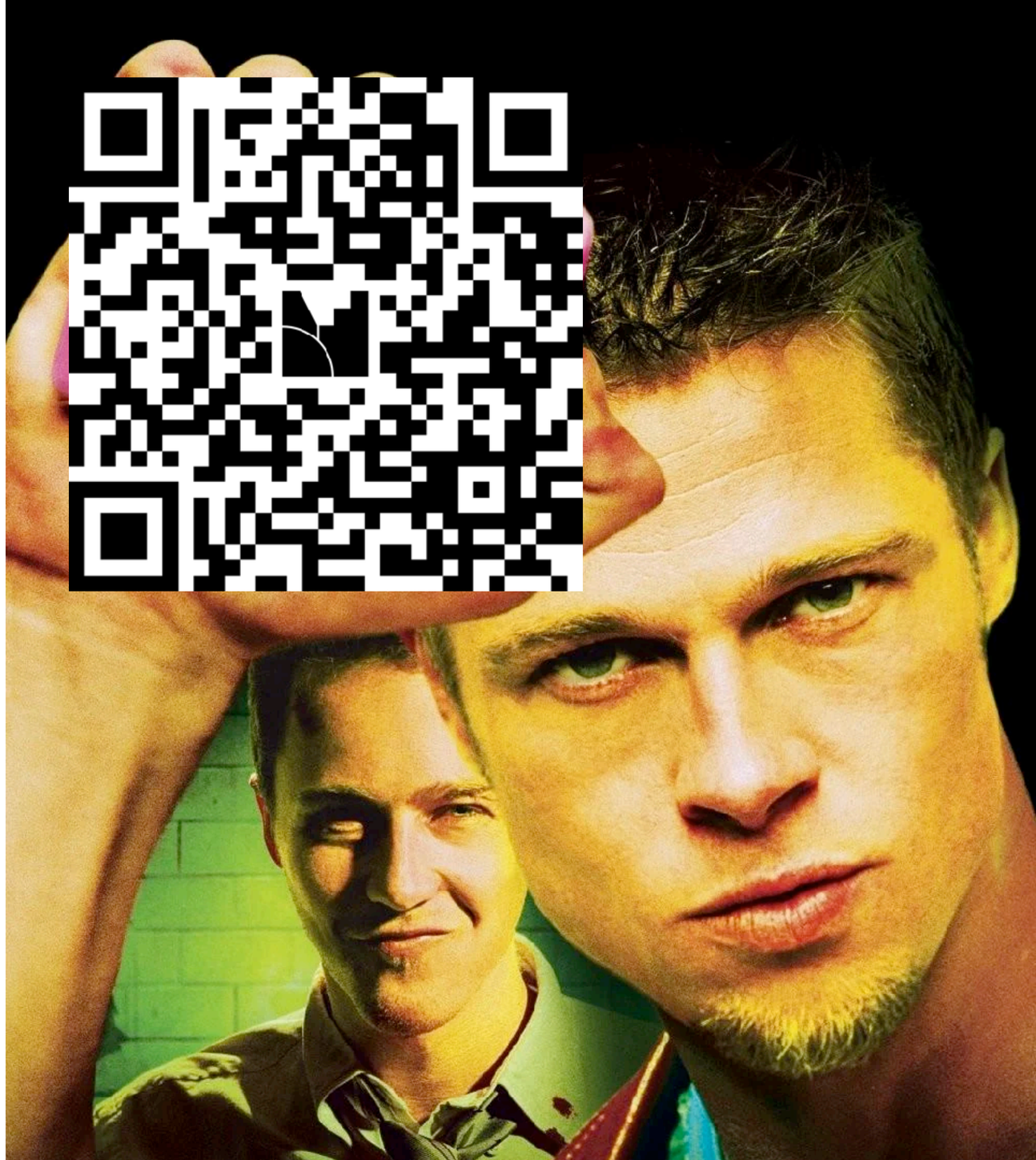
- **Embeddings Misserfolg zeigt uns:** Nicht alle "fortgeschrittenen" Methoden funktionieren universell
  - Wortvektoren erfassen Semantik, aber simples Averaging verliert Kontext
  - Dokumentstruktur ist wichtiger als Wortähnlichkeit für Klassifikation
- **Business Kategorie** war am schwersten über alle Modelle (Überschneidung mit Technology/Science)
- Vorteil von Milliarden Trainingsbeispielen bei Pre-trained Modellen klar ersichtlich

Modell Fight Club

Rules:

1. You don’t talk about fight club!
2. You don’t talk about fight club!
3. You scan the QR code and select your champion before the fight begins.

https://github.com/Colinho22/text-analytics-project/blob/main/demo/DEMO_README.md

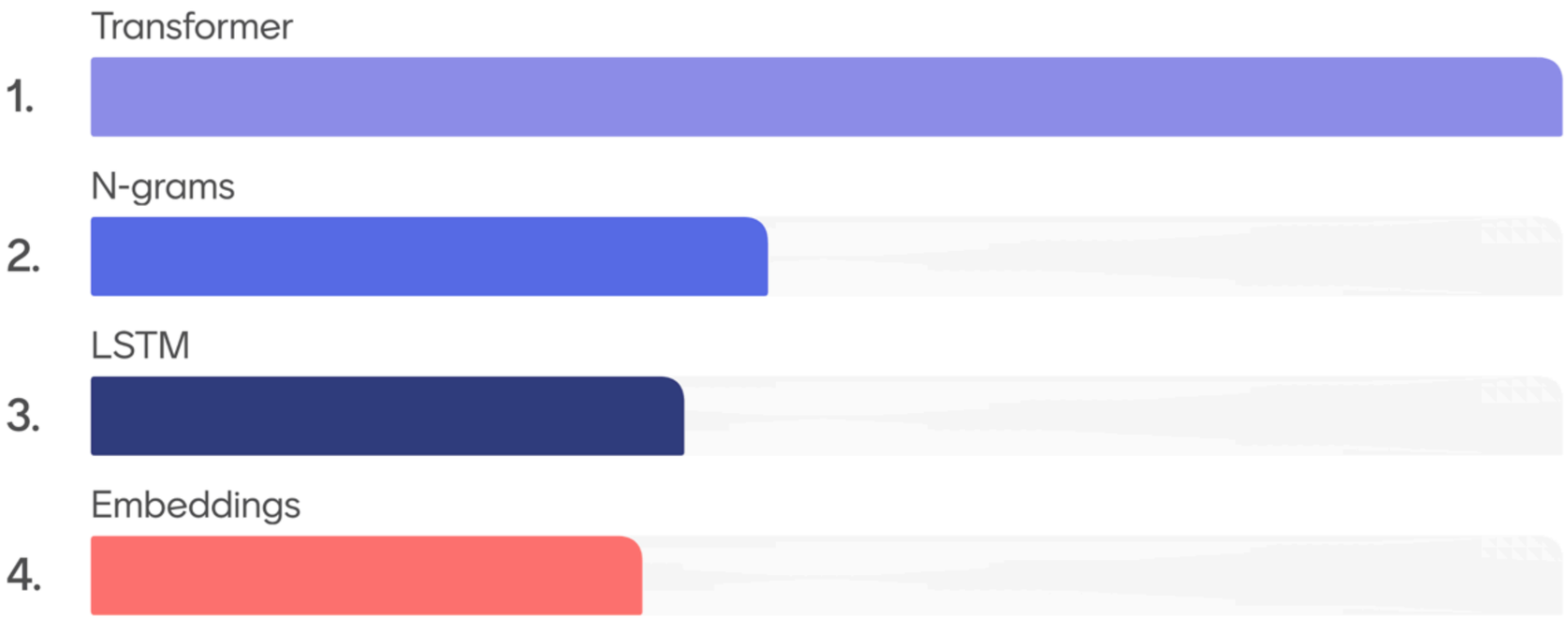


Model Fight Club - Voting

Join at menti.com | use code 8286 1677

Mentimeter

Modell Fight Club



End presentation

9 / 10

Model Fight Club - Results

Model Comparison

Compare 4 different text classification approaches

Test Models | View Results

Results Analytics

Summary (14 tests)

N-grams	Embeddings	LSTM	Transformer
51 pts	39 pts	19 pts	31 pts
↑ Avg rank: 1.36	↑ Avg rank: 2.21	↑ Avg rank: 3.64	↑ Avg rank: 2.79
Wins: 9 (64.3%)	Wins: 5 (35.7%)	Wins: 0 (0.0%)	Wins: 0 (0.0%)
Accuracy: 92.9%	Accuracy: 57.1%	Accuracy: 71.4%	Accuracy: 71.4%
Top-2: 100.0%	Top-2: 64.3%	Top-2: 0.0%	Top-2: 35.7%
Avg time: 8ms	Avg time: 8ms	Avg time: 9877ms	Avg time: 123ms