Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Magnus Bender
bender@ifis.uni-luebeck.de
Wintersemester 2023/24

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) *NumPy, SciPy*
6. Datenverarbeitung und -visualisierung
   a) *Pandas, matplotlib, NLTK*
7. Machine Learning (scikit-learn)
   a) *Grundlegende Ansätze (Datensätze, Auswertung)*
   b) Einfache Verfahren (Clustering, …)
8. DeepLearning
   a) TensorFlow, PyTorch, HuggingFace Transformers

Themen

I. Empfehlungen

II. Sprachverarbeitung

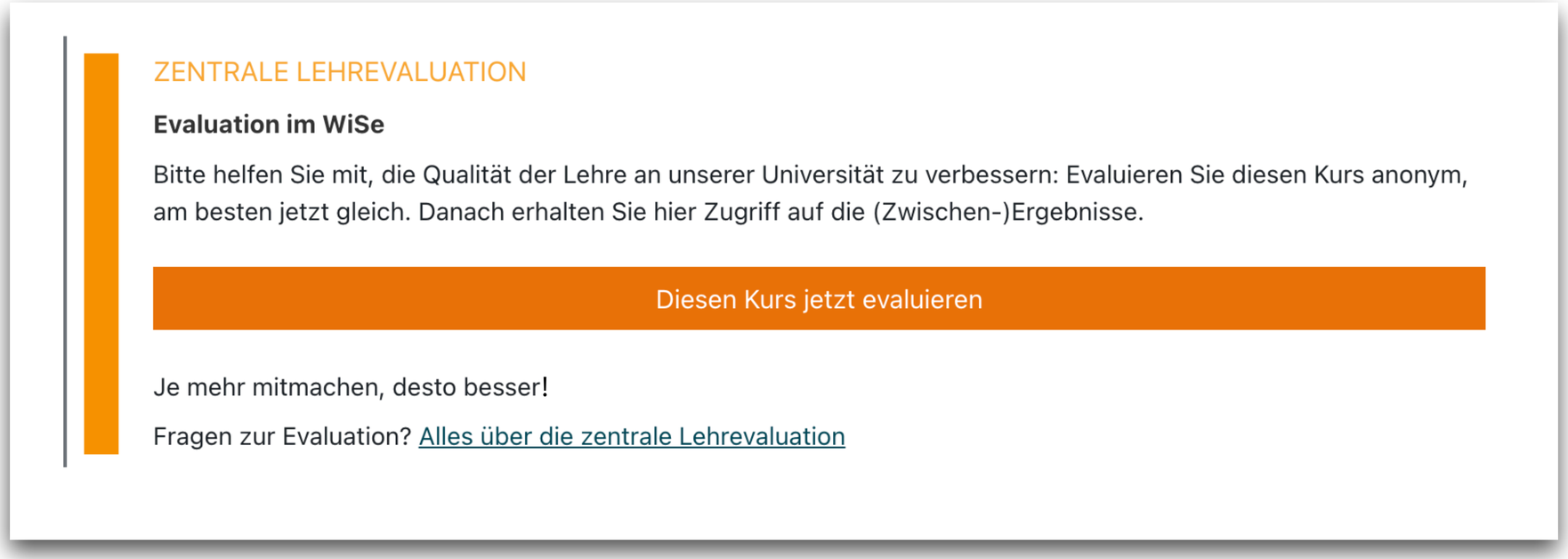
    1. *TF.IDF*

    2. Clustering von Dokumenten

    3. Empfehlung von Dokumenten

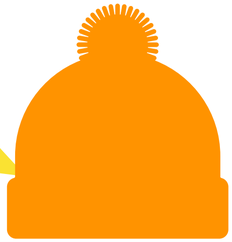
III. *Word2Vec*

Evaluation



Die Evaluation ist gestartet!

Anmeldung zur Prüfung im QiS nicht vergessen!



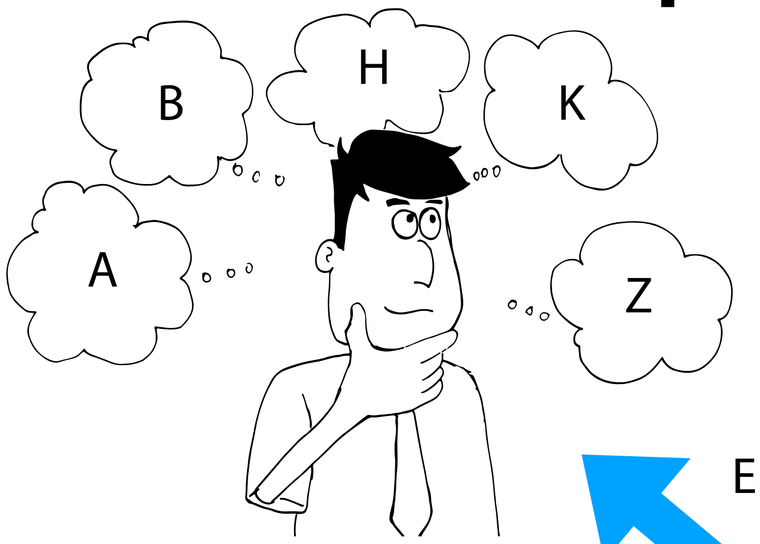
Danksagung

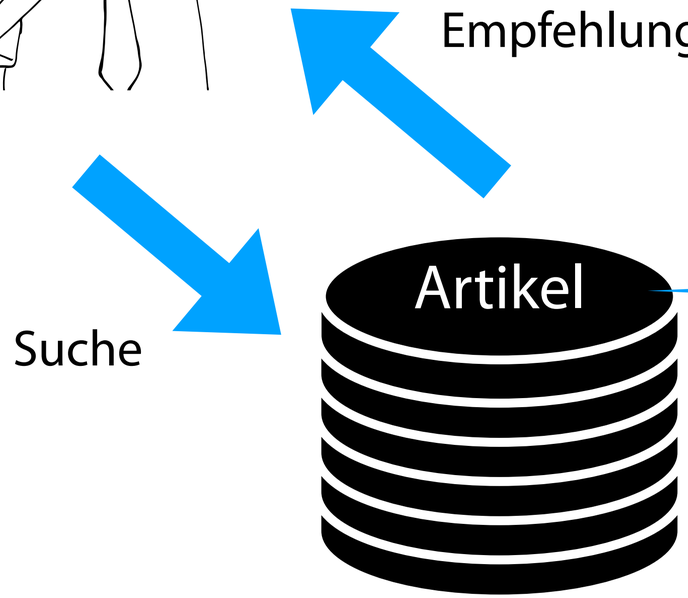
- Nachfolgende Folien sind teilweise übernommen aus folgenden Vorlesungen und Vorträgen
  - Prof. Ralf Möller: „Non-Standard Datenbanken und Data-Mining“
  - Dr. Marcel Gehrke, Prof. Ralf Möller: „Einführung in Web und Data Science“

I.
Empfehlungen

Kosinusähnlichkeit

Empfehlungsgenerierung





- Was sind gute Empfehlungen?
  - Steigerung Kundenzufriedenheit
  - Steigerung Umsatz des Anbieters

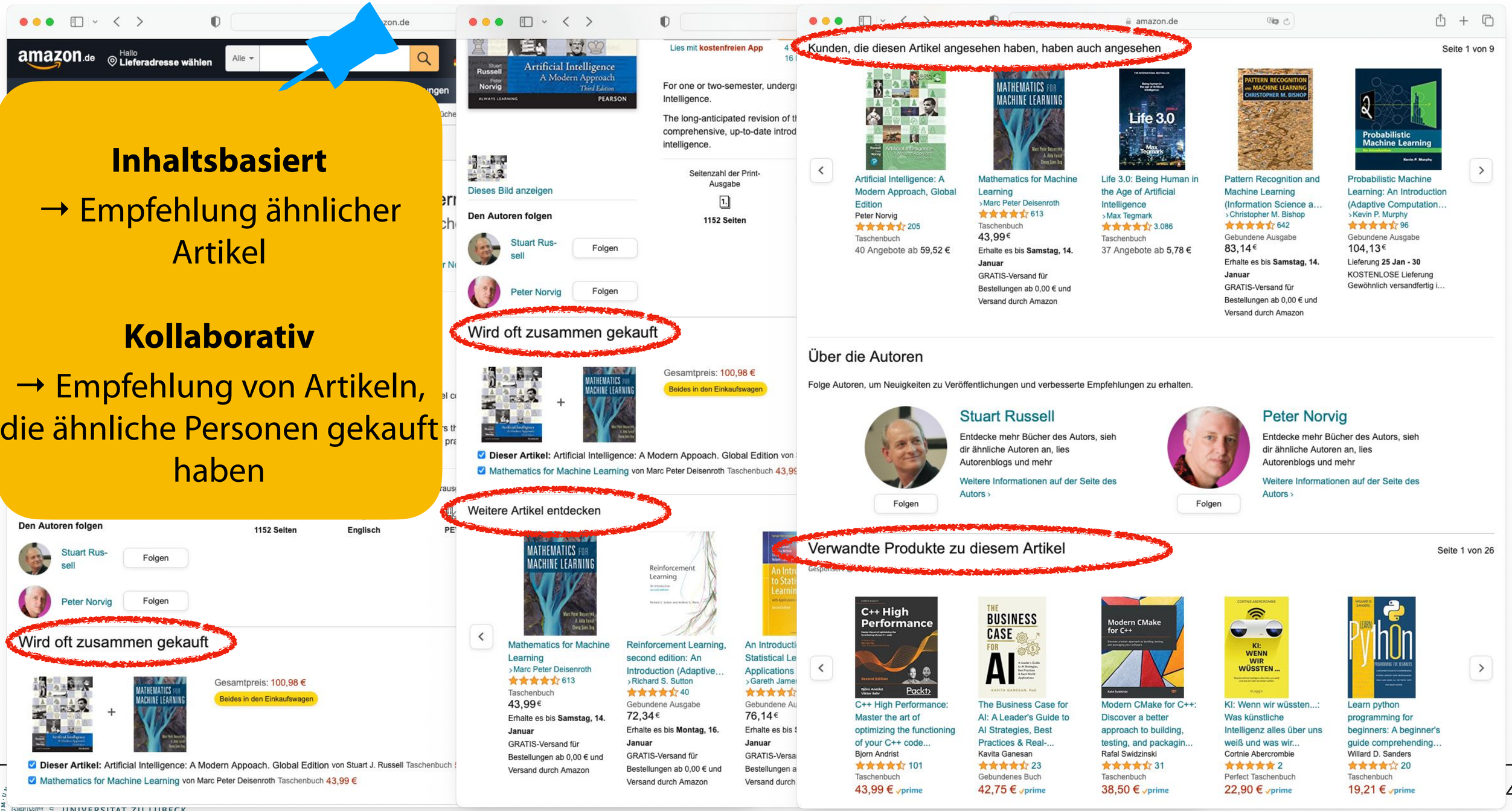
- Vorhersagen wie stark das Interesse eines „Kunden“ an einem Objekt ist
- „Kunden“ genau die Objekte aus der Menge aller vorhandenen Objekte empfehlen, für die das meiste Interesse besteht

Inhaltsbasiert

→ Empfehlung ähnlicher Artikel

Kollaborativ

→ Empfehlung von Artikeln, die ähnliche Personen gekauft haben



Warenkörbe

- Gegeben Transaktionen in Form von Warenkörben verschiedener Personen
- Gesucht Empfehlungen von weiteren Artikeln für jede Personen

Transaktions ID	Personen ID	Artikel
1	1	Brot, Haferflocken, Kartoffeln
2	2	Äpfel, Brot, Zucker
3	3	Äpfel, Brot, Kartoffeln, Zucker
4	2	Brot, Kartoffeln, Zucker
5	4	Brot, Haferflocken, Kartoffeln, Zucker

Modellierung der Nützlichkeit

Annahme: Empfehlung eines nützlichen Artikels ist das Ziel.

- Modellierung der Nützlichkeit einen Artikels für eine Person gesucht
  - C Menge von Personen (Kunden, User-IDs)
  - S Menge aller verfügbaren Artikeln (Sortiment)
  - $u : C \times S \rightarrow R$ Nützlichkeitsfunktion (Utility)
  - R Menge von Bewertungen (Sterne, Intervall $[0,1]$, o. ä.)

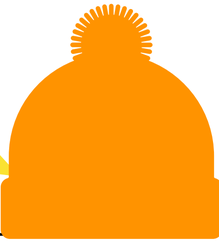
Maximierung der Nützlichkeit

Annahme: Der nützlichste Artikel ist eine gute Empfehlung.

„Bestimme für jede Person $c \in C$ denjenigen Artikel s'_c aus dem Sortiment S, der die Nützlichkeit für Person c maximiert."

$$\forall c \in C : s'_c = \arg\max_{s \in S} u(c, s)$$

Jetzt brauchen wir $u(c, s)$!



Inhaltsbasierte Nützlichkeit

s	Artikel	Gewicht	Haltbarkeit	Preis
A	Äpfel	6	4	1
B	Brot	2	1	8
H	Haferflocken	4	4	6
K	Kartoffeln	9	4	1
Z	Zucker	1	10	9

Attribute für Artikel bekannt.

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix} \quad content(B) = \begin{pmatrix} 2 \\ 1 \\ 8 \end{pmatrix}$$

$$content(H) = \begin{pmatrix} 4 \\ 4 \\ 6 \end{pmatrix} \quad content(K) = \begin{pmatrix} 9 \\ 4 \\ 1 \end{pmatrix} \quad content(Z) = \begin{pmatrix} 1 \\ 10 \\ 9 \end{pmatrix}$$

Artikel durch Vektoren beschrieben.

Personenprofile

Personen ID	Artikel
1	Brot, Haferflocken, Kartoffeln
2	Äpfel, 2x Brot, 2x Zucker, Kartoffeln
3	Äpfel, Brot, Kartoffeln, Zucker
4	Brot, Haferflocken, Kartoffeln, Zucker

Bestimmung eines Profils (auch ein Vektor der Attribute) für jede Person.

$$profile(c) = \frac{\sum_{s \in items(c)} content(s)}{|items(c)|}$$

$$profile(c_1) = \frac{1}{3}\left(content(B) + content(H) + content(K)\right)$$

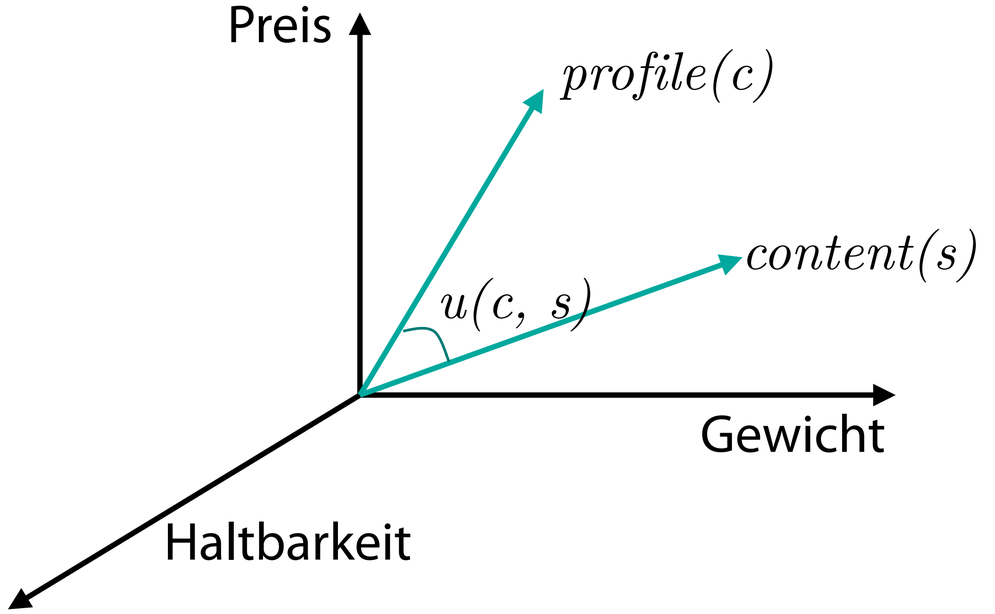
$$= \frac{1}{3}\left(\begin{pmatrix}2\\1\\8\end{pmatrix} + \begin{pmatrix}4\\4\\6\end{pmatrix} + \begin{pmatrix}9\\4\\1\end{pmatrix}\right) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

Nützlichkeitsfunktion

$$u(c, s) = \cos(profile(c), content(s))$$

Daher der Name „Kosinusähnlichkeit“

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\sum_{i=1}^{K} v_i w_i}{\sqrt{\sum_{i=1}^{K} v_i^2} \sqrt{\sum_{i=1}^{K} w_i^2}}$$



- Darstellung von Artikeln als Vektoren
- Profile als Vektoren basierend auf Warenkörben
- Winkel zwischen Artikeln und Profilen bestimmen
  → Nützlichkeitsschätzung
- Empfehlung über einen Schwellwert

Beispiel

$$
\begin{aligned}
u(c,s) &= \cos(profile(c), content(s)) \\
&= \cos\left(\begin{pmatrix}5\\3\\5\end{pmatrix}, \begin{pmatrix}5\\2\\1\end{pmatrix}\right) \\
&= \frac{5\cdot 5 + 3\cdot 2 + 5\cdot 1}{\sqrt{5^2+3^2+5^2}\sqrt{5^2+2^2+1^2}} \\
&= \frac{36}{7.68\cdot 5.48} \\
&= 0.86
\end{aligned}
$$

- Neuer Artikel *Salat (S)* verfügbar
- Bestimme die Nützlichkeit für Person 1

$$profile(c_1) = \begin{pmatrix}5\\3\\5\end{pmatrix}$$

$$content(S) = \begin{pmatrix}5\\2\\1\end{pmatrix}$$

Mittels Python I

Daten als Python-Listen und -Wörterbücher.

In der Realität aus einer Datenbank (notfalls Datei) gelesen.

```
ATTRIBUTES = [
  "gewicht",
  "haltbarkeit",
  "preis"
]
ARTICLES = {
  "A" : {
    "name" : "Aepfel",
    "gewicht" : 6,
    "haltbarkeit" : 4,
    "preis" : 1
  },
  "B" : {
    "name" : "Brot",
    "gewicht" : 2,
    "haltbarkeit" : 1,
    "preis" : 8
  },
  "H" : {
    "name" : "Haferflocken",
    "gewicht" : 4,
    "haltbarkeit" : 4,
    "preis" : 6
  },
  "K" : {
    "name" : "Kartoffeln",
    "gewicht" : 9,
    "haltbarkeit" : 4,
    "preis" : 1
  },
  "Z" : {
    "name" : "Zucker",
    "gewicht" : 1,
    "haltbarkeit" : 10,
    "preis" : 9
  }
}

TRANSACTIONS = [
  (1, ["B", "H", "K"]),
  (2, ["A", "B", "Z"]),
  (3, ["A", "B", "K", "Z"]),
  (2, ["B", "K", "Z"]),
  (4, ["B", "H", "K", "Z"])
]
```

Mittels Python II

NumPy-Array für den Content-Vektor erstellen.

```python
import numpy as np

def content(s:str) -> np.ndarray:
  return np.array([ ARTICLES[s][attribute] for attribute in ATTRIBUTES ])

print(content("A"))          [6 4 1]
```

$$content(A) = \begin{pmatrix} 6 \\ 4 \\ 1 \end{pmatrix}$$

Profil einer Person erstellen.

```python
def profile(c:int) -> np.ndarray:
  v, n = np.zeros(len(ATTRIBUTES)), 0
  for user, articles in TRANSACTIONS:
    if c == user:
      n += len(articles)
      for article in articles:
        v += content(article)

  return v/n

print(profile(1))          [5. 3. 5.]
```

$$profile(c_1) = \frac{1}{3}\left(content(B) + content(H) + content(K)\right)$$

$$= \begin{pmatrix} 5 \\ 3 \\ 5 \end{pmatrix}$$

Mittels Python III

```
from sklearn.metrics.pairwise import cosine_similarity

contents = np.vstack((content("A"), content("B"), content("H"), content("K"), content("Z")))
profiles = np.vstack((profile(1), profile(2), profile(3), profile(4)))

print(contents, profiles)
```

Attribute aller Artikel (Zeilen: Artikel, Spalten: Attribute).

```
[[ 6  4  1]
 [ 2  1  8]
 [ 4  4  6]
 [ 9  4  1]
 [ 1 10  9]]
```

```
[[5.   3.   5.  ]
 [3.5  5.   6.  ]
 [4.5  4.75 4.75]
 [4.   4.75 6.  ]]
```

Profile aller Personen (Zeilen: Person, Spalten: Attribute).

```
print(cosine_similarity(profiles, contents))
```

Nützlichkeit für jeden Artikel und jede Person (Zeilen: Personen, Spalten: Artikel).

```
[[0.84049264 0.83066386 0.97883892 0.81536609 0.77201953]
 [0.75432084 0.843962   0.99183578 0.67865803 0.93104132]
 [0.86216872 0.77051298 0.98238335 0.80270181 0.8686357 ]
 [0.77946733 0.84695728 0.99711135 0.71360283 0.90564196]]
```

Mittels Python IV

```
def new_article(profiles, article_content):
  utilities = cosine_similarity(profiles, article_content)
  print(utilities)
```

```
[[0.85568884]
 [0.71462855]
 [0.82983331]
 [0.75059816]]
```

```
  best = np.argmax(utilities)

  print("Recommend", article_content, "to User", best+1)
```

```
Recommend [[5, 2, 1]] to User 1
```

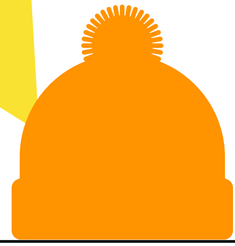
```
  return best

new_article(profiles, [[5, 2, 1]])
```

Die Eingabe und Verarbeitung der Daten (Transaktionen, Artikel) ist nicht effizient. Hier sollte eine Datenbank (oder Pandas) zum Einsatz kommen.

Weiterhin könnten die Matrizen `profiles` and `contents` auf der Festplatte gespeichert werden.

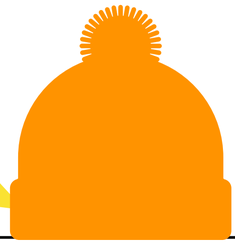
Wir speichern die Profile als NumPy-Array und können anschließend mittels `sklearn` effizient mit der Matrix arbeiten.



II.
Sprachverarbeitung

1. TF.IDF

In Vorlesung 6a haben wir bereits die Vorverarbeitung betrachtet.



Ergebnis aus 6a

- *Bag-of-Words*-Modell
  - Texte als Vektoren dargestellt

Für die *Kosinusähnlichkeit* und für *k-Means* brauchen wir Vektoren als Eingaben!

Wort	Satz 0	Satz 1	Satz 2
dabei	0	1	0
daran	0	0	1
denk	0	0	1
erstellt	0	0	1
geb	1	0	0
gefragt	0	1	0
gewahr	0	0	1
git	0	0	1
github	0	0	1
link	1	0	0
moodl	1	0	0
offent	0	1	0
repositori	1	1	2
user	0	0	1
verborg	0	1	1
werkzeugewissarbeit	0	0	1
zugriffsrecht	0	0	1

So einfach geht es leider nicht …

- Term-Häufigkeit in einem Dokument ignoriert
- Term-Seltenheit in über alle Dokumente hinweg ignoriert
- Länge der Dokumente nicht berücksichtigt

Wir betrachten nun nicht mehr nur einzelne Sätze, sondern Korpora bestehend aus Dokumenten (die wieder aus Sätzen und Worten bestehen).

Term-Frequenz (TF)

- Lange Dokumente erreichen hohe Zähler für Wörter
- Normalisierung mittels Dokumentenlänge

$$tf_{t,d} = \frac{t_d}{|d|}$$

mit

t = Term/ Wort

d = Dokument

$|d|$ = Anzahl Wörter/ Länge von d

Reicht das schon?

Term „Hallo“ vs. Term „Willenserklärung“

Inverse Dokumentenfrequenz (IDF)

- Maß der Information
- Seltenheit eines Terms im Korpus
  - „Hallo“ kommt häufig vor und *sagt wenig aus*
  - „Willenserklärung“ kommt als jur. Fachterminus selten vor und *sagt daher viel aus*

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

mit

t = Term/ Wort

N = Anzahl Dokumente im Korpus

df_t = Anzahl Dokumente mit Term t

TF.IDF

$$tf.idf_{t,d} = tf_{t,d} \cdot idf_t = \frac{t_d}{|d|} \cdot \log\left(\frac{N}{df_t}\right)$$

- Für jeden Term und jedes Dokument definiert
- Gewichtung der Relevanz jedes Wortes im Korpus
- Nutzung als Gewichtung für Wortvektoren

Mittels SKLearn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
  'Hallo Franz, ich habe eine Info.',
  'Danke, ich habe auch eine Info.',
  'Hallo, ich habe eine Willenserklaerung!',
  'Ich habe eine Info.',
]
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

print(vectorizer.get_feature_names_out())
```

```
['auch' 'danke' 'eine' 'franz' 'habe' 'hallo' 'ich' 'info' 'willenserklaerung']
```

```python
print(vectorizer.transform(corpus).toarray())
```

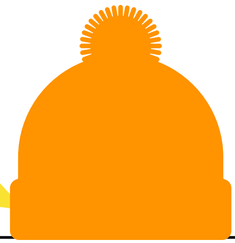
```
[[0.   0.   0.30 0.59 0.30 0.46 0.30 0.37 0.   ]
 [0.55 0.55 0.29 0.   0.29 0.   0.29 0.35 0.   ]
 [0.   0.   0.33 0.   0.33 0.50 0.33 0.   0.64 ]
 [0.   0.   0.47 0.   0.47 0.   0.47 0.57 0.   ]]
```

„Franz“ und „Willenserklärung“ sind hoch gewichtet.
„Ich“ kommt in jedem Satz Dokument vor und hat geringes Gewicht.

II.
Sprachverarbeitung

2. Clustering von Dokumenten

TD.IDF zusammen mit
k-Means in *SKLearn*



Korpus

```python
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans


X_train, y_train = fetch_20newsgroups(
  remove=["headers", "footers", "quotes"],
  categories=["comp.graphics", "rec.autos"],
  subset="train",
  return_X_y=True
)
num_labels = np.unique(y_train).shape[0]

print(y_train)
```

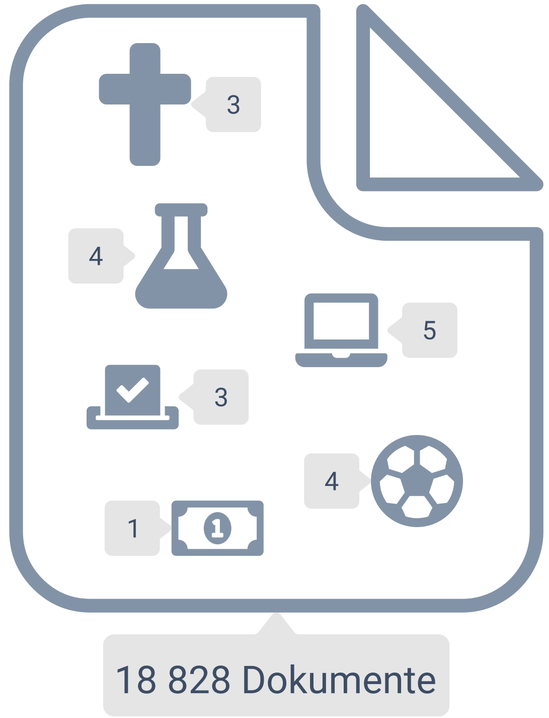
```
[0 0 1 ... 0 0 0]
```

```python
print(X_train[1:2])
```

```
['Hello netters!\n \nI have a fairly weak question to ask everybody in netland.  I\'ve looked though\nthe last FAQ for comp.graphics but I didn\'t find my answer.  Thus the post.\n \nI\'ll keep it short.\n \nQUESTION:  How do I display any raster files, gif files, iff or tiff images\nthat I have on my "root window" or background?  I have a sun ipc, openwindows\n3.0, Sun OS 4.1.3 if that helps any.\n \nI\'ve compiled POV for the sun and would like to display some of the work I have\ndone as a background/tile.  Thanks for any help or information that you\nprovide.

Have a good day. \n \nScott Fleming\nOSI']
```

20 Newsgroups



Modell trainier

1. Vorverarbeitung (Stopwords, …)
2. Korpus in TF.IDF überführen (Matrix: Anzahl Dokumente × Wörter)
3. K-Means Clustering bestimmen

Die Vorverarbeitung mittels NLTK erzielt i.A. bessere Ergebnisse.

„Modell“ für Texte `X_train` erstellen.

K-Means soll zwei Cluster bestimmen, da nur zwei Newsgruppen gewählt.

```
tfidf = TfidfVectorizer(stop_words="english")
tfidf.fit(X_train)

kmeans = KMeans(n_clusters=num_labels, n_init="auto", random_state=42)
kmeans.fit(tfidf.transform(X_train))
```

Zuerst die Texte in die TF.IDF-Matrix überführen, dann das Clustering bestimmen.

Modell evaluieren

Jetzt die Testdaten laden!

```
from sklearn.metrics import accuracy_score

X_test, y_test = fetch_20newsgroups(
  remove=["headers", "footers", "quotes"],
  categories=["comp.graphics", "rec.autos"],
  subset="test",
  return_X_y=True
)
y_prediction = kmeans.predict(tfidf.transform(X_test))

print(np.unique(y_test == y_prediction, return_counts=True))

      (array([False,  True]), array([151, 634]))

print(accuracy_score(y_test, y_prediction))

      0.8076433121019109
```

Wieder zuerst in TF.IDF-Matrix überführen und dann Vorhersage mittels k-Means.

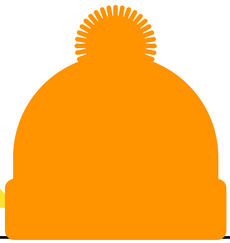
Ist das Label „1“ auch immer der Cluster mit Index „1“?

Auszählen der gleichen und verschiedenen Label zwischen den wahren und den vorhergesagten Labeln.

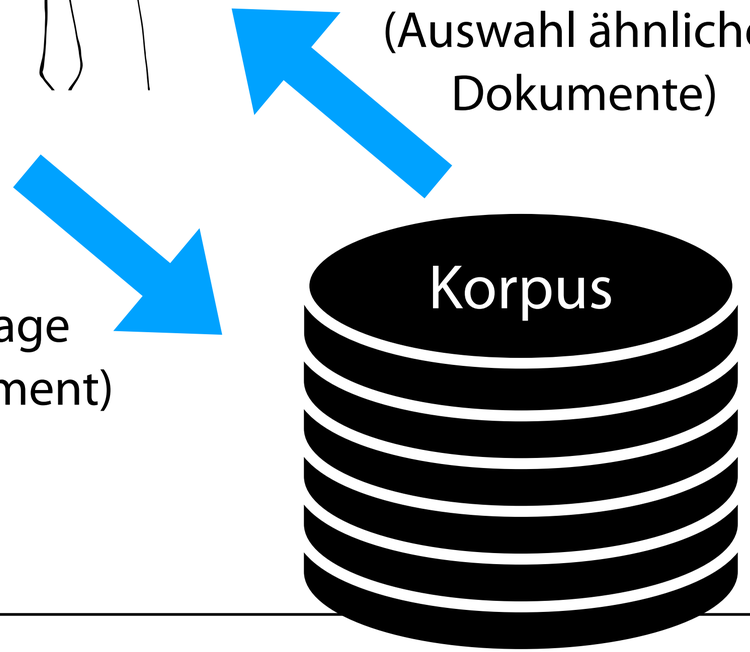
II.
Sprachverarbeitung

3. Empfehlungen von Dokumenten

Ähnliche Dokumente mittels *TD.IDF* und der *Kosinusähnlichkeit* in *SKLearn* bestimmen.



Suche nach Dokumenten



- Ähnlich zu einer Suchmaschine
  - „Document retrieval“
- Beantworten von Anfragen
  - Anfrage z.B. Dokument
  - Antwort ähnliche Dokumente aus dem Korpus

Wieder Korpus

```python
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer

X_train, y_train = fetch_20newsgroups(
  remove=["headers", "footers", "quotes"],
  categories=["comp.graphics", "rec.autos"],
  subset="train",
  return_X_y=True
)
X_test, y_test = fetch_20newsgroups(
  remove=["headers", "footers", "quotes"],
  categories=["comp.graphics", "rec.autos"],
  subset="test",
  return_X_y=True
)
num_labels = np.unique(y_train).shape[0]

tfidf = TfidfVectorizer(stop_words="english")
X_train_tfidf = tfidf.fit_transform(X_train)
```

Kennen wir schon, nur der Vollständigkeit halber hier.

Und gleich TF.IDF vorbereiten.

Ähnliche Dokumente bestimmen

```
from sklearn.metrics.pairwise import cosine_similarity
from collections import Counter

def fetch_similar_documents(doc:str, top_n:int=10):
  sim = cosine_similarity(tfidf.transform([doc]), X_train_tfidf)
  return (-sim).argsort(axis=None)[:top_n]

for i in [2, 10, 20, 30, 50, 100, 400]:
  best = fetch_similar_documents(X_test[i])
  y_counts = Counter([y_train[b] for b in best])
  print("Document {: >3} (class {}) fetched {: >2} times same class: {}".format(
    i, y_test[i], y_counts[y_test[i]], best
  ))
```

```
Document   2 (class 0) fetched  7 times same class: [102
Document  10 (class 1) fetched  9 times same class: [ 52
Document  20 (class 1) fetched  6 times same class: [ 474  403 …  67]
Document  30 (class 0) fetched 10 times same class: [ 985  680 …  33]
Document  50 (class 1) fetched 10 times same class: [ 827  301 … 703]
Document 100 (class 0) fetched 10 times same class: [1126  142 … 576]
Document 400 (class 0) fetched 10 times same class: [ 946  677 … 287]
```

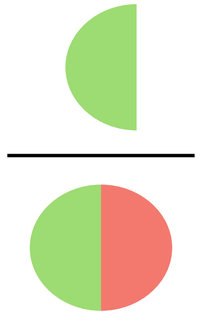
TF.IDF anwenden und dann Koninusähnlichkeit nutzen.

Ähnlichste 10 Dokumente (Indizes) zurückgeben.

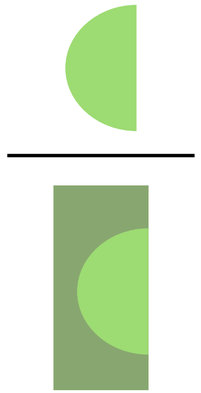
Stichprobenhafte Auswertung durch Vergleich der Klassen.

Metriken

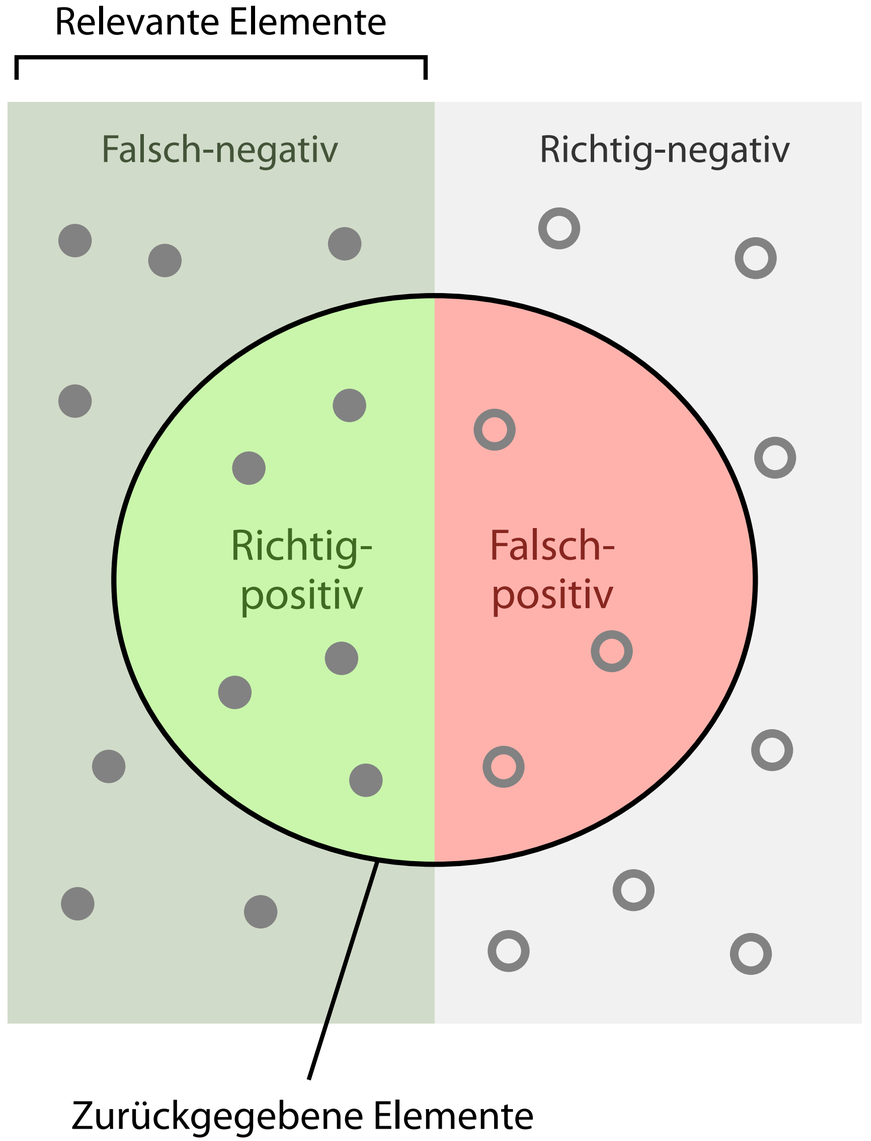
Wie viele gefundene Elemente sind relevant?

Precision = 

Wie viele relvante Elemente wurden gefunden?

Recall = 

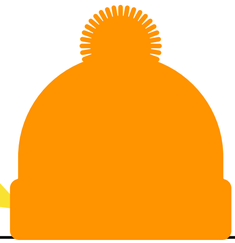
```
sklearn.metrics.precision_score()
sklearn.metrics.recall_score()
sklearn.metrics.precision_recall_fscore_support()
```



III.
Word2Vec

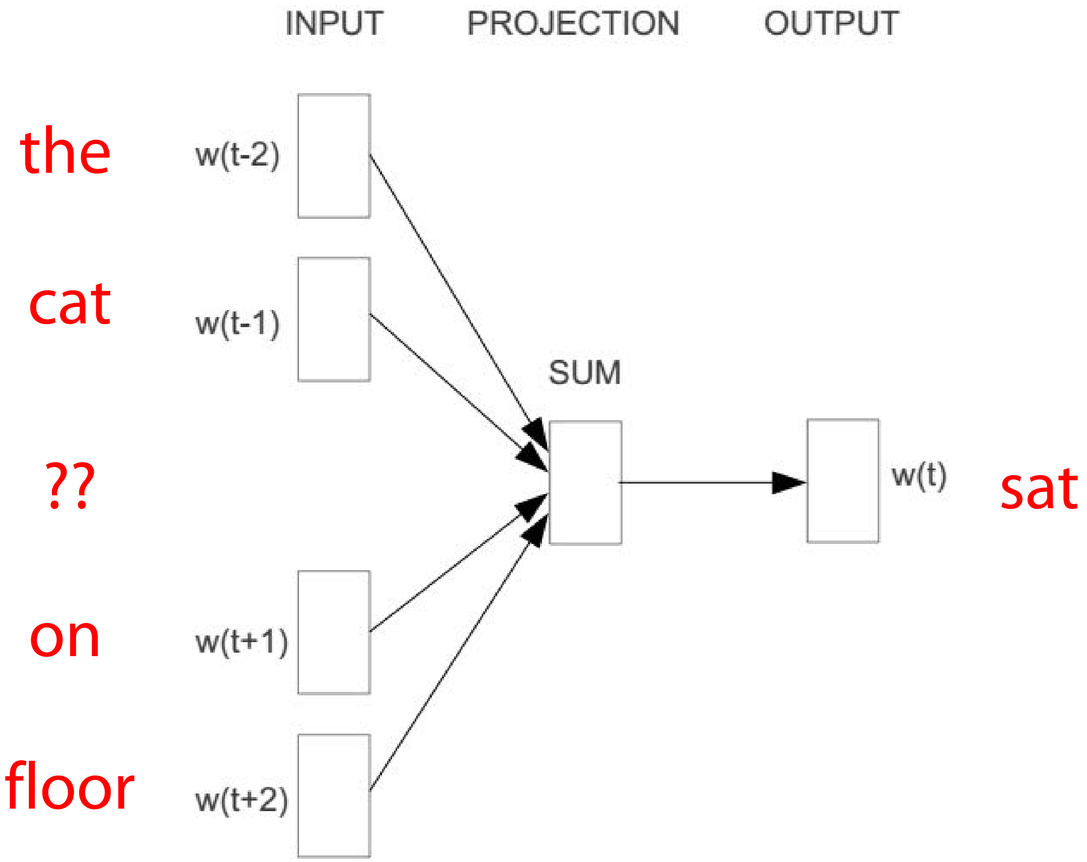
Ausblick auf weitere Techniken aus der Sprachverarbeitung

Nicht Texte sondern Wörter als Vektoren darstellen.



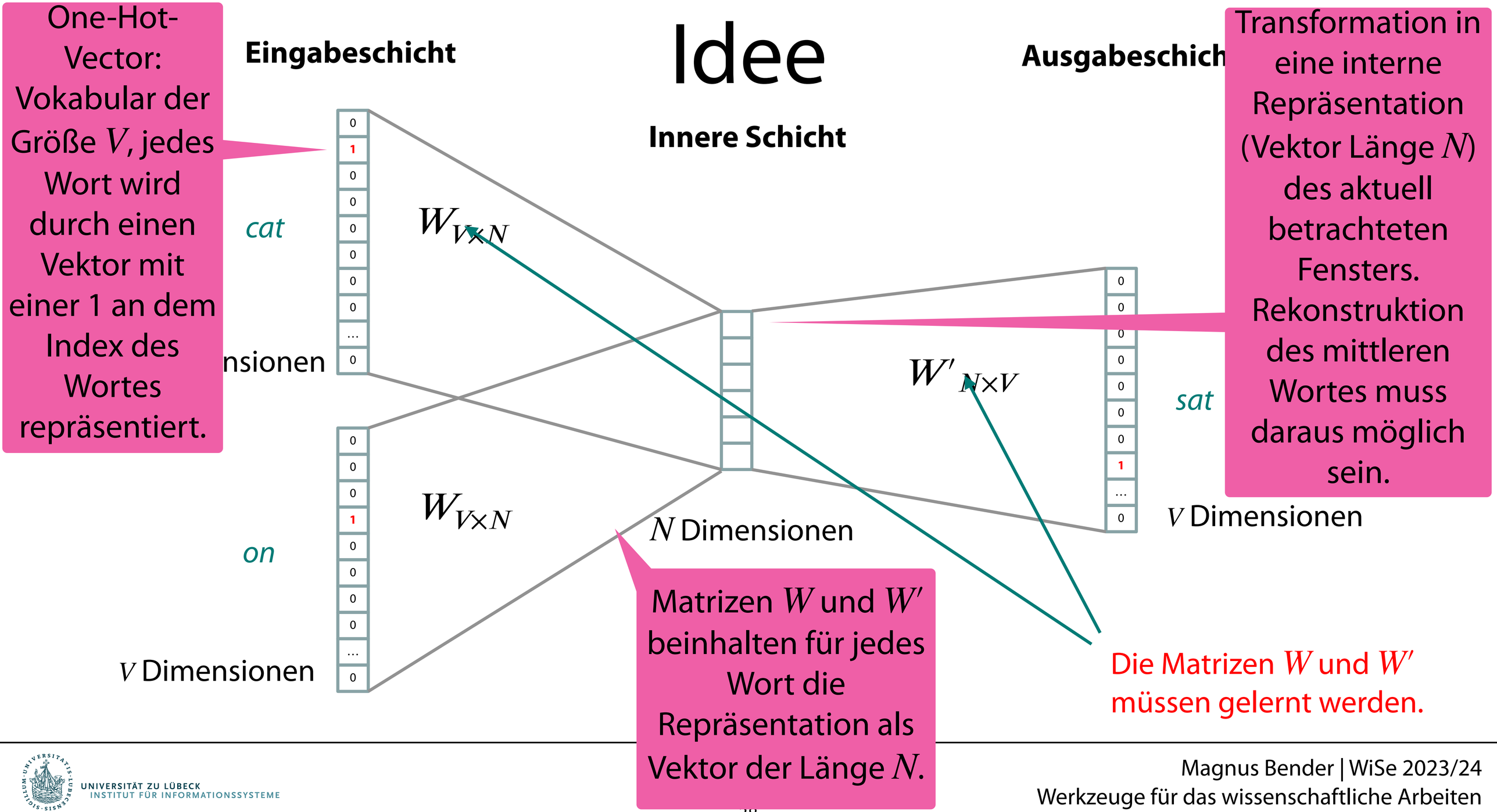
Wörter als Vektoren

- Bedeutung von Wörtern in Vektoren darstellen
- *Continuous Bag of Words (CBOW)*
  - Nutze ein Fenster von Wörtern um das mittlere Wort vorherzusagen



Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean „Efficient Estimation of Word Representations in Vector Space“ 2013, https://arxiv.org/abs/1301.3781

Idee



One-Hot-Vector: Vokabular der Größe V, jedes Wort wird durch einen Vektor mit einer 1 an dem Index des Wortes repräsentiert.

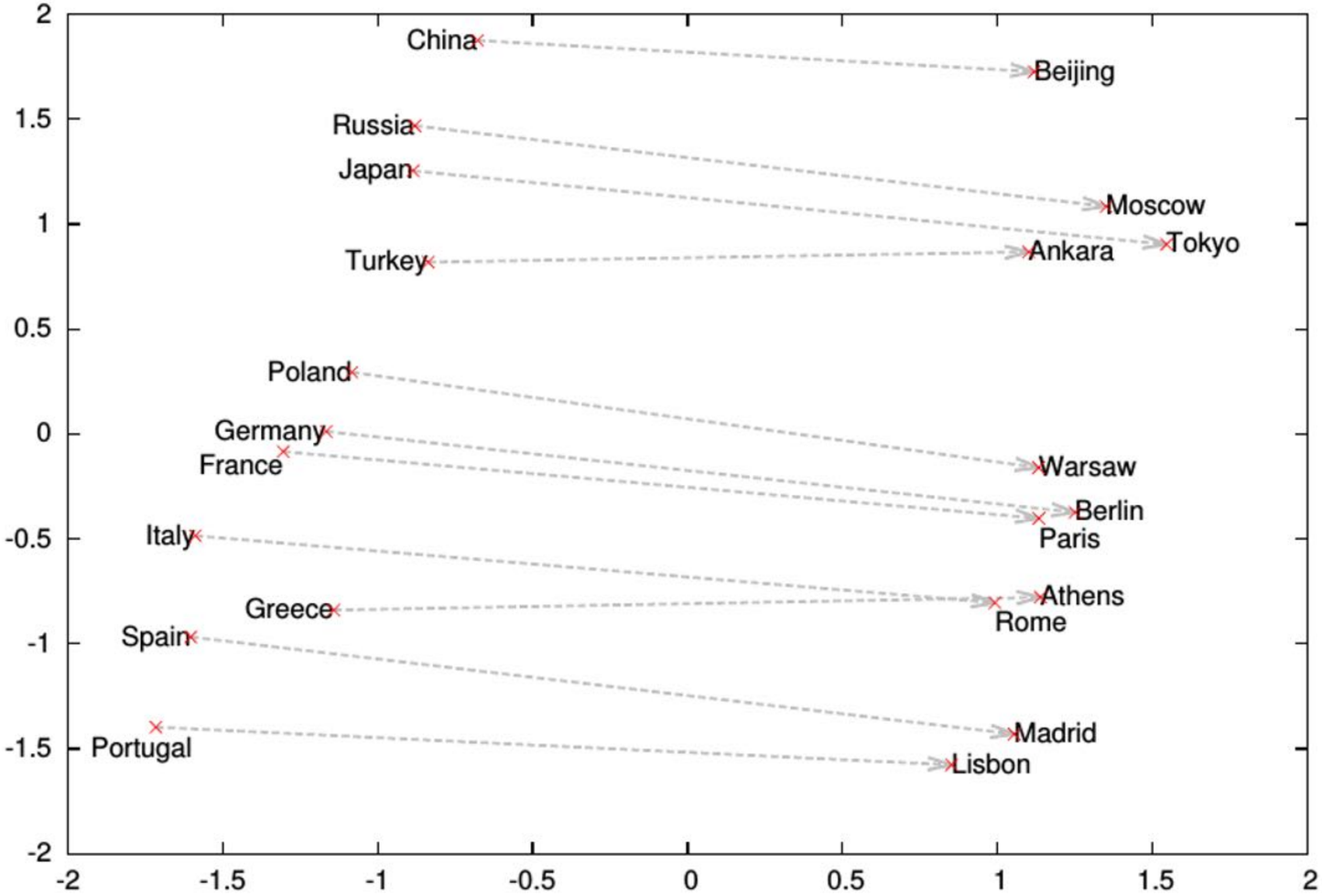
Transformation in eine interne Repräsentation (Vektor Länge N) des aktuell betrachteten Fensters. Rekonstruktion des mittleren Wortes muss daraus möglich sein.

Matrizen W und W' beinhalten für jedes Wort die Repräsentation als Vektor der Länge N.

Die Matrizen W und W' müssen gelernt werden.

Wortanalogien

- Man → Woman
- King → ?
- King - Man + Woman = Queen



https://opensource.googleblog.com/2013/08/learning-meaning-behind-words.html

Zusammenfassung

I. Empfehlungen

II. Sprachverarbeitung

    1. *TF.IDF*

    2. Clustering von Dokumenten

    3. Empfehlungen von Dokumenten

III. *Word2Vec*

Inhaltsübersicht

1. Programmiersprache Python
   a) *Einführung, Erste Schritte*
   b) *Grundlagen*
   c) *Fortgeschritten*
2. Auszeichnungssprachen
   a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
   a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
   a) *Git, GitHub*
5. Wissenschaftliches Rechnen
   a) *NumPy, SciPy*
6. Datenverarbeitung und -visualisierung
   a) *Pandas, matplotlib, NLTK*
7. Machine Learning (scikit-learn)
   a) *Grundlegende Ansätze (Datensätze, Auswertung)*
   b) *Einfache Verfahren (Clustering, …)*
8. DeepLearning
   a) **TensorFlow, PyTorch, HuggingFace Transformers**