



Information Retrieval

03: TF-IDF

Philipp Schaer, Technische Hochschule Köln, Cologne, Germany

Version: 2021-04-08

Technology
Arts Sciences
TH Köln

Boolesches Retrieval: Bibliothekskatalog

[Suche](#)[Merkliste](#)[Mein Konto / Verlängerung](#)[weitere Angebote](#)[Neue Suche](#)

Sprachauswahl [deutsch](#) | [englisch](#) | [französisch](#)

Sucheingabe

[Index](#)[Index](#)

✓ und
oder
und nicht

[Index](#)[Zurücksetzen](#)[Suchen](#)

Zusammenfassung Boolesches Retrieval

Suche mit einfachen booleschen/binären Entscheidungen (vorhanden / nicht vorhanden).

Vorteile:

- Simple Anfragen sind leicht zu verstehen
- Relativ leicht zu implementieren (**Term-Dokument-Matrix**)

Nachteile:

- Schwierig, genaue Anfragen zu spezifizieren
- Zu viel / zu wenig (**Feast or Famine**)
- Sortierung, aber nicht Ranking

Meistgenutzte IR-Modell bis zum Durchbruch des Web.

Daten- vs. Information Retrieval

IR- und DB-Systeme **verwalten beide Daten**, unterscheiden sich jedoch erheblich **im Zugriff auf die Daten**.

- Datenbankanfrage ist „**scharf**“ bzw. „**konkret**“:
select ISBN
from Buch
where Titel = “Information Retrieval”
- IR-Anfrage ist in der Regel „**unscharf**“ bzw. „**vage**“ formuliert:
*Finde alle Text-Dokumente, die sich mit dem Thema
“Information Retrieval” beschäftigen.*

Das Modell des Ranked Retrieval

- Im Gegensatz zum Booleschen Retrieval erhält der Nutzer keine **Menge** von Dokumenten, sondern eine nach **Relevanz geordnete Liste**.
 - Menge: ungeordnet
 - Liste: geordnet
- **Freitext-Anfragen:** Es werden keine spezielle Anfragesprache aus Operatoren und Ausdrücken verwendet, sondern ein oder mehrere Worte der natürlichen Sprache.
- Diese beiden Konzepte sind nicht zwangsläufig miteinander verbunden, gewöhnlich verbindet man mit Freitextsuchen allerdings ein Ranked Retrieval und umgekehrt.

Kein Feast-or-Famine-Problem

Wenn das Retrievalsystem **große Ergebnismengen** produziert, ist dies kein Problem und die Größe der Ergebnismenge kann ignoriert werden, da

- nur die Top-10/20/k Treffer ermittelt werden müssen,
- der Nutzer nicht überlastet wird.

Voraussetzungen:

- Der Ranking-Algorithmus arbeitet **zuverlässig**, relevantere Dokumente stehen weiter oben in der Ergebnisliste.
- Der Ranking-Algorithmus arbeitet **abhängig von der Anfrage** und liefert so gute Ergebnisse, die zur Anfrage passen!

Scoring: Grundlage des Ranked Retrieval

Ziel: Wir möchten die Dokumente in solch **einer Reihenfolge zurückgeben**, wie sie **am nützlichsten** für den Anwender ist.

Wie können wir eine **Rangordnung für Dokumente** in einer Datenmenge **in Abhängigkeit von der Anfrage** festlegen?

- Wir fügen eine Wert (**Score**) zu jedem Dokument hinzu (z.B. im Wertebereich $[0,1]$).
- Die Einteilung ist also **nicht mehr 0 oder 1**, wie im booleschen Retrieval.
- Dieser Score sagt aus, wie sehr **ein Dokument und eine Anfrage „matchen“** bzw. übereinstimmen.

Anfrage-Dokument-Score

Wir benötigen einen Weg eine **Score für ein Anfrage-Dokument-Paar** zu vergeben.

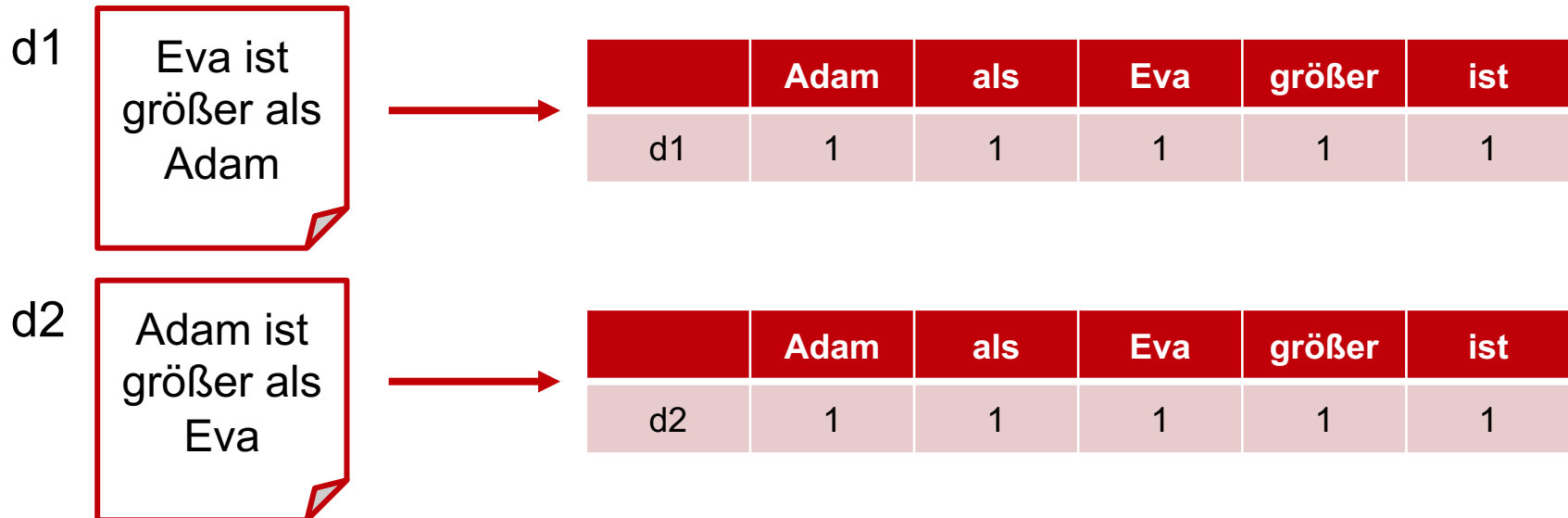
- Wir gehen von einer einfachen Ein-Wort-Anfrage aus.
- Wenn der Anfrageterm nicht im Dokument vorkommt ist $\text{Score} = 0$.
- Je häufiger der Anfrageterm im Dokument vorkommt, desto höher sollte der Score sein.

Wir werden hier einige Verfahren kennenlernen...

- **Vektorraummodell** (im Detail!)
- Probabilistisches Modell (ganz grob)
- Language Models (ganz, ganz grob)

Das Bag-of-Words-Modell

- Die Darstellung mittels Vektoren **ignoriert die Reihenfolge** von Wörtern in einem Dokument.
- „Adam ist größer als Eva“ und „Eva ist größer als Adam“ werden durch die **gleichen Term-Vektoren** dargestellt.
- Dies nennt man das **Bag-of-Words-Modell**.



Term-Dokument-Matrix (binär)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calphurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Jedes Dokument ist durch einen **binären Vektor** (besteht nur aus 0/1) repräsentiert.

1 wenn **Stück** das **Wort** enthält, ansonsten 0

Term-Dokument-Matrix (Häufigkeiten)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calphurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

Jedes Dokument ist durch einen Vektor der **Termhäufigkeiten** repräsentiert.

Termfrequenz tf

- Die **Termfrequenz** (*term frequency, Termhäufigkeit*) $tf_{t,d}$ eines Terms t in Dokument d ist die **Häufigkeit von t in d** .
- Wir möchten die Dokumente anhand Ihres Scores, der die Übereinstimmung von Anfrage und Dokument beschreibt ranken. Hierzu wollen wir die Termfrequenz verwenden.
- Aber wie...?

Die **reinen Termfrequenzen sind ungeeignet**, weil:

- Ein Dokument mit $tf = 10$ ist relevanter als ein Dokument mit $tf = 1$.
- Aber nicht unbedingt 10-mal relevanter...

Die **Relevanz steigt nicht proportional mit der Termfrequenz**.

Log-Termfrequenz-Gewichtung

Um die Wirkung der Termfrequenz zu dämpfen wird häufig mit der **logarithmierten Termfrequenz oder einer anderen Gewichtung** (Englisch: weight) gearbeitet:

$$w_{t,d} = \begin{cases} 1 + \log_{10}(\text{tf}_{t,d}), & \text{wenn } \text{tf}_{t,d} > 0 \\ 0, & \text{sonst} \end{cases}$$

“Das Gewicht des Terms t für das Dokument d ”

Der Score für ein Anfrage-Dokument-Paar ist die Summe über alle **Gewichtungen** aller Terme t , die sowohl in q als auch in d enthalten sind:

$$\text{Score}_{q,d} = \sum_{t \in q \cap d} w_{t,d} = \sum_{t \in q \cap d} (1 + \log_{10}(\text{tf}_{t,d}))$$

Kurzes Durchatmen..

$$w_{t,d} = \begin{cases} 1 + \log_{10}(\text{tf}_{t,d}), & \text{wenn } \text{tf}_{t,d} > 0 \\ 0, & \text{sonst} \end{cases}$$

$$\text{Score}_{q,d} = \sum_{t \in q \cap d} w_{t,d} = \sum_{t \in q \cap d} (1 + \log_{10}(\text{tf}_{t,d}))$$



We
know you
 don't like it but
IT'S
QUIZ TIME!

- *Wann ist der Score = 0?*
- *Was ist eigentlich mit „Gewichtung“ gemeint?*
- *Was war nochmal der Logarithmus?*

Was war nochmal der Logarithmus?

“Als Logarithmus einer Zahl bezeichnet man den Exponenten, mit dem eine vorher festgelegte Zahl, die Basis, potenziert werden muss, um die gegebene Zahl zu erhalten.”

z.B.: $10^x = 1000$ $x = ?$

→ Der Logarithmus berechnet den Exponenten x!

- $\log_{10}n$ = lg n (dekadischer Logarithmus)
- \log_2n = ld n (binärer Logarithmus)
- uvm...

POTENZ

$$a^n = z$$

LOGARITHMUS

$$\log_a z = n$$

Logarithmuswert
(= Exponent)

Basis Numerus

Text-Statistik (am Beispiel der IMDB)

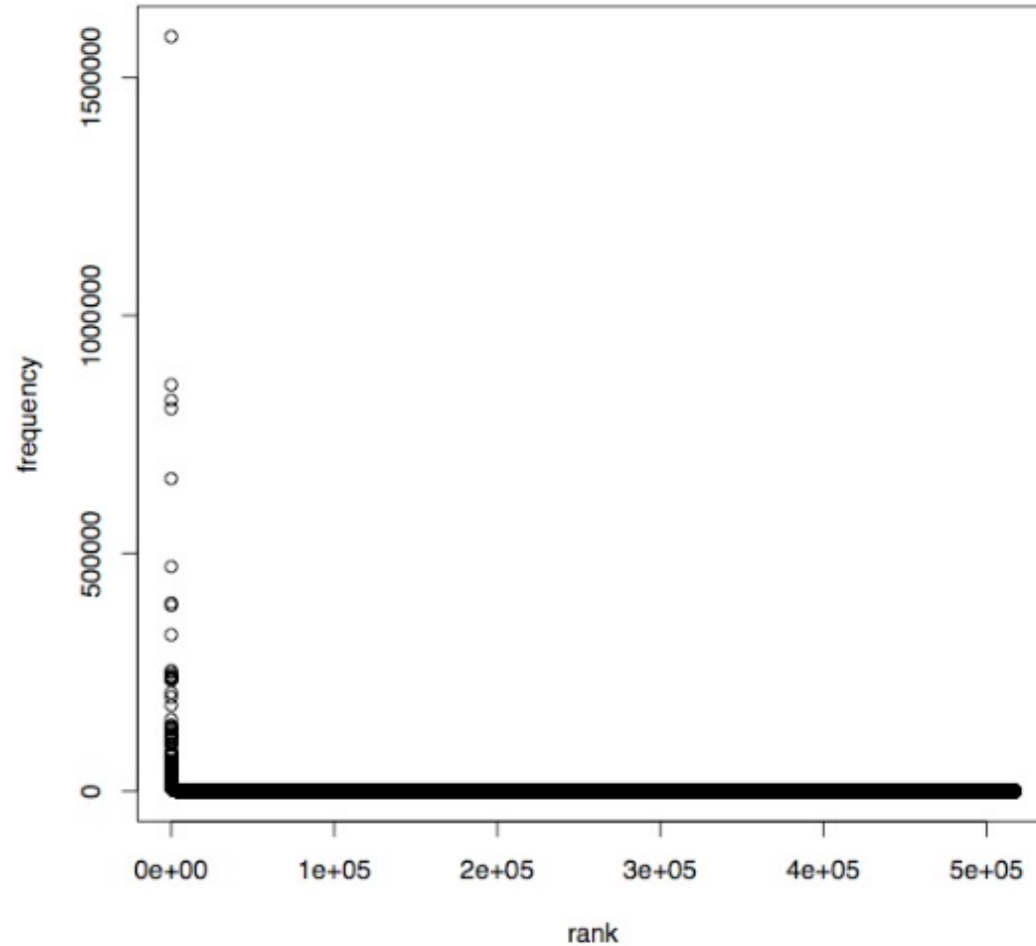
- IMDB – Internet Movie Database (<http://www.imdb.com>)
- Jedes Dokument ist hier eine Filmbeschreibung inkl. der Schauspieler, Regisseure, Plotbeschreibung etc.
 - Anzahl der Film-Datensätze: **230.721**
 - Anzahl der Terme: **36.989.629**
 - Anzahl der Terme (dublettenbereinigt): **424.035**

Stand der Daten ca. 2017.
Aktuelle Daten gibt es hier:
<https://www.imdb.com/interfaces/>

IMDB-Korpus-Statistik

Rang	Term	Häufigkeit	Rang	Term	Häufigkeit
1	the	1.586.358	11	year	250.151
2	a	854.437	12	he	242.508
3	and	822.091	13	movie	241.551
4	to	804.137	14	her	240.448
5	of	657.059	15	artist	236.286
6	in	472.059	16	character	234.754
7	is	395.968	17	cast	234.202
8	i	390.282	18	plot	234.189
9	his	328.877	19	for	207.319
10	with	253.153	20	that	197.723

Termverteilungen sind extrem schief!



Einschub: Zipfs Gesetz



George Kingsley Zipf hat sich grundsätzlich Gedanken zur statistischen Verteilung von Wörtern in Texten gemacht und Folgendes herausgefunden:

- „Term-frequency decreases rapidly as a function of rank“
- Oder anders: Wenn die Wörter eines Textes nach ihrer Häufigkeit geordnet werden, ist die Wahrscheinlichkeit p ihres Auftretens umgekehrt proportional zur Position n innerhalb der Reihenfolge.

$$f_t = \frac{k}{r_t}$$

- f_t = Termhäufigkeit von t in der Gesamtkollektion
- k = konstanter Wert
- r_t = Rang des Termes (nach Häufigkeit sortiert)

Das formen wir etwas um...

$$\frac{1}{N} \times f_t = \frac{1}{N} \times \frac{k}{r_t}$$

$$P_t = \frac{c}{r_t}$$

Wir teilen beide Seiten durch N (Gesamtanzahl der Terme in der Textkollektion) und erhalten so:

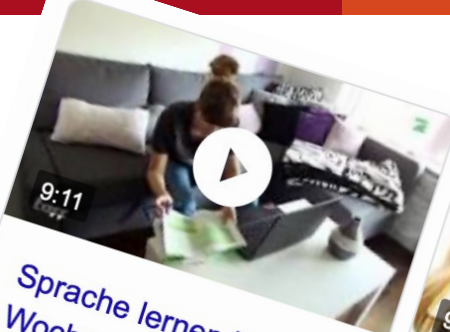
- P_t = Anteil des Terms t an der Kollektion von Texten
- c = konstanter Wert (für Englisch, z.B. 0,1)
- r_t = Rang des Termes (nach Häufigkeit sortiert)

Was heißt das konkret?

$$P_t = \frac{c}{r_t}$$

Für die Englische Sprache ($c = 0,1$) heißt das z.B.:

- Der häufigste Term ist für 10% des Textes verantwortlich
- Der 2. häufigste Term für 5%
- Der 3. häufigste Term für ca. 3%
- Die Top 10-Terme für knapp 30% des Gesamttextes
- Die Top 50-Terme für knapp 45% des Gesamttextes
- Das ist **fast die Hälfte** des gesamten Textes!



Sprache lernen in nur 3 Wochen? Mit dieser App funktioniert es!

ProSieben - 05.07.2017



SPRACHE LERNEN in nur 3 WOCHEN? Mit dieser App ...

taff
YouTube - 31.07.2017



Wie gut kann man in 4 Wochen Englisch lernen?

Babbel Deutsch
YouTube - 04.04.2018

Das könnte dich auch interessieren



Wie du mit einer App jede Sprache meistern kannst

ARTIKEL VON **ED M. WOOD**



8 spanische Wörter, die wir auf Deutsch brauchen

ARTIKEL VON **ED M. WOOD**



Sieben Sätze, mit denen du jeden in Berlin stark beeindruckst

ARTIKEL VON **THEA BOHN**

Share

Zipfs Gesetz visualisiert

Zipfs Gesetz

$$f_t = \frac{k}{r_t}$$

immer noch...

$$\log(f_t) = \log\left(\frac{k}{r_t}\right)$$

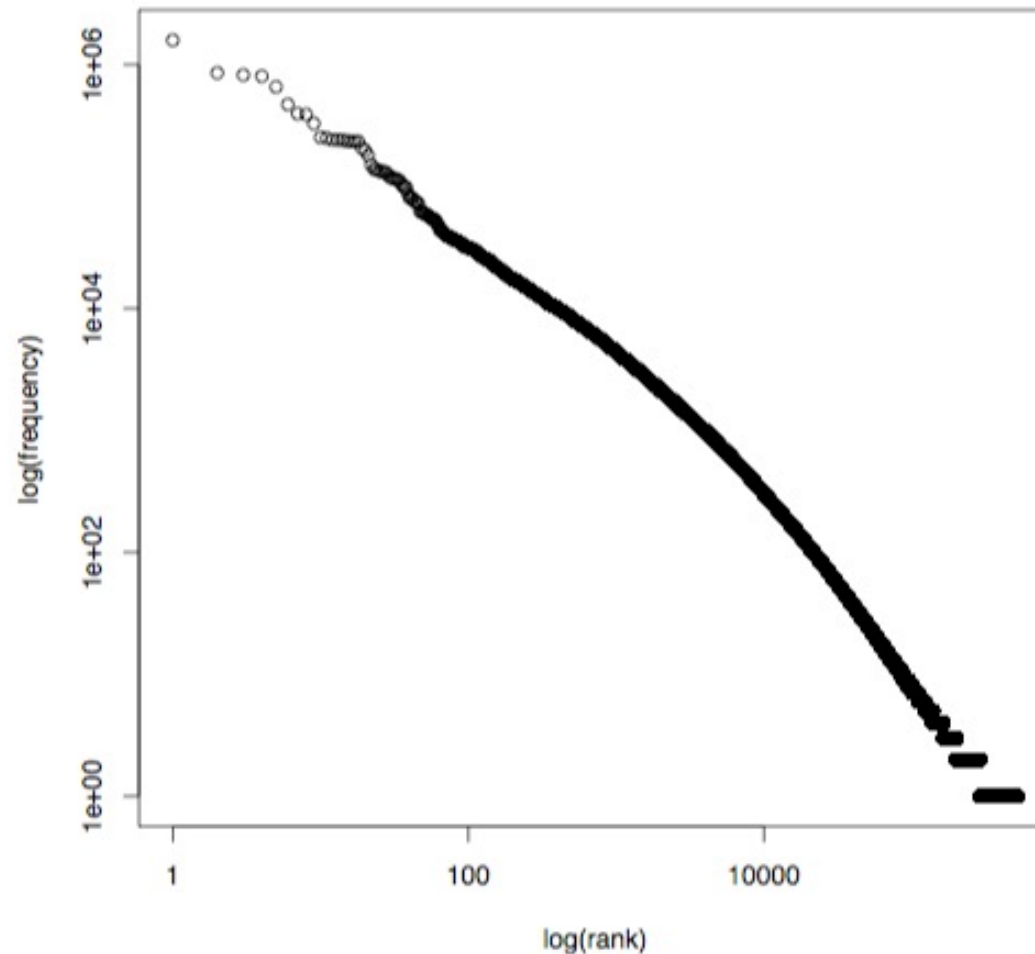
immer noch...

$$\log(f_t) = \log(k) - \log(r_t)$$

Das heißt:

- Wenn Zipfs Gesetz gültig ist, sollten wir $\log(f)$ und $\log(r)$ plotten können und eine gerade Linie sehen...

Zipfs Gesetz – Logarithmierte Darstellung



Zipfs Gesetz

Zipfs Gesetz ist gültig für

- Verschiedene **Korpusgrößen**
- Verschiedene **Textgattungen**
- Verschiedene **Themen**
- Verschiedene **Komplexitätsgrade** des Inhalts
- Verschiedene **Sprachen** (!)

Für uns heißt das:

- Die aussagekräftigen Begriffe stehen nicht in jedem Dokument!
- Zipfs Gesetz erlaubt es uns die nicht-aussagekräftigen Terme zu finden und mit Ihnen anders umzugehen...
- Wir sollten dies bei unseren Berechnungen berücksichtigen!

Karen Spärck Jones



- 1935 – 2007
- Informatiker-Professorin Cambridge University
- Eine der Pionierinnen der Computer-Linguistik und des Information Retrieval

Wichtige Arbeiten

- **TF-IDF** (term-frequency-inverse-document-frequency)
- 1994 wurde auf Grundlage Ihrer Arbeiten die erste erfolgreiche Web-Suchmaschine **AltaVista** gebaut

„I think it's very important to get more women into computing.
My slogan is: Computing is too important to be left to men.“

Dokumentfrequenz

Häufige Terme sind weniger informativ als seltene Terme.

- Stellen Sie sich einen Anfrageterm vor, der oft vorkommt, z.B. *hoch*, *sicher*, *teuer*...
- Ein Dokument, dass einen solchen Term beinhaltet ist wahrscheinlich relevanter, als eines das diesen Term nicht enthält. (das Grundprinzip von tf)
- Aber: Es ist kein sicherer Indikator für Relevanz.

Wir wollen **positive Gewichte** für Wörter wie *hoch*, *sicher*, *teuer*, **aber diese sollen niedriger sein als solche für seltene Terme.**

- Hierzu verwenden wir die **Dokumentfrequenz (df)**.

idf-Gewichtung

df_t ist die **Dokumentfrequenz** für t : Die Anzahl der Dokumente, die t enthält.

- df_t ist ein Maß für den **inversen Informationsgehalt** von t .
- $df_t \leq N$ (N ist die Anzahl aller Dokumente)

Wir definieren **idf** (inverse Dokumentfrequenz) von t als:

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

Wir verwenden $\log_{10}(N/df_t)$ anstelle von N/df_t um den Effekt von idf zu dämpfen.

Ein Beispiel

- Vokabular: {Kaffee, Tee, Tasse, Kanne, Wasser}
- Dokumente (Bag of Words)
 1. Kaffee, Kaffee
 2. Tee, Tee, Tasse, Kanne, Kanne
 3. Kaffee, Tasse, Tasse, Kanne
 4. Kaffee, Kaffee, Kaffee, Tee, Tasse, Tasse, Tasse, Kanne, Kanne, Kanne
 5. Kanne, Kanne, Wasser, Wasser
- Dokumentenfrequenz zu „Tasse“:
 - Enthalten in d_2, d_3, d_4
 - $df(„Tasse“) = 3$
- idf-Gewicht: $\log_{10} \left(\frac{5}{3} \right) = 0,22$

tf-idf-Gewichtung

Die **tf-idf-Gewichtung** von Termen ist das **Produkt der tf- und idf-Werte**:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} * \log_{10} \left(\frac{N}{\text{df}_t} \right)$$

tf-idf ist **DAS bekannteste Gewichtungsschema** im IR.

- Beachten Sie: Das „-“ ist ein Bindestrich, kein Minus.
- Andere Schreibweisen: tf.idf, tf x idf, tf*idf, TF*IDF, etc...
- Andere Kombinationen möglich... (z.B. mit diversen Logarithmen)

Der tf-idf-Wert steigt an für

- die Anzahl der **Termhäufigkeiten** in Dokumenten und
- die **Seltenheit** eines Terms in der Kollektion.

tf-idf-Varianten

Term frequency		Document frequency	
n (natural)	$tf_{t,d}$	n (no)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

Binär

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calphurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Jedes Dokument ist durch einen **binären Vektor** (besteht nur aus 0/1) repräsentiert.

1 wenn **Stück** das **Wort** enthält, ansonsten 0

Binar → Häufigkeiten

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calphurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

Jedes Dokument ist durch einen Vektor der **Termhäufigkeiten** repräsentiert.

Binar → Häufigkeiten → Gewichte

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	4,73	2,20	0	0	0	0,03
Brutus	0,12	4,73	0	0,06	0	0
Caesar	4,09	4,00	0	0,04	0,02	0
Calphurnia	0	0,78	0	0	0	0
Cleopatra	4,44	0	0	0	0	0
mercy	0,02	0	0,02	0,06	0,04	0,06
worser	0,02	0	0,01	0,01	0,01	0,04


Jedes Dokument wird nun
repräsentiert durch eine Vektor
mit **tf-idf-Gewichten** $\in \mathbb{R}^{|M|}$

Berechnung Score pro Dokument

$$Score(q, d) = \sum_{t \in q \cap d} \text{tf-idf}_{t,d}$$

Es gibt viele, sehr viele Varianten, wie

- tf berechnet wird (mit oder ohne Logarithmus)
- die Terme in der Anfrage gewichtet werden, und, und, und...



We
know you
don't like it but
IT'S
QUIZ TIME!

- *An dieser Stelle könnten wir bereits Dokumente ranken... Wie?*

Erweiterung des Booleschen Retrieval

Mit Hilfe der tf-idf-Gewichte ließe sich ein **Relevance Ranking** für das bisher bekannte Boolesche Retrieval umsetzen:

1. Zunächst würde man eine Ergebnismenge mit Hilfe einer booleschen Anfrage erzeugen,
 2. dann mit Hilfe der tf-idf-Gewichte pro Anfrageterm den Dokumenten einen Score zuweisen und
 3. letztlich nach diesem die Ergebnisliste sortieren/ranken.
- **Dies löst aber nicht die zuvor beschriebenen Probleme,** wie bspw. Feast-or-Famine...
 - Wir suchen immer noch die zur Anfrage ähnlichsten Dokumente und wollen ein richtiges **Ranked Retrieval, ohne die Probleme des Booleschen Modells!**