Analyzing Massive Data Sets
Summer Semester 2019

Prof. Dr. Peter Fischer

Institut für Informatik

Lehrstuhl für Datenbanken und Informationssysteme

Chapter 6: Fulltext Retrieval – Part 1: Retrieval Models

High-Dimensional Data and Similarity

- First *conceptual* and *algorithmic* part of the lecture
- Two core concepts:
  - **High-Dimensional Data**: Data items represented by many data points (hundreds, thousands, … possibly out of a much large space)
  - Analyzing a *single* or *few* dimensions insufficient to understand items
  - **Similarity/Distance**: Expressing pair-wise similarity over all features

- Applications:
  - **Finding Similar Items**: pairwise (this chapter)
  - **Clustering**: Identify structure / groups using similarity
  - **Retrieval**: Similarity between search expression and data set

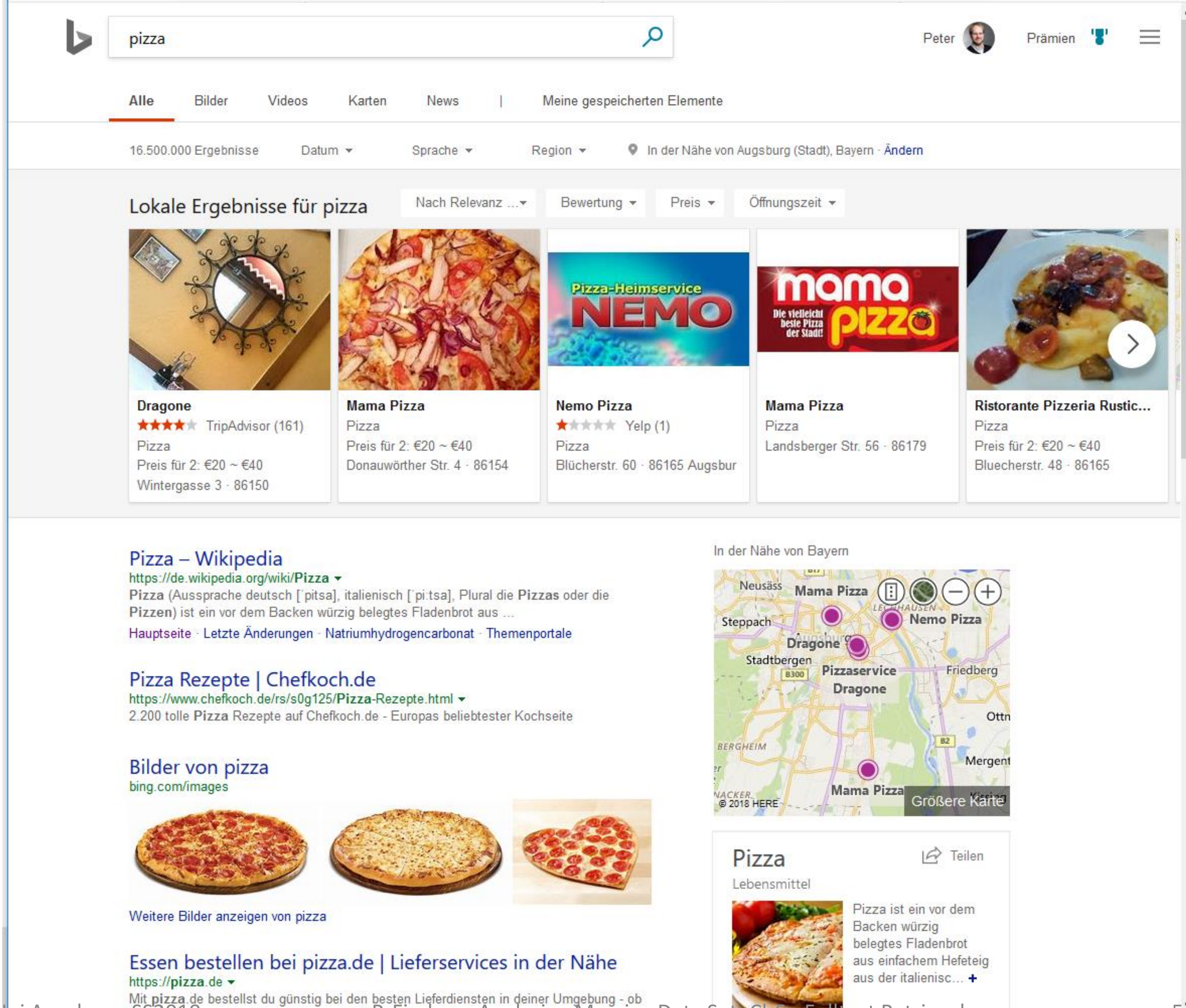
- Strategies for massive volumes:
  - Appropriate **retrieval models** to express **relevance**
  - **Scoring** approaches to determine **importance** of **terms**
  - Efficient **indexing** and **processing**

Simple Retrieval: Keyword Search

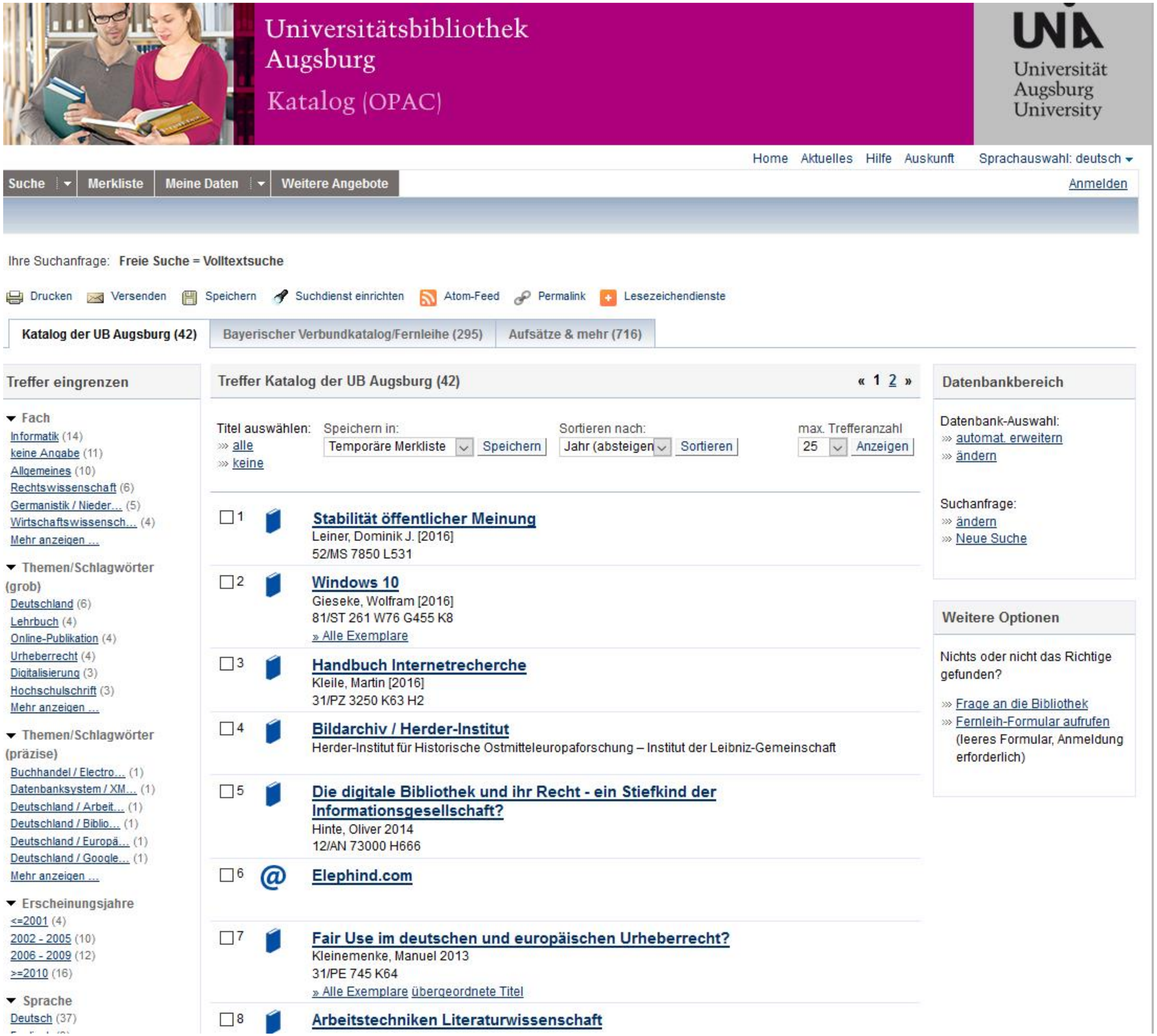
- So far, we performed limited data discovery
  - Find very similar items
  - Find a neighborhood with minimal structure
- New task
  - find **relevant** data items documents
  - from a **very large collection**
    (~ 50 billion web sites on Google)
  - Using **simple expressions**
- Common approach: keyword search
- Challenges:
  - Semantics
  - Relevance
  - Performance/Scalability

Starting Point: Single-Word Searches

- Query:
  - Single Word
  - e.g. "holidays", "food", "informatics"
- Semantics:
  - Return documents that contain the word
- What is a single word?
  - *Grundstücksverkehrsgenehmigungszuständigkeits-übertragungsverordnung*
  - Muvaffakiyetsizleştiricileştiriveremeyebileceklerimizdenmişsi nizcesinesiniz
- Open Issues:
  - Will searching for a single word yield useful results?
  - What is actually relevant?
  - How are results presented?



pizza
Peter
Prämien
Alle
Bilder
Videos
Karten
News
Meine gespeicherten Elemente
16.500.000 Ergebnisse
Datum
Sprache
Region
In der Nähe von Augsburg (Stadt), Bayern · Ändern
Lokale Ergebnisse für pizza
Nach Relevanz ...
Bewertung
Preis
Öffnungszeit
Dragone
TripAdvisor (161)
Pizza
Preis für 2: €20 ~ €40
Wintergasse 3 · 86150
Mama Pizza
Pizza
Preis für 2: €20 ~ €40
Donauwörther Str. 4 · 86154
Pizza-Heimservice
NEMO
Nemo Pizza
Yelp (1)
Pizza
Blücherstr. 60 · 86165 Augsbur
mama pizza
Die vielleicht beste Pizza der Stadt!
Mama Pizza
Pizza
Landsberger Str. 56 · 86179
Ristorante Pizzeria Rustic...
Pizza
Preis für 2: €20 ~ €40
Bluecherstr. 48 · 86165
Pizza – Wikipedia
https://de.wikipedia.org/wiki/Pizza
Pizza (Aussprache deutsch [ˈpitsa], italienisch [ˈpiːtsa], Plural die Pizzas oder die Pizzen) ist ein vor dem Backen würzig belegtes Fladenbrot aus ...
Hauptseite · Letzte Änderungen · Natriumhydrogencarbonat · Themenportale
Pizza Rezepte | Chefkoch.de
https://www.chefkoch.de/rs/s0g125/Pizza-Rezepte.html
2.200 tolle Pizza Rezepte auf Chefkoch.de - Europas beliebtester Kochseite
Bilder von pizza
bing.com/images
Weitere Bilder anzeigen von pizza
Essen bestellen bei pizza.de | Lieferservices in der Nähe
https://pizza.de
Mit pizza.de bestellst du günstig bei den besten Lieferdiensten in deiner Umgebung - ob
In der Nähe von Bayern
Neusäss
Mama Pizza
Steppach
Nemo Pizza
Dragone
Stadtbergen
Pizzaservice Dragone
Friedberg
Mergent
Mama Pizza
© 2018 HERE
Größere Karte
Pizza
Teilen
Lebensmittel
Pizza ist ein vor dem Backen würzig belegtes Fladenbrot aus einfachem Hefeteig aus der italienisc... +


Universitätsbibliothek
Augsburg
Katalog (OPAC)
Universität
Augsburg
University
Home Aktuelles Hilfe Auskunft Sprachauswahl: deutsch
Suche
Merkliste
Meine Daten
Weitere Angebote
Anmelden
Ihre Suchanfrage: Freie Suche = Volltextsuche
Drucken Versenden Speichern Suchdienst einrichten Atom-Feed Permalink Lesezeichendienste
Katalog der UB Augsburg (42)
Bayerischer Verbundkatalog/Fernleihe (295)
Aufsätze & mehr (716)
Treffer eingrenzen
Fach
Informatik (14)
keine Angabe (11)
Allgemeines (10)
Rechtswissenschaft (6)
Germanistik / Nieder... (5)
Wirtschaftswissensch... (4)
Mehr anzeigen ...
Themen/Schlagwörter (grob)
Deutschland (6)
Lehrbuch (4)
Online-Publikation (4)
Urheberrecht (4)
Digitalisierung (3)
Hochschulschrift (3)
Mehr anzeigen ...
Themen/Schlagwörter (präzise)
Buchhandel / Electro... (1)
Datenbanksystem / XM... (1)
Deutschland / Arbeit... (1)
Deutschland / Biblio... (1)
Deutschland / Europä... (1)
Deutschland / Google... (1)
Mehr anzeigen ...
Erscheinungsjahre
<=2001 (4)
2002 - 2005 (10)
2006 - 2009 (12)
>=2010 (16)
Sprache
Deutsch (37)
Treffer Katalog der UB Augsburg (42)
« 1 2 »
Titel auswählen:
» alle
» keine
Speichern in:
Temporäre Merkliste
Speichern
Sortieren nach:
Jahr (absteigen
Sortieren
max. Trefferanzahl
25
Anzeigen
1 Stabilität öffentlicher Meinung
Leiner, Dominik J. [2016]
52/MS 7850 L531
2 Windows 10
Gieseke, Wolfram [2016]
81/ST 261 W76 G455 K8
» Alle Exemplare
3 Handbuch Internetrecherche
Kleile, Martin [2016]
31/PZ 3250 K63 H2
4 Bildarchiv / Herder-Institut
Herder-Institut für Historische Ostmitteleuropaforschung – Institut der Leibniz-Gemeinschaft
5 Die digitale Bibliothek und ihr Recht - ein Stiefkind der Informationsgesellschaft?
Hinte, Oliver 2014
12/AN 73000 H666
6 Elephind.com
7 Fair Use im deutschen und europäischen Urheberrecht?
Kleinemenke, Manuel 2013
31/PE 745 K64
» Alle Exemplare übergeordnete Titel
8 Arbeitstechniken Literaturwissenschaft
Datenbankbereich
Datenbank-Auswahl:
» automat. erweitern
» ändern
Suchanfrage:
» ändern
» Neue Suche
Weitere Optionen
Nichts oder nicht das Richtige gefunden?
» Frage an die Bibliothek
» Fernleih-Formular aufrufen (leeres Formular, Anmeldung erforderlich)

Google

volltextsuche

Alle | Bilder | News | Shopping | Videos | Mehr | Einstellungen | Tools

Ungefähr 5.840.000 Ergebnisse (0,42 Sekunden)

Eine Volltextrecherche (oftmals auch **Volltextsuche**) ist das Auffinden von Texten in einer Vielzahl gleicher oder verschiedenartiger Dateien auf einem Computer, einem Server und/oder im Internet. Die Suchbereiche werden zuvor mit entsprechenden programminternen oder -unabhängigen Index-Werkzeugen indiziert.

Volltextrecherche – Wikipedia
https://de.wikipedia.org/wiki/Volltextrecherche

Informationen zu diesem Ergebnis   Feedback

Volltextrecherche – Wikipedia
https://de.wikipedia.org/wiki/Volltextrecherche
Eine Volltextrecherche (oftmals auch Volltextsuche) ist das Auffinden von Texten in einer Vielzahl gleicher oder verschiedenartiger Dateien auf einem Computer ...

Was ist eine Volltextsuche? - Lookeen
lookeen.de/blog/was-ist-eine-volltextsuche/
14.07.2016 - Was bedeutet der Begriff Volltextsuche eigentlich? Und wie genau kann Ihnen die Volltextsuche weiterhelfen? Hier erfahren Sie mehr!

Volltextsuche/Volltextindex - bitfarm-Archiv DMS
https://www.bitfarm-archiv.de/dokumentenmanagement/glossar/volltextsuche.html
Neben anderen Suchmethoden bieten Dokumentenmanagementsysteme wie bitfarm-Archiv Dokumentenmanagement die Volltextsuche an.

Volltextsuche für Dokumente und über 100 Dateiformaten - Amagno
https://amagno.de/volltextsuche/
Leistungsstarke Volltextsuche bei der es unerheblich ist, wie Dateien benannt oder wo abgespeichert sind. Mit AMAGNO finden Sie Dateien innerhalb ...

Volltextsuche | Microsoft Docs
https://docs.microsoft.com/de-de/sql/relational-databases/search/full-text-search
10.04.2018 - Wenn Sie bei der Installation von SQL Server nicht die Volltextsuche ausgewählt haben, führen Sie SQL Server-Setup erneut aus, um sie ...
Übersicht · Volltextsuchabfragen · Architektur der Volltextsuche · Verarbeitung der ...

Volltextsuche - MSDN - Microsoft
https://msdn.microsoft.com/de-de/library/ms142571(v=sql.120).aspx
Veröffentlicht: November 2016. Mit der Volltextsuche in SQL Server und Azure SQL-Datenbank können

Naïve implementation

- **Scan all** documents
- Test for **regular expressions** (even more expressive power than simple keywords + well-defined semantics)
- Command Line version: grep
- Parallelizable (think Hadoop, Spark)
- Surprisingly **effective** on **small** to **medium** collections
  - Substring match/simple Regexp: >> 1 GB/s per CPU core
  - mostly I/O-bound (how fast is your disk/SSD/network/RAM in sequential access)
- Large collections take a significant of time
  - 100 GB is already in the range of minutes from RAM
  - The Web is at least several petabytes of text (50 B pages * 50 KB/page)

Index-Based Keyword Search

- Remember Inverted Indexes from chapter 4
- Keep a list of documents containing a term

  Vocabulary = {Term1, Term2, Term3, Term4}
  Document Set = {D_1, D_2, D_3, D_4}

  Term1 : D_1, D_2, D_3, D_4
  Term2 : D_1, D_2
  Term3 : D_1, D_2, D_3
  Term4 : D_1
- Each document is stored only once, regardless of the number of occurrences of the term in it
- Two considerations:
  - Typically, the document list are sorted by document id
  - Instead of document IDs, we can store pairs of (ID, count/score)

Multi-word-queries

- Getting all results for single keyword query obvious
  Term1: $\{D_1, D_2, D_3, D_4\}$
- If we already have **scoring information** per document, we can **order** accordingly (and stop after Top K)
- What happens if you add **more terms**?

  "free beer"
- Do **all terms** need to show up?
- Do we care more about **beer** or **free**?
- How about

  "free delicious beer"?

- We need a (formal) description on the meaning

Retrieval Models

- Provide a **mathematical framework** for defining the search process (basis of many ranking algorithms)
- Good models should **produce outputs** that **correlate** well with **human decisions** on **relevance**
- Progress in retrieval models has corresponded with improvements in effectiveness over the last 10 years
- **Relevance** is a complex concept:
  - **difficult** for a person to explain why one **document** is more **relevant** than another
  - **Many factors** to consider
  - People often **disagree** when making relevance judgments

Assumptions on Retrieval Models

- ***topical*** vs. ***user relevance***
  - a document is relevant to a query if it is judged to be on the **same topic**
  - A web page containing a biography of Abraham Lincoln would topically relevant to the query *"Abraham Lincoln"* and would also relevant to the queries *"U.S. presidents"* and *"Civil War"*
  - However, a document just containing a list of all U.S. presidents may not be considered relevant because we are looking for more detail on Lincoln's life
- ***binary*** vs. ***multi-valued*** **relevance**
  - a document is either **relevant or not**
  - e.g., we may consider the list of U.S presidents to be less topically relevant than Lincoln's biography, but certainly more relevant than an advertisement for a Lincoln automobile
  - Hence, some retrieval models introduce **relevance** as a **multi-valued variable** (e.g., *relevant, non-relevant, unsure)*
- *...*

Boolean Retrieval

- Simplest and oldest IR model
  - **Documents** = set of words (index terms)
  - **Query language** = **Boolean expressions** over index terms
  - **Result** = Set of documents satisfying the query formula
  - Query usually specified using Boolean operators:
    AND, OR, NOT
  - Two possible outcomes for query processing:
    TRUE and **FALSE**
  - "**Exact-match**" retrieval, since documents are retrieved if they match the query, otherwise not
- Simplest form of **ranking: Binary ranking function**, i.e., 0/1-valued
- Retrieval based on **set membership**
  - "Find all documents indexed by the word 'tropical'!"
  - "Find all documents indexed by the word 'tropical' or/and 'fish'!"

Boolean Connectives

- Boolean Algebra
  - Conjunction
  - Disjunction
  - Negation

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\neg$	
0	1
1	0

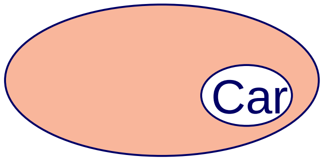
Example

- **Document D1** = {Lincoln, automobile, car}
- **Document D2** = {president, Lincoln, biography}
- **Document D3** = {Lincoln, Gettysburg, president}
- **Document D4** = {Ford, Hazel, president, Lincoln, Mercury, car}

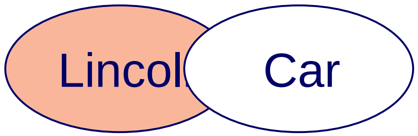
- Query Q1 = "*lincoln*"
  - Result: {D1, D2, D3, D4}
- Query Q2 = *"president AND lincoln"*
  - Result: {D2, D3, D4} (both word must be contained)
- Query Q3 = *"president AND lincoln AND NOT (automobile OR car)"*
  - Result: {D2, D3} (take away documents contained latter two words)
- Queries can be *quite complex:*
- Query Q4 = *"president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)*

Caveats of Boolean Retrieval

- Exclusive use of negation will result in large result sets!
- Query Q5 = "**NOT** car"



- To match *natural language* better,
  "BUT NOT" can be used instead of "AND NOT"
- Query Q6 = "lincoln **BUT NOT** car"



- Use **"OF"** to search for subsets of a given size:
  - Query Q7 = "2 **OF** {lincoln, biography, president}"
  - Q7 ≡ "(lincoln **AND** biography)
    OR (lincoln **AND** president)
    OR (biography **AND** president)"

Boolean Operations on Lists

Direct mapping of connectives to set operators:

- result of "lincoln **AND** biography" =
  {result of "Lincoln"} ∩ {result of "biography"}
- result of "lincoln **OR** biography" =
  {result of "Lincoln} ∪ {result of "biography}

- How expensive is it to intersect or merge two lists?
  - If the two lists ordered in the same way: linear complexity

Example:

- **Lincoln:** 13, 57, 61, 114, 987, ...
- **Gettysburg:** 5, 23, 57, 63, 114, 257, ...

- For k intersections/unions L1 , L2 , …, Lk, do a pairwise intersection L1 and L2 -> L12, L12 and L3 → L123, …
- Ordering the merge/intersection starting with the smallest list creates smallest intermediate results

CNF and DNF for Query Processing

- **Idea: Normalize** queries for effective processing
- **Conjunctive Normal Form (CNF)**
  - A propositional formula is in **CNF** if it is a **conjunction of clauses**
  - A clause is a disjunction of literals
  - A literal is a variable or its negation
- **Disjunctive Normal Form (DNF)**
  - A propositional formula is in DNF if it is a **disjunction of conjunctive clauses**
  - A conjunctive clause is a conjunction of literals

Any propositional formula can be converted into an equivalent formula that is in CNF or DNF

Normalization Example

- Query Q8 = "lincoln AND ((biography AND gettysburg) OR president)"
- **CNF**
- Q8_C = "lincoln AND (biography OR president) AND (gettysburg OR president)"
  - Compute unions (**might become very large**)
  - Compute intersections
- **DNF**
- Q8_D = "(lincoln AND biography AND gettysburg) OR (lincoln AND president)"
  - Compute intersections (**smaller intermediate results**)
  - Compute unions

Boolean Retrieval - Assessment

- **Advantages**
  - Simple query paradigm, **easy to understand**
  - **Results** are **predictable**, relatively easy to explain to the users
  - Many **different features** can be incorporated
  - **Efficient processing** since many documents can be eliminated from search
- **Disadvantages**
  - **Effectiveness depends** entirely on **user**
  - **Simple queries** usually **don’t work** well due to the lack of a sophisticated ranking algorithm
  - Complex queries are difficult
  - A binary ranking function returns a **set of results**, i.e., it is **unordered**
  - **Similarity queries** are not supported
  - Usually, most of the documents found are relevant;
    but **many relevant documents are not found**

Fuzzy Retrieval

- **Observation**:
  - **Not all index terms representing a document are equally important, or equally characteristic**
  - Are there any synonyms to the document's terms?
  - Does a term occur more than once in the document?
- Can we assign **weights** to terms in documents?
- **Idea:**
  - Improve Boolean retrieval (Extended Boolean Retrieval)
  - Describe documents by **fuzzy sets** of terms
  - No binary set membership, but **graded membership**
  - **Advantage:** Fuzzy (i.e., **ordered**) result sets

Fuzzy Retrieval

- **Fuzzy sets:**

  {gettysburg, lincoln, president} ->

  {gettysburg/**0.4**, lincoln/**0.9**, president/**0.8**}

- **Open Problems:**
  - How to deal with **fuzzy logic?**
  - Where to get **membership degrees** from?

Fuzzy Logic

- Developed by **Lotfi Zadeh** (1965)
- Possible **truth values** are not just "true" (1) and "false" (0) but **any number in [0;1]**
- Designed to deal with classes whose **boundaries** are **not well defined**
- **Key idea:** Introduce the notion of a degree of membership associated with the elements of a set

Zadeh Operators

- How to **translate** Boolean operators into **fuzzy logic**?
  - Propositional (=classic Boolean) logic should be a special case
  - Fuzzy operators should have **"nice" properties:** *commutativity, associativity, monotony, continuity, …*

Zadeh's original operators:

- Let μ(X) denote the truth value of the variable X
- **Conjunction/Intersection:**
$$\mu(A \wedge B) = \min(\mu(A), \mu(B)$$
- **Disjunction/Union:**
$$\mu(A\ v\ B) = \mathrm{m}ax(\mu(A), \mu(B)$$
- **Negation:**
$$\mu(\neg A) = 1 - \mu(A)$$

Example

- **Document** ={gettysburg/**0.4**, lincoln/**0.9**, president/**0.8**}
- **Query** = *"(gettysburg **BUT NOT** lincoln) **OR** president"*

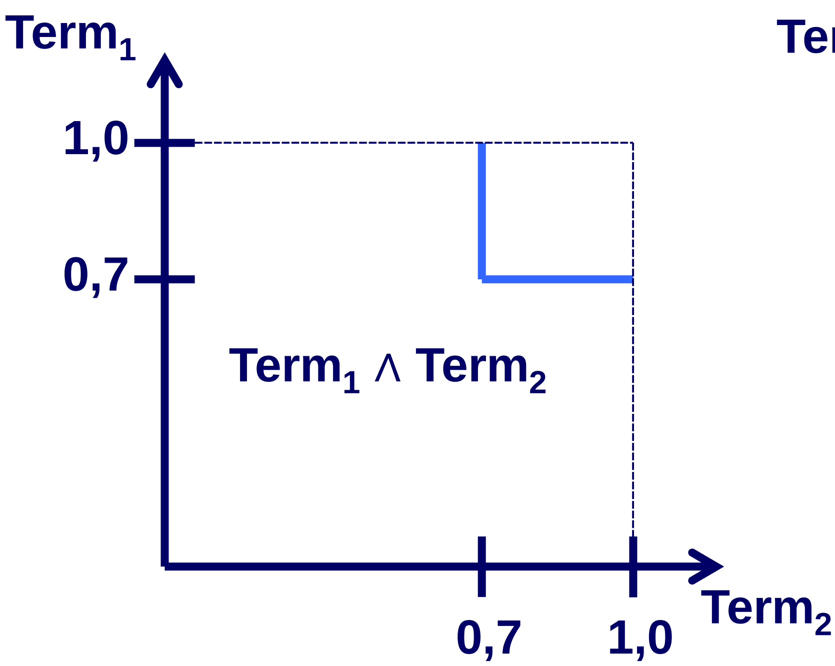
- Document's degree of **query satisfaction** is
  - **μ(**gettysburg **BUT NOT** lincoln) = 0.1
  - **μ**((gettysburg **BUT NOT** lincoln) **OR** president) **= 0.8**

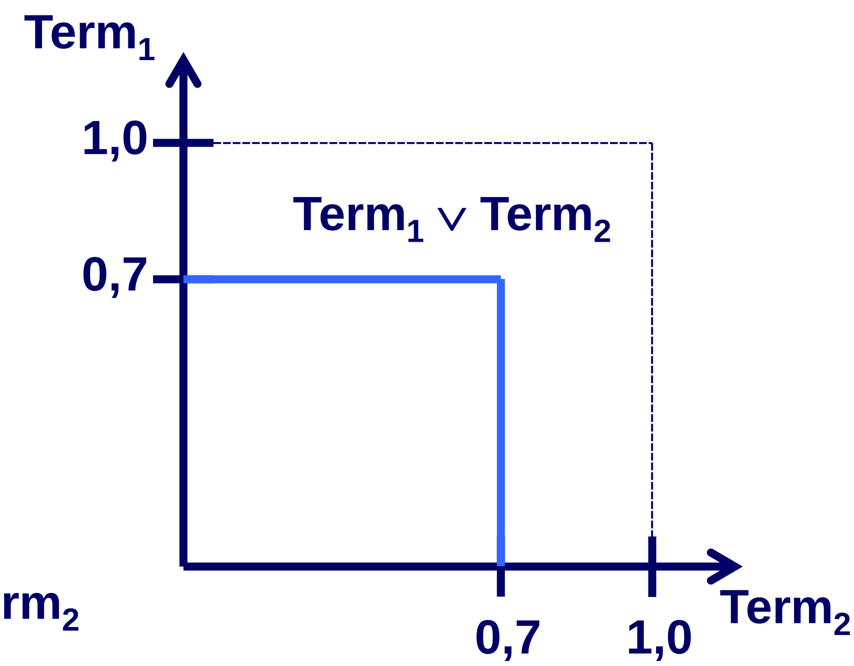
Problems with Fuzzy Logic

- Zadeh operators indeed have **"nice" properties**
- But sometimes, they behave strange:
  - **Document1** = {gettysburg/**0.4**, lincoln/**0.4**}
  - **Document2** = {gettysburg/**0.3**, lincoln/**1**}
  - **Query:** *"gettysburg AND lincoln"*
  - **Result** = {Document1/**0.4**, Document2/**0.3**}
- Where to get **fuzzy membership degrees** for index terms from?
  - Take **crisp bag of words representation** of documents, and **convert it to a fuzzy set representation**

Intuitive?

- All documents lying on the blue line are satisfying the query equally well (degree 0.7):





Fuzzy Index Terms

- Approach by **Ogawa** et al. (1991):
  - **Idea**: **Expand** the set of **index terms** in the query with **related terms** such that additional relevant documents can be retrieved by a user query
  - A thesaurus is constructed by defining a term-term correlation matrix
    (**keyword connection matrix)**
    - Its **crisp terms** (use fuzzy degree 1)
    - • **Additional terms** being similar to these crisp terms (**use degree <= 1**)

{gettysburg, lincoln, president} → {gettysburg/**1**, lincoln/**1**, president/**1,**
16th president of US,
premier, head of state, CEO}

1. Use the **Jaccard index** to get a notion of **term similarity**
2. Compute **fuzzy membership degree** for each term-document-pair using this similarity

Jaccard Index

- **Document1** = {gettysburg, president, biography}
- **Document2** = {gettysburg, president, lincoln}
- **Document3** = {gettysburg, biography}

c(t,u)	**gettysburg**	**president**	**biography**	**lincoln**
gettysburg	1	0.67	0.67	0.33
president		1	0.33	0.5
biography			1	0
lincoln				1

Fuzzy Index Terms: Ogawa (1991)

The **fuzzy degree of membership** between a **document D** and an index **term t** is

$$W(D,t) = 1 - \prod_{u \in D} (1 - c(t,u))$$

- *1-c(t,u)* is the fraction of documents containing one of term *t* and term *u* but not both
- If *D* contains term *t,* than W(D,t) = 1
- **Idea**: Give terms a high fuzzy membership degree that usually occur together with the other document terms; those terms will capture the document’s topic best

Example for Ogawa Fuzzy Index Terms

- **Document1** = {gettysburg, president, biography}
- **Document2** = {gettysburg, president, lincoln}
- **Document3** = {gettysburg, biography}

c(t,u)	gettysburg	president	biography	lincoln
gettysburg	1	0.67	0.67	0.33
president		1	0.33	0.5
biography			1	0
lincoln				1

- **Membership degree of term t w.r.t. document D**

c(t,u)	gettysburg	president	biography	lincoln
Document1	1	1	1	0.67
Document2	1	1	0.78	1
Document3	1	0.78	1	0.33

Fuzzy Retrieval - Assessment

Cons:

- Computation of fuzzy membership **weights** usually **difficult**
  - Main problem: All weights must be within [0,1]
- Lack of intuitive **query processing**
  - But: There are many other ways to define fuzzy conjunction and disjunction (using **t-norms** and **t-conorms**)

Pros:

- Supports **non-binary assignment** of index terms to documents
  - It is possible to find relevant documents that do not satisfy the query in a strict Boolean sense
- **Ranked** result sets

Bag of Words Models

- Propositional formulas are mathematically handy, but often hard to use for querying
  (⇔SQL: formal approach with "nice" syntax accepted)
- Alternative: **Bag-of-words queries** ("virtual documents")
- **Sketch the document** that is requested (similar to QBE instead of SQL)
- **Advantage:** Comparing queries to documents gets simpler!
- Many successful retrieval models are based on bag-of-words queries
  - Coordination Level Matching
  - Vector Space

Coordination Level Matching

- **Idea:** Documents whose index records have

  n **different terms** in common with the query are
  more relevant than documents with
  n-1 **different terms** held in common

- The **coordination level** (also called "size of overlap") between a query Q and a document D is the **number of terms they have in common (remember similarity metrics)**

- **How to answer a query?**
  - 1. **Sort** the document collection **by coordination level**
  - 2**. Return the head** of this sorted list to the user
    (say, the best 10 documents)

CLM Example

- **Given document collection**
  - **Document D1** = {Lincoln, president, Gettysburg}
  - **Document D2** = {Lincoln, president, bibliography}
  - **Document D3** = {Lincoln, Gettysburg}
- **Queries**
  - Query1 = *{president, gettysburg}*
  - **Result:**
    - D1 (2)
    - D2, D3 (1)
  - Query2 = *{bibliography, president, gettysburg}*
  - **Result:**
    - D1, D2 (2)
    - D3 (1)

Vector Space Model

- **Documents** and **queries** are represented by a **t-dimensional vector** of **term weights**, where *t* is the number of index terms (words, terms, phrases, etc.)
- Usually, ***t* is very large:** hundreds of thousands or even *millions* of dimensions
- A document D_i is represented by a **vector** of **index terms**, where d_{ij} represents the weight of the *j*-th term
- Obvious first choice: Represent documents by its **incidence vectors**
- A query is represented the same way as documents, i.e., a vector of *t* weights, where q_j is the weight of the *j*-th term in the query

Vector Space Model

- A document collection containing *n* documents can then be represented by a matrix of term weights
  - each row represents a document
  - each column describes weights that were assigned to a term for a particular document

$$D_i = (d_{i1}, d_{i2}, \dots, d_{it})$$

$$Q = (q_1, q_2, \dots, q_t)$$

$$\begin{array}{cccc} & Term_1 & \dots & Term_t \\ Doc_1 & d_{11} & \dots & d_{1t} \\ \vdots & \vdots & \ddots & \vdots \\ Doc_n & d_{n1} & \dots & d_{nt} \end{array}$$

Example

- Term-document matrix **rotated**, terms are rows, documents are columns
- **Term weights** are simply the **count** of the **terms** in the document
- Stopwords are not indexed in this example
- Words have been stemmed
- e.g., Document **D3** is represented by
  (1,1,0,2,01,0,1,0,0,1)
- e.g., the query "tropical fish" would be (0,0,0,1,0,0,0,0,0,0,1)

D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	**Documents**			
	D_1	**D_2**	**D_3**	**D_4**
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Vectors: Ranking and Distance Functions

- One can use simple diagrams to visualize documents and queries, e.g., vectors in a 3-dimensional space
- Documents can be **ranked** by computing the **distance** between the points representing the **document** and the **query**
- A ***similarity measure*** is used, often
  - **Euclidean distance**
  - **Cosine correlation**
- The **highest scores** are the most similar to the query
- Ranking based on the vector space model is able to **reflect term importance** and the **number of matching terms**, which is not possible in Boolean retrieval

Terms Weights – TF/IDF

- **Repetition of words is an indication of emphasis (Luhn, 1961)**
  - **Already present in CLM**
- **Problem:** some words are frequent in many documents, regardless of the content
- university … , **57 5** , … … … , **123 2** , …
- of … , **57 14** , … … … , **123 23** , …
- augsburg … , **57 3** , … … … , **123 1** , …

Aggregate per Document (here SUM)

- Document … , **57 22** , … … … , **123 26** , …

- Do stopwords help?
  - Remove only the top K "offenders" (what threshold)
  - Missing phrases ("Flight from London to Paris")
  - Dependent on language and context
  - Binary approach, not weighted

(Inverse) Document Frequency

- Idea: **Specificity** (Spärck Jones, 1972)
  - The **more documents** a term **occurs** in, the **less discriminating** the term is, and consequently, the **less useful** it will be in retrieval
- The number of documents containing a particular word

  $\mathbf{df}_{\text{university}}$ = 16.384 , $\mathbf{df}_{\text{of}}$ = 524.288 , $\mathbf{df}_{\text{augsburg}}$ = 1.024
- Inverse document frequency (**idf**)

  idf = $\log_2$ (N / df) N = total number of documents
- For the example df scores above and N = 1.048.576 = 220

  $\mathbf{idf}_{\text{university}}$ = 6 , $\mathbf{idf}_{\text{of}}$ = 1, $\mathbf{idf}_{\text{augsburg}}$ = 10
- Motivation for logarithm:
  - Word frequencies vary significantly, often Zipf or Power Law
  - A word showing up several orders of magnitude more frequently is not several orders of magnitude more/less discriminating
  - Without the **log2** , small differences in the value of **df** would have too much of an effect

Combining TF and IDF

- Reconsider our earlier **tf** only example
- university … , **57 5** , … … … , **123 2** , …
- of … , **57 14** , … … … , **123 23** , …
- augsburg … , **57 3** , … … … , **123 1** , …
- Document … , **57 22** , … … … , **123 26** , …

- Now combined with **idf** scores from previous slide
- university … , **57 30** , … … … , **123 12** , …
- of … , **57 14** , … … … , **123 23** , …
- augsburg … , **57 30** , … … … , **123 10** , …
- Document … , **57 74** , … … … , **123 45** , …

Issues with TF

- IDF helps significantly, but the basic form of TF causes problems

Let D1,D2 documents, w word/token:

- 1) D1 is longer than D2:
  - -> often tf(D1,w) > tf(D2,w)
    (because of length, not because of relevance)
- 2) D1 and D2 have same length,
  tf(D1, w) twice of tf(D2, w)
  - -> Is D2 twice as relevant?

Refining TF and IDF

- For 1), normalize tf for each document

$$ntf(d,t) = \frac{f(d,t)}{\sum_{j=1}^{n} f(d,j)}$$

where f(d,t) # of occurrences of t in d
(our previous tf)

- Most common TF-IDF variant

$$w(d,t) = ntf(d,t) * \log\left(\frac{N + 0.5}{df(t) + 0.5}\right)$$

- +0.5 smoothing for very common/very rare terms

BM25 (aka Okapi Best Match 25)

- Popular and well-performing TF-IDF derivative

$$BM25(d,t) = tf^{*} * log\left(\frac{N}{df(t)}\right)$$

$$tf^{*} = tf * \frac{(k+1)}{k\left(1-b+b*\frac{DL}{AVDL}\right)+tf}$$

  - DL Document length
  - AVDL average document length
- Standard values for BM25: k=1.75 [1.2;2], b=0.75
- Binary/Boolean: k=0, b=0
- Standard tf-idf: k=inf, b=0

Motivating BM25 – Score growth

- tf and tf* should share the following properties

1. Tf*=0 iff tf=0

2. Tf* increases when tf increases

3. Tf* grows to a fixed limit if tf grows to inf

- Simplest(?) formula to fulfill 1-3

- $tf^+ = tf * \frac{k+1}{k+tf}$

Motivating BM25 – Document Lengths

- $tf^+ = tf * \frac{k+1}{k+tf}$
- Normalize by document lenght: tf -> $\frac{tf}{\alpha}$
- $\frac{tf}{\alpha} * \frac{k+1}{k+\frac{tf}{\alpha}} = tf * \frac{k+1}{k*\alpha+tf}$
- α expresses the amount of document length normalization
- Full normalization: $\alpha = \frac{DL}{AVDL}$
- Some tuneable normalization: $\alpha = (1-b) + b(\frac{DL}{AVDL})$
- No normalization: b=0 -> α=1

Implementing Ranking – Building Indexes

- First compute the inverted lists with **tf** scores
  - For each document, iterate over each word
  - Add (word,count) entries to list for document
- Along with that compute the document length (DL) for each document, and the average document length (AVDL)
- You can measure DL (and AVDL) via the number of words
- Make a second pass over the inverted lists and replace the **tf** scores by **tf* · idf** scores
- Note that the **df** of a word is just the length (number of postings) in its inverted list

Assessment of the Vector Space Model

Pros:

- Simple and clear computational framework for ranking
- Any similarity measure or term weighting scheme could be used
- **Intuitive querying** yields high usability
- Founded on "real" document rankings, not based on result sets
- **Highly customizable** and adaptable to specific collections
  - Distance / similarity functions
  - Normalization schemes
  - Methods for term weighting
- **High retrieval quality**
- Relevance feedback possible

Cons:

- Assumption of term independence
- High-dimensional vector spaces, specialized algorithms are required
- Relies on **implicit assumptions,** which do not hold in general:
  - **Cluster hypothesis:** "Closely associated documents tend to be relevant w.r.t. the same queries
  - **Independence/orthogonality assumption:** "Whether a term occurs in a document, is independent of other terms occurring in the same document"