CS 589: Text Mining and Information Retrieval

LECTURE 1: Introduction to Information Retrieval - Complete Study Guide

Course Information:

- Instructor: Professor Susan Liu (xliu127@stevens.edu)
- Office Hours: Monday 3:30-5:30 PM (Discord)
- Class: Tuesday 6:30-9:00 PM
- Grading: 4 Assignments (40%) + Midterm (25%) + Pop Quiz (5%) + Final (30%)

📋 Table of Contents

---

1. What is Information Retrieval?

1.1 Core Definition

Information Retrieval (IR) is the process of obtaining information system resources that are relevant to an information need from a collection of those resources.

Key Components:

- **Information Need**: What the user actually wants to know (mental concept)
- **Query**: How the user expresses that need (text keywords)
- **Documents**: The collection to search

- **Relevance**: How well a document satisfies the information need

1.2 Information Need vs Query

Critical Distinction (appears on exams!):

```
Information Need ≠ Query
```

Example from slides:

```
Information need: "name of the first US president"
↓
Query: "first president of the united states"
↓
Result: George Washington
```

The query is an imperfect representation of the need!

1.3 Types of IR Tasks

1.3.1 Navigational IR ⭐

Goal: Find a specific known page/document

Example:

```
Query: "cs 589 stevens"
Expected result: Course website
User knows what they want, just needs to find it
```

Characteristics:

- Clear target in mind
- Success = finding the right page
- Binary outcome (found it or didn't)

1.3.2 Exploratory IR ⭐

Goal: Learn about a topic, discover information

Example from slides:

```
Student: "I need a book on American history for my thesis research!"
Librarian: "What's the topic of your thesis?"
Student: "civil war"
→ Information need: to study about the history of the civil war
```

Characteristics:

- User learning about topic
- May refine query multiple times
- Success = finding useful related information

1.3.3 Conversational Agent IR ⭐

Example: ChatGPT, Claude using RAG (Retrieval Augmented Generation)

How it works:

```
User query → Retrieve relevant context → Generate response with context
```

Example from slides:

```
User: "Find all the phone numbers in a string: 3 digits followed by
      3 digits followed by 4 digits..."
Agent: [retrieves regex patterns from knowledge base]
      [generates Python code with regex]
```

1.3.4 Database/Filter IR

Example: Google Shopping

Features:

- Structured filters (price, brand, availability)
- Faceted search
- Pre-defined categories

1.3.5 Vector Database IR

Modern AI Systems

Example: Finding similar documents using embeddings

```
Query embedding: [0.2, 0.5, -0.3, 0.8, ...]
Document embeddings in vector space
→ Find nearest neighbors using cosine similarity
```

---

2. History of IR

2.1 Timeline of Major Developments

300 BC: Library of Alexandria

- **Callimachus** created the first library catalog

- Manual organization of scrolls
- "Pinakes" = tables/tablets listing authors and works

1880s: Punch Cards

- **Herman Hollerith** invented punch card system
- Used for 1890 US Census
- Searching at **600 cards/minute**
- Mechanical retrieval!

1950s: Early Computer-Based IR

- First attempts to automate information retrieval
- Development of early indexing algorithms
- Transition from manual to computational methods

1958: The Cranfield Experiment ⭐⭐⭐

Revolutionary Finding:

> "Simple keyword-based systems worked as well as complex classification schemes!"

Four Systems Compared:

1. Universal Decimal Classification (hierarchical)
2. Alphabetical Subject Catalogue
3. Faceted Classification Scheme
4. **Uniterm System** (keyword co-ordinate indexing) ← **WINNER!**

Impact: This discovery shaped all modern IR systems!

1960s: Building IR Systems

- Boolean retrieval models
- Development of relevance feedback
- Foundation of modern IR

1970s: Mathematical Models

- **TF-IDF** invented
- **Probability Ranking Principle** established
- Theoretical foundations solidified

1980s: Standardization & Evaluation

- **TREC** (Text Retrieval Conference) established
- Learning to Rank approaches
- Latent Semantic Indexing

1990s-Present: The Web Era

- 1990: Web created
- 1998: **Google** (PageRank algorithm)
- Supporting natural language queries
- Modern evolution:
  - 1997: LSTM
  - 2015: Word2Vec
  - 2017: **Transformers**
  - 2020: GPT-3
  - 2024: RAG systems, Claude, ChatGPT

---

3. The Cranfield Experiment

3.1 The Problem

Question: How should we organize and search documents in a digital library?

3.2 System 1: Boolean Retrieval

Structure:

```
Computer Science
    ├── Artificial Intelligence
    └── Bioinformatics
```

Query Example:

```sql
SELECT * FROM documents
WHERE subject = "AI" AND subject = "bioinformatics"
```

Advantages:

- ✅ Returns exactly what you specify
- ✅ Works well for structured data with attributes

- ✅ Precise control

Disadvantages:

- ❌ Limited to pre-defined categories
- ❌ Doesn't work well for unstructured text
- ❌ Requires exact matches
- ❌ No ranking (all or nothing)

3.3 System 2: Word-Based Indexing (Bag of Words) ⭐⭐⭐

Core Concept: Represent documents as vectors of word counts

Example:

```
Documents:
D1: "the artificial intelligence book"
D2: "the business intelligence"
D3: "the artificial world"

Query: "artificial intelligence book"
```

Step 1: Build Vocabulary

```
Vocabulary = {artificial, business, book, intelligence, the, world}
```

Step 2: Create Document-Term Matrix

Term	D1	D2	D3	Query
artificial	1	0	1	1
business	0	1	0	0
book	1	0	0	1
intelligence	1	1	0	1
the	1	1	1	0
world	0	0	1	0

Vector Representation:

```
D1 = [1, 0, 1, 1, 1, 0]
D2 = [0, 1, 0, 1, 1, 0]
D3 = [1, 0, 0, 0, 1, 1]
Q  = [1, 0, 1, 1, 0, 0]
```

Step 3: Measure Similarity

- Compare query vector to each document vector
- Higher similarity = more relevant

Cranfield Result: This simple system performed as well as complex classification!

---

4. Term Frequency (TF)

4.1 Raw Term Frequency

Definition:

```
tf(t,d) = count of term t in document d
```

Example:

```
Document: "cat cat cat dog dog bird"

tf(cat, d) = 3
tf(dog, d) = 2
tf(bird, d) = 1
```

4.2 The Problem with Raw TF

Issue 1: Scale

- A document with tf=100 shouldn't be 100× more relevant than tf=1
- Diminishing returns for repetition

Issue 2: Spam

- Easy to game: "buy cheap viagra cheap cheap cheap cheap..."
- Keyword stuffing

Issue 3: Document length

- Longer documents naturally have higher counts
- Unfair advantage

4.3 Log-Normalized TF ⭐⭐⭐

Formula:

$$\mathrm{tf}(t,d) = \begin{cases} 1 + \log_{10}(\mathrm{count}(t,d)) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{if } \mathrm{count}(t,d) = 0 \end{cases}$$

Why logarithm?

1. **Diminishing returns**: Each additional occurrence matters less
2. **Compression**: Reduces range of values
3. **Fairness**: tf=10 doesn't dominate tf=1

Visualization:

```
Score

3 |              /---  Log-normalized
  |            /
2 |          /
  |        /
1 |      /          Raw count
  |   /           /
0 |__/___________/_________ Count
    1  2  5  10  20  50
```

4.4 Complete TF Examples ⭐

Example 1: Basic Calculation

```
Document: "machine learning machine learning"

count(machine) = 2
tf(machine, d) = 1 + log₁₀(2)
               = 1 + 0.301
               = 1.301
```

Example 2: Multiple Terms

```
Document: "the cat sat on the mat the"

count(the) = 3 → tf = 1 + log₁₀(3) = 1.477
count(cat) = 1 → tf = 1 + log₁₀(1) = 1.000
count(sat) = 1 → tf = 1.000
count(on)  = 1 → tf = 1.000
count(mat) = 1 → tf = 1.000
```

Example 3: Zero Count

```
Document: "neural network"
Term: "database"

count(database) = 0 → tf = 0
```

Key Values to Remember:

```
count = 1  → tf = 1.000
count = 2  → tf = 1.301
count = 3  → tf = 1.477
count = 5  → tf = 1.699
count = 10 → tf = 2.000
count = 100 → tf = 3.000
```

---

5. Inverse Document Frequency (IDF)

5.1 The Problem: Not All Words Are Equal

Zipf's Law: In natural language, a few words appear very frequently while most words are rare.

Most common English words:

```
the, and, to, of, a, in, that, it, for, is, on, with, you, as, this,
was, but, be, he, at, we, have, by, from, his, are, will...
```

Frequency Distribution (from slides):

```
Rank 1: "the"  = 13,000+ occurrences
Rank 2: "and"  = 6,000+ occurrences
Rank 3: "to"   = 5,000+ occurrences
...
Rank 50: "my"  = 500+ occurrences
```

5.2 Why These Words Are Problematic

Example:

```
Query: "the artificial intelligence book"
D1: "the cat, the dog, the book"
D2: "the business intelligence"
```

Without IDF:

- Both D1 and D2 get high scores because of "the"

- "the" appears in both but carries NO meaning
- Not discriminative!

Intuition:

- **Common words** (appear in many docs) → Less informative
- **Rare words** (appear in few docs) → More discriminative

5.3 IDF Formula ⭐⭐⭐

```
IDF(t) = log₁₀(N / df(t))

where:
N = total number of documents in collection
df(t) = document frequency = # documents containing term t
```

Properties:

- **High IDF**: Term is rare (appears in few documents) → Discriminative
- **Low IDF**: Term is common (appears in many documents) → Not discriminative

5.4 IDF Calculation Examples ⭐⭐

Example 1: Basic IDF

```
Collection: N = 100 documents

Term "the":      appears in 99 docs → df = 99
Term "neural":   appears in 5 docs → df = 5
Term "xylophone": appears in 1 doc → df = 1

IDF(the)       = log₁₀(100/99)  = log₁₀(1.010) = 0.004  ← Very low!
IDF(neural)    = log₁₀(100/5)   = log₁₀(20)    = 1.301
IDF(xylophone) = log₁₀(100/1)   = log₁₀(100)   = 2.000  ← Very high!
```

Interpretation:

- "the" is almost useless (appears everywhere)
- "neural" is moderately discriminative
- "xylophone" is highly discriminative

Example 2: Complete Calculation

```
Documents (N=3):
D1: "the cat sat on the mat"
D2: "the dog sat on the log"
D3: "cats and dogs play"

Query: "the cat dog"

Calculate IDF for each query term:

df(the) = 2  (appears in D1, D2)
df(cat) = 1  (appears in D1 only)
df(dog) = 2  (appears in D2, D3)

IDF(the) = log₁₀(3/2) = log₁₀(1.5) = 0.176
IDF(cat) = log₁₀(3/1) = log₁₀(3)   = 0.477  ← Most discriminative!
IDF(dog) = log₁₀(3/2) = log₁₀(1.5) = 0.176
```

Key Insight: "cat" is more discriminative than "the" or "dog"!

5.5 IDF Properties

Property 1: Monotonicity

```
More documents containing term → Lower IDF
Fewer documents containing term → Higher IDF
```

Property 2: Logarithmic Scale

```
Like TF, uses log to compress range
Prevents rare terms from dominating
```

Property 3: Collection-Dependent

```
IDF values change with the collection
"neural" has different IDF in:
- Medical journal collection (common → low IDF)
- General news collection (rare → high IDF)
```

---

6. TF-IDF Weighting

6.1 Combining TF and IDF ⭐⭐⭐

Complete Formula:

```
w(t,d) = tf(t,d) × IDF(t)
```

Interpretation:

- **High TF-IDF**: Term is frequent in this document but rare across collection
  - Example: "quantum" appears 10 times in physics paper, rare overall
- **Low TF-IDF**: Term is either:
  - Rare in document, OR
  - Common in collection

The "Sweet Spot":

```
Best terms are:
- High frequency in document (high TF)
- Low frequency in collection (high IDF)
```

6.2 Complete TF-IDF Example ⭐⭐⭐

Setup:

```
Documents (N=3):
D1: "the cat sat on the mat"     (6 words)
D2: "the dog sat on the log"     (6 words)
D3: "cats and dogs play"         (4 words)

Query: "the cat dog"
```

Step 1: Calculate TF (Log-normalized)

For D1:

```
count(the) = 2 → tf(the,D1) = 1 + log₁₀(2) = 1.301
count(cat) = 1 → tf(cat,D1) = 1 + log₁₀(1) = 1.000
count(dog) = 0 → tf(dog,D1) = 0
```

For D2:

```
count(the) = 2 → tf(the,D2) = 1.301
count(cat) = 0 → tf(cat,D2) = 0
count(dog) = 1 → tf(dog,D2) = 1.000
```

For D3:

```
count(the) = 0 → tf(the,D3) = 0
count(cat) = 0 → tf(cat,D3) = 0  (note: "cats" ≠ "cat")
count(dog) = 0 → tf(dog,D3) = 0  (note: "dogs" ≠ "dog")
```

Step 2: Calculate IDF

```
df(the) = 2  (appears in D1, D2)
df(cat) = 1  (appears in D1 only)
df(dog) = 1  (appears in D2 only)

IDF(the) = log₁₀(3/2) = 0.176
IDF(cat) = log₁₀(3/1) = 0.477
IDF(dog) = log₁₀(3/1) = 0.477
```

Step 3: Calculate TF-IDF Weights

For D1:

```
w(the,D1) = 1.301 × 0.176 = 0.229
w(cat,D1) = 1.000 × 0.477 = 0.477
w(dog,D1) = 0     × 0.477 = 0.000
```

For D2:

```
w(the,D2) = 1.301 × 0.176 = 0.229
w(cat,D2) = 0     × 0.477 = 0.000
w(dog,D2) = 1.000 × 0.477 = 0.477
```

For D3:

```
w(the,D3) = 0 × 0.176 = 0.000
w(cat,D3) = 0 × 0.477 = 0.000
w(dog,D3) = 0 × 0.477 = 0.000
```

Step 4: Create TF-IDF Vectors

```
        [the,  cat,  dog]
D1_tfidf = [0.229, 0.477, 0.000]
D2_tfidf = [0.229, 0.000, 0.477]
D3_tfidf = [0.000, 0.000, 0.000]
Q_tfidf  = [0.176, 0.477, 0.477]  (using IDF as query weights)
```

Observations:

- D1 and D2 have similar "the" weights (0.229)

- D1 has high "cat" weight, D2 has high "dog" weight
- D3 has all zeros (no matching terms!)

---

7. Vector Space Model

7.1 The Geometric Interpretation ⭐⭐⭐

Core Idea: Represent documents and queries as vectors in high-dimensional space

Visualization (2D example):

```
intelligence ↑
             |
    D2 ●     |
        \    |
         \   |
          \  |
Q ●───────────●──────────→ artificial
           D1
```

Properties:

- Each dimension = one unique term in vocabulary
- Document vector = TF-IDF weights for all terms
- Query vector = TF-IDF weights for query terms

7.2 Cosine Similarity Formula ⭐⭐⭐

Formula:

$$\text{cos\_sim}(d,q) = (d \cdot q) / (||d|| \times ||q||)$$

where:

$$d \cdot q = \text{dot product} = \Sigma_i (d_i \times q_i)$$
$$||d|| = \text{Euclidean norm} = \sqrt{(\Sigma_i d_i^2)}$$
$$||q|| = \text{Euclidean norm} = \sqrt{(\Sigma_i q_i^2)}$$

Geometric Meaning:

- Measures the **cosine of angle** between vectors
- Range: [0, 1] for non-negative weights
  - 1 = vectors point in same direction (most similar)
  - 0 = vectors are orthogonal (no similarity)

Why Cosine and not Euclidean Distance?

1. **Length invariant**: Normalizes for document length
2. **Direction matters**: Focuses on term distribution, not magnitude
3. **Works in high dimensions**: Stable in sparse spaces

7.3 Complete Cosine Similarity Example ⭐⭐⭐

Using our previous example:

```
Query q = "the artificial intelligence book"

D1 = "the cat, the dog, the book"
D2 = "the business intelligence"
D3 = "the artificial world"
```

Build Term-Document Matrix (using raw TF first):

Term	D1	D2	D3	Query
intelligence	0	1	0	1
book	1	0	0	1
the	3	1	1	1
cat	1	0	0	0
artificial	0	0	1	1
dog	1	0	0	0
business	0	1	0	0
world	0	0	1	0

Vectors (term order: intelligence, book, the, cat, artificial, dog, business, world):

```
q  = [1, 1, 1, 0, 1, 0, 0, 0]
D1 = [0, 1, 3, 1, 0, 1, 0, 0]
D2 = [1, 0, 1, 0, 0, 0, 1, 0]
D3 = [0, 0, 1, 0, 1, 0, 0, 1]
```

Step 1: Calculate Dot Products

$$
\begin{aligned}
q \cdot D1 &= (1\times0) + (1\times1) + (1\times3) + (0\times1) + (1\times0) + (0\times1) + (0\times0) + (0\times0) \\
&= 0 + 1 + 3 + 0 + 0 + 0 + 0 + 0 \\
&= 4
\end{aligned}
$$

$$
\begin{aligned}
q \cdot D2 &= (1\times1) + (1\times0) + (1\times1) + (0\times0) + (1\times0) + (0\times0) + (0\times1) + (0\times0) \\
&= 1 + 0 + 1 + 0 + 0 + 0 + 0 + 0 \\
&= 2
\end{aligned}
$$

$$
\begin{aligned}
q \cdot D3 &= (1\times0) + (1\times0) + (1\times1) + (0\times0) + (1\times1) + (0\times0) + (0\times0) + (0\times1) \\
&= 0 + 0 + 1 + 0 + 1 + 0 + 0 + 0 \\
&= 2
\end{aligned}
$$

Step 2: Calculate Norms

$$
\begin{aligned}
||q|| &= \sqrt{(1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2)} \\
&= \sqrt{(1 + 1 + 1 + 0 + 1 + 0 + 0 + 0)} \\
&= \sqrt{4} \\
&= 2.000
\end{aligned}
$$

$$
\begin{aligned}
||D1|| &= \sqrt{(0^2 + 1^2 + 3^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2)} \\
&= \sqrt{(0 + 1 + 9 + 1 + 0 + 1 + 0 + 0)} \\
&= \sqrt{12} \\
&= 3.464
\end{aligned}
$$

$$
\begin{aligned}
||D2|| &= \sqrt{(1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2)} \\
&= \sqrt{(1 + 0 + 1 + 0 + 0 + 0 + 1 + 0)} \\
&= \sqrt{3} \\
&= 1.732
\end{aligned}
$$

$$
\begin{aligned}
||D3|| &= \sqrt{(0^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 0^2 + 1^2)} \\
&= \sqrt{(0 + 0 + 1 + 0 + 1 + 0 + 0 + 1)} \\
&= \sqrt{3} \\
&= 1.732
\end{aligned}
$$

Step 3: Calculate Cosine Similarities

```
cos_sim(q,D1) = 4 / (2.000 × 3.464)
              = 4 / 6.928
              = 0.5773

cos_sim(q,D2) = 2 / (2.000 × 1.732)
              = 2 / 3.464
              = 0.5773

cos_sim(q,D3) = 2 / (2.000 × 1.732)
              = 2 / 3.464
              = 0.5773
```

Problem: All documents get the SAME score! 🚨

Why? The term "the" dominates all documents, masking the important terms.

Solution: Use TF-IDF weights instead of raw TF!

7.4 Cosine Similarity with TF-IDF Weights ⭐⭐⭐

Using TF-IDF vectors from Section 6.2:

```
         [the,   cat,   dog]
D1_tfidf = [0.229, 0.477, 0.000]
D2_tfidf = [0.229, 0.000, 0.477]
D3_tfidf = [0.000, 0.000, 0.000]
Q_tfidf  = [0.176, 0.477, 0.477]
```

Calculate with TF-IDF:

```
q · D1 = (0.176×0.229) + (0.477×0.477) + (0.477×0.000)
       = 0.040 + 0.227 + 0.000
       = 0.267

q · D2 = (0.176×0.229) + (0.477×0.000) + (0.477×0.477)
       = 0.040 + 0.000 + 0.227
       = 0.267

||q|| = √(0.176² + 0.477² + 0.477²) = √0.486 = 0.697
||D1|| = √(0.229² + 0.477² + 0.000²) = √0.280 = 0.529
||D2|| = √(0.229² + 0.000² + 0.477²) = √0.280 = 0.529

cos_sim(q,D1) = 0.267 / (0.697 × 0.529) = 0.267 / 0.369 = 0.724
cos_sim(q,D2) = 0.267 / (0.697 × 0.529) = 0.267 / 0.369 = 0.724
```

Still the same! Let's understand why with the full 8-dimensional space...

With all 8 dimensions and proper IDF weighting:

```
After applying IDF to downweight "the":
score(q, D1) → 0.3582
score(q, D2) → 0.4220  ← Higher!
```

D2 ranks higher because it contains "intelligence" which is more discriminative than D1's terms.

---

8. Document Length Pivoting

8.1 The Problem ⭐ ⭐

Issue: Cosine similarity **over-penalizes** long documents

Example from slides:

```
Query: "artificial intelligence"

D1 (21 words): "Artificial intelligence was founded as an academic
               discipline in 1955, and in the years since has
               experienced several waves of optimism"

D2 (4 words): "the journal of Artificial intelligence"

Vector magnitudes:
D1: ||d1|| = √21 = 4.583
D2: ||d2|| = √4 = 2.000

Effect on cosine similarity:
D2 gets higher weight just because it's shorter!
```

Problem:

- D1 might be much more comprehensive and relevant
- But gets penalized for being longer
- This is unfair!

8.2 The Intuition

Two Competing Effects:

1. **Verbosity Hypothesis** (favor short docs):
    - Longer documents contain more topics
    - Query may match only a small subset of content
    - More "noise" dilutes signal

2. **Scope Hypothesis** (favor long docs):
   - Longer documents cover topics more thoroughly
   - More content = more chance to be relevant
   - More comprehensive = better for user

Reality: Neither extreme is optimal!

- No normalization (norm=1): Long docs favored too much
- Full cosine normalization (norm=|d|): Short docs favored too much
- **Need something in between**

8.3 Singhal's Pivoted Document Length Normalization ⭐⭐⭐

Paper: Singhal, Buckley, Mitra (1995)

Experimental Setup:

1. For many queries and documents, calculate TF-IDF scores
2. Manually evaluate true relevance
3. Plot: document length vs retrieval score
4. Plot: document length vs true relevance
5. Find bias!

The Discovery (from slide 56):

```
Relevance ↑
      |
      |  ╱ True relevance (slope near 0)
      | ╱
    p | ╱────────
      |╱
      |╱
      |  ╱ TF-IDF score (positive slope)
      | ╱
      | ╱
      |╱
      |╱_____________→ Document length
         ℓp
```

Observations:

- TF-IDF score increases with document length (bias!)
- True relevance is roughly constant with length

- Need to correct this bias

8.4 The Normalization Formula ⭐⭐⭐

Pivoted Normalization:

```
normalization = 1 - b + b × (dl/avgdl)

where:
dl = length of document d (number of words)
avgdl = average document length in collection
b = tuning parameter (typically 0.75)
```

Behavior:

```
When dl = avgdl:  norm = 1 - b + b×1 = 1      (no change)
When dl > avgdl:  norm > 1                   (penalize long docs)
When dl < avgdl:  norm < 1                   (boost short docs)
```

Visual Representation (from slide 59):

```
normalization ↑
              |   ╱ y(|d|) = (1-b) + b × |d|/avgdl
              |  ╱
              | ╱
            1 |─┼──────────────────────
              |╱
              |╱
          1-b |────────────────────────
              └──────────────────────────── dl →
                  avgdl
```

Key Properties:

1. **Linear in document length**: y = mx + c form
2. **Crosses 1 at avgdl**: Average-length docs get no adjustment
3. **Slope controlled by b**:
   - b=0: Flat line at y=1 (no normalization)
   - b=1: Line from 0 to 2×(dl/avgdl) (full normalization)
   - b=0.75: Standard compromise

8.5 Parameter b: The Tuning Knob ⭐

Effect of parameter b:

```
b = 0:    norm = 1                  (no length normalization)
b = 0.5:  norm = 0.5 + 0.5×(dl/avgdl)  (moderate normalization)
b = 0.75: norm = 0.25 + 0.75×(dl/avgdl) (standard BM25 value)
b = 1:    norm = dl/avgdl           (full normalization)
```

Choosing b:

- Small b (≈0): When long docs should be favored
  - Example: Looking for comprehensive coverage
- Large b (≈1): When verbosity is a problem
  - Example: Looking for focused, specific content
- b=0.75: Good general-purpose compromise

8.6 Complete Example: Document Length Pivoting ⭐⭐⭐

Scenario:

```
Collection statistics:
- Total documents: 1000
- Average document length: avgdl = 500 words
- Parameter: b = 0.75

Documents:
D1: 200 words  (short)
D2: 500 words  (average)
D3: 1000 words (long)

Suppose each gets raw score = 10 (before normalization)
```

Calculate normalized scores:

For D1 (short document):

```
norm_D1 = 1 - 0.75 + 0.75 × (200/500)
        = 0.25 + 0.75 × 0.4
        = 0.25 + 0.30
        = 0.55

normalized_score_D1 = 10 / 0.55 = 18.18  ← Boosted!
```

For D2 (average document):

```
norm_D2 = 1 - 0.75 + 0.75 × (500/500)
       = 0.25 + 0.75 × 1.0
       = 0.25 + 0.75
       = 1.00

normalized_score_D2 = 10 / 1.00 = 10.00  ← Unchanged
```

For D3 (long document):

```
norm_D3 = 1 - 0.75 + 0.75 × (1000/500)
       = 0.25 + 0.75 × 2.0
       = 0.25 + 1.50
       = 1.75

normalized_score_D3 = 10 / 1.75 = 5.71   ← Penalized!
```

Results:

```
D1 (short):   18.18  [+82% boost]
D2 (average): 10.00  [no change]
D3 (long):     5.71  [-43% penalty]
```

Interpretation:

- Short doc gets significant boost (compensates for having fewer words)
- Average doc unchanged (reference point)
- Long doc penalized (prevents verbosity from dominating)

8.7 Complete Scoring Formula with Pivoting ⭐⭐⭐

Final TF-IDF with Document Length Pivoting:

```
score(q,d) = Σ IDF(w) × c(w,d) / normalization
          w∈q

where:
c(w,d) = tf(w,d) = 1 + log₁₀(count(w,d))
IDF(w) = log₁₀(N/df(w))
normalization = 1 - b + b×(|d|/avgdl)
```

This satisfies the three desiderata:

1. ✅ **Document length pivoting**: Proper normalization between 1 and |d|
2. ✅ **IDF**: Discounts common words

3. ✅ **TF saturation**: Log scale provides diminishing returns

---

9. Complete BM25 Preview

BM25 (Best Match 25) is the most widely used retrieval model!

Preview Formula (full derivation in Lecture 2):

```
BM25(d,q) = Σ IDF(tᵢ) × [tf(tᵢ,d)×(k₁+1)] / [tf(tᵢ,d) + k₁×(1-b+b×dl/avgdl)]
         tᵢ∈q

where:
IDF(t) = log[(N - df(t) + 0.5)/(df(t) + 0.5)]  [with smoothing]
k₁ = TF saturation parameter (typical: 1.2)
b = length normalization parameter (typical: 0.75)
```

Key Components:

1. **IDF term**: From RSJ probabilistic model
2. **TF saturation**: From 2-Poisson model (prevents spam)
3. **Length pivoting**: Same as we learned!

Why BM25 > TF-IDF:

- Theoretically grounded (probabilistic model)
- Better TF saturation (handles repetition better)
- Tunable parameters (k_1, b)
- State-of-the-art performance

---

10. Practice Problems

Problem 10.1: TF Calculation ⭐

Calculate log-normalized TF:

Document: "data mining data analysis big data"

Questions:

1. tf(data, d) = ?
2. tf(mining, d) = ?
3. tf(analysis, d) = ?
4. tf(big, d) = ?

5. tf(machine, d) = ?

<details> <summary>Click for Solution</summary>

Solution:

1. count(data) = 3

```
tf(data, d) = 1 + log₁₀(3) = 1 + 0.477 = 1.477
```

2. count(mining) = 1

```
tf(mining, d) = 1 + log₁₀(1) = 1 + 0 = 1.000
```

3. count(analysis) = 1

```
tf(analysis, d) = 1.000
```

4. count(big) = 1

```
tf(big, d) = 1.000
```

5. count(machine) = 0

```
tf(machine, d) = 0
```

TF vector: [1.477, 1.000, 1.000, 1.000, 0]

</details>

Problem 10.2: IDF Calculation ⭐ ⭐

Given:

- Collection: N = 1000 documents
- Term "machine": appears in 100 documents
- Term "learning": appears in 200 documents
- Term "xylophone": appears in 2 documents
- Term "the": appears in 998 documents

Calculate IDF for each term.

<details> <summary>Click for Solution</summary>

Solution:

```
IDF(machine) = log₁₀(1000/100)
             = log₁₀(10)
             = 1.000

IDF(learning) = log₁₀(1000/200)
              = log₁₀(5)
              = 0.699

IDF(xylophone) = log₁₀(1000/2)
               = log₁₀(500)
               = 2.699  ← Very discriminative!

IDF(the) = log₁₀(1000/998)
         = log₁₀(1.002)
         = 0.001  ← Nearly useless!
```

Ranking by discriminative power: xylophone > machine > learning > the

</details>

Problem 10.3: TF-IDF Vectors ⭐⭐⭐

Given:

```
Collection: N = 4 documents

D1: "machine learning is fun"
D2: "deep learning is powerful"
D3: "machine translation works"
D4: "learning is important"

Query: "machine learning"
```

Questions:

1. Calculate df for each term
2. Calculate IDF for query terms
3. Calculate TF for D1
4. Calculate TF-IDF vector for D1
5. Which document is most relevant?

<details> <summary>Click for Solution</summary>

Solution:

Step 1: Document frequencies

```
df(machine) = 2  (D1, D3)
df(learning) = 3  (D1, D2, D4)
df(deep) = 1
df(is) = 3
df(fun) = 1
df(powerful) = 1
df(translation) = 1
df(works) = 1
df(important) = 1
```

Step 2: IDF for query terms

```
IDF(machine) = log₁₀(4/2) = log₁₀(2) = 0.301
IDF(learning) = log₁₀(4/3) = log₁₀(1.333) = 0.125
```

Step 3: TF for D1 = "machine learning is fun"

```
All counts = 1
tf(machine, D1) = 1 + log₁₀(1) = 1.000
tf(learning, D1) = 1.000
tf(is, D1) = 1.000
tf(fun, D1) = 1.000
```

Step 4: TF-IDF for D1 (query terms only)

```
w(machine, D1) = 1.000 × 0.301 = 0.301
w(learning, D1) = 1.000 × 0.125 = 0.125

D1_tfidf (for query terms) = [0.301, 0.125]
```

Step 5: Calculate for all documents

D1: machine=0.301, learning=0.125 → Total: 0.426

D2: machine=0, learning=0.125 → Total: 0.125

D3: machine=0.301, learning=0 → Total: 0.301

D4: machine=0, learning=0.125 → Total: 0.125

Most relevant: D1 (contains both query terms!)

</details>

Problem 10.4: Cosine Similarity ⭐⭐⭐

Given vectors:

```
d = [3, 4, 0, 1]
q = [1, 1, 2, 0]
```

Calculate cosine similarity.

<details> <summary>Click for Solution</summary>

Solution:

Step 1: Dot product

```
d · q = (3×1) + (4×1) + (0×2) + (1×0)
      = 3 + 4 + 0 + 0
      = 7
```

Step 2: Norms

```
||d|| = √(3² + 4² + 0² + 1²)
      = √(9 + 16 + 0 + 1)
      = √26
      = 5.099

||q|| = √(1² + 1² + 2² + 0²)
      = √(1 + 1 + 4 + 0)
      = √6
      = 2.449
```

Step 3: Cosine similarity

```
cos_sim(d,q) = 7 / (5.099 × 2.449)
             = 7 / 12.487
             = 0.561
```

Interpretation: Moderate similarity (56%)

</details>

Problem 10.5: Document Length Pivoting ⭐⭐⭐

Given:

```
Collection: avgdl = 500 words
Parameter: b = 0.75

Documents with raw scores:
D1: 250 words, raw_score = 8
D2: 500 words, raw_score = 8
D3: 1000 words, raw_score = 8
```

Calculate normalized scores and rank documents.

<details> <summary>Click for Solution</summary>

Solution:

For D1 (short):

```
norm = 1 - 0.75 + 0.75×(250/500)
     = 0.25 + 0.75×0.5
     = 0.25 + 0.375
     = 0.625

normalized_score = 8 / 0.625 = 12.8
```

For D2 (average):

```
norm = 1 - 0.75 + 0.75×(500/500)
     = 0.25 + 0.75×1.0
     = 1.0

normalized_score = 8 / 1.0 = 8.0
```

For D3 (long):

```
norm = 1 - 0.75 + 0.75×(1000/500)
     = 0.25 + 0.75×2.0
     = 0.25 + 1.5
     = 1.75

normalized_score = 8 / 1.75 = 4.57
```

Final Ranking:

1. D1: 12.8 (short doc boosted)
2. D2: 8.0 (average unchanged)
3. D3: 4.57 (long doc penalized)

Interpretation: Short document wins despite all having same raw score!

</details>

Problem 10.6: Complete TF-IDF Ranking ⭐ ⭐ ⭐

Given:

```
Collection: N = 100, avgdl = 10 words

D1: "neural network deep learning neural" (5 words)
D2: "neural network architecture" (3 words)

Query: "neural network"

Document frequencies:
df(neural) = 30
df(network) = 40
df(deep) = 10
df(learning) = 20
df(architecture) = 5
```

Calculate complete TF-IDF scores with pivoting (b=0.75).

<details> <summary>Click for Solution</summary>

Solution:

Step 1: Calculate IDF

```
IDF(neural) = log₁₀(100/30) = log₁₀(3.333) = 0.523
IDF(network) = log₁₀(100/40) = log₁₀(2.5) = 0.398
```

Step 2: Calculate TF for D1

```
count(neural, D1) = 2 → tf = 1 + log₁₀(2) = 1.301
count(network, D1) = 1 → tf = 1 + log₁₀(1) = 1.000
```

Step 3: Calculate TF-IDF for D1

```
w(neural, D1) = 1.301 × 0.523 = 0.680
w(network, D1) = 1.000 × 0.398 = 0.398

raw_score_D1 = 0.680 + 0.398 = 1.078
```

Step 4: Calculate TF for D2

```
count(neural, D2) = 1 → tf = 1.000
count(network, D2) = 1 → tf = 1.000
```

Step 5: Calculate TF-IDF for D2

```
w(neural, D2) = 1.000 × 0.523 = 0.523
w(network, D2) = 1.000 × 0.398 = 0.398

raw_score_D2 = 0.523 + 0.398 = 0.921
```

Step 6: Apply document length pivoting

For D1 (dl=5):

```
norm_D1 = 1 - 0.75 + 0.75×(5/10) = 0.25 + 0.375 = 0.625
final_score_D1 = 1.078 / 0.625 = 1.725
```

For D2 (dl=3):

```
norm_D2 = 1 - 0.75 + 0.75×(3/10) = 0.25 + 0.225 = 0.475
final_score_D2 = 0.921 / 0.475 = 1.939
```

Final Ranking:

1. **D2: 1.939** ← Winner!
2. D1: 1.725

Interpretation:

- D2 wins despite lower raw TF-IDF score
- Shorter length gives it advantage
- Both docs contain query terms once, but D2 is more focused

</details>

---

11. Key Formulas Summary

TF (Term Frequency)

```
        ⎧ 1 + log₁₀(count(t,d))   if count > 0
tf(t,d) = ⎨
        ⎩ 0                       if count = 0
```

IDF (Inverse Document Frequency)

```
IDF(t) = log₁₀(N / df(t))

where:
N = total number of documents
df(t) = number of documents containing term t
```

TF-IDF Weight

```
w(t,d) = tf(t,d) × IDF(t)
```

Cosine Similarity

```
cos_sim(d,q) = (d · q) / (||d|| × ||q||)

where:
d · q = Σᵢ (dᵢ × qᵢ)        [dot product]
||d|| = √(Σᵢ dᵢ²)          [Euclidean norm]
||q|| = √(Σᵢ qᵢ²)
```

Document Length Pivoting

```
normalization = 1 - b + b × (dl/avgdl)

where:
dl = document length
avgdl = average document length in collection
b = tuning parameter (typically 0.75)
```

Complete TF-IDF Score with Pivoting

```
score(q,d) = Σ IDF(w) × tf(w,d) / [1 - b + b×(|d|/avgdl)]
          w∈q
```

📌 Key Concepts to Memorize

Must Know Cold ⭐⭐⭐

1. **TF-IDF formula** and how to calculate
2. **Cosine similarity** formula and geometric meaning
3. **Document length pivoting** formula and why we need it

4. **IDF** formula and interpretation
5. **Log-normalized TF** formula

Conceptual Understanding ⭐⭐

1. **Why log normalization?** → Diminishing returns for term repetition
2. **Why IDF?** → Common words are less discriminative
3. **Why document length pivoting?** → Fair comparison between docs of different lengths
4. **Cosine vs Euclidean distance?** → Cosine is length-invariant
5. **Cranfield experiment result** → Simple keyword indexing works!

Important Values to Remember ⭐

```
b = 0.75  (standard document length pivoting parameter)
k₁ = 1.2  (BM25 TF saturation parameter, preview for Lecture 2)

Common TF values:
count = 1  → tf = 1.000
count = 2  → tf = 1.301
count = 10 → tf = 2.000

Log properties:
log₁₀(1) = 0
log₁₀(10) = 1
log₁₀(100) = 2
```

🎯 Exam Tips for Lecture 1

What You'll Be Asked to Calculate

1. ✅ TF-IDF weights for given documents and query
2. ✅ Cosine similarity between vectors
3. ✅ Document length normalization
4. ✅ Complete retrieval scores
5. ✅ Ranking of documents

Common Mistakes to Avoid

- ❌ Forgetting the +1 in TF formula: tf = 1 + log(count)
- ❌ Using log(df/N) instead of log(N/df) for IDF
- ❌ Not squaring terms when calculating norm: $||d|| = \sqrt{\Sigma d_i^2}$

- ❌ Dividing by wrong normalization in document pivoting
- ❌ Forgetting log is base 10 ($\log_{10}$) not natural log (ln)

Calculation Tips

1. **Show your work** - partial credit matters!
2. **Check units** - does the answer make sense?
3. **Use calculator** - log values need precision
4. **Verify range** - cosine similarity should be [0,1]
5. **Label clearly** - indicate what each step calculates

Time Management

- Simple TF calculation: ~2 minutes
- IDF calculation: ~2 minutes
- Cosine similarity: ~5 minutes
- Complete TF-IDF ranking: ~10 minutes
- Document length pivoting: ~3 minutes

📚 Additional Resources

Video Tutorials

- **TF-IDF Explained**: StatQuest on YouTube
- **Vector Space Model**: Stanford CS276 lectures
- **Cosine Similarity**: 3Blue1Brown linear algebra series

Papers to Read

- Singhal et al. (1995): "Pivoted Document Length Normalization"
- Salton & Buckley (1988): "Term-weighting approaches in automatic text retrieval"

Online Tools

- TF-IDF Calculator: https://www.tfidf.com/
- Vector Similarity Calculator: Various online tools
- Elasticsearch Documentation: Real-world implementation

Practice More

- Stanford IR Book: https://nlp.stanford.edu/IR-book/

- TREC datasets: Standard evaluation collections
- Kaggle text mining competitions

✅ Self-Assessment Checklist

Before the exam, make sure you can:

- [ ] Calculate log-normalized TF for any term in any document
- [ ] Calculate IDF given document frequencies
- [ ] Compute TF-IDF weights for a term-document pair
- [ ] Build a TF-IDF vector for a document
- [ ] Calculate dot product of two vectors
- [ ] Calculate Euclidean norm of a vector
- [ ] Compute cosine similarity between two vectors
- [ ] Apply document length pivoting formula
- [ ] Explain why we need IDF (not just the formula)
- [ ] Explain why we need document length pivoting
- [ ] Explain the Cranfield experiment result
- [ ] Compare Boolean retrieval vs keyword-based retrieval
- [ ] Rank multiple documents given a query

If you can do all of the above, you're ready for Lecture 1 content! ✨

Good luck with your studies! 🎓

Remember: Understanding the WHY is more important than memorizing formulas. Focus on the intuition behind each technique!

Study Guide Created: Fall 2024 Based on: CS 589 Lecture 1 slides and materials Instructor: Professor Susan Liu