Natural Language Processing

Term Weighting: Incidence, TF-IDF and Similarity

Ramaseshan Ramachandran

From Text to a Matrix

WHERE THIS SESSION SITS

Last session counted words and found that the counts are wildly unequal. This session turns those counts into something a machine can compare.

- **Incidence matrix.** Does the term appear in the document at all? A matrix of bits
- **Term frequency.** How often does it appear? A matrix of counts
- **Inverse document frequency.** How informative is the term? A weight per row
- **Cosine similarity.** How close are two documents? An angle

---

Two ideas leave this session and never depart. **Weight terms by how much they discriminate. Measure similarity as an angle, not a distance.**

INCIDENCE MATRIX

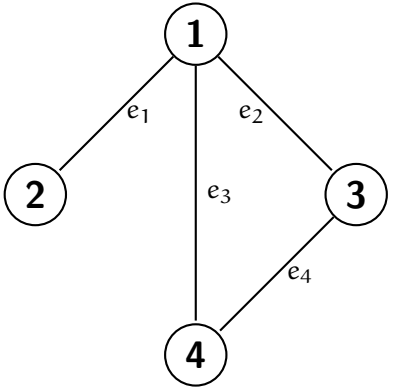
The term comes from graph theory. Let G be a graph with n vertices $(v_1, v_2, \ldots, v_n)$ and m edges $(e_1, e_2, \ldots, e_m)$. The *incidence matrix* of size $n \times m$ is defined by

$$x_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is an endpoint of edge } e_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

It is also called the vertex–edge incidence matrix and is written $X(G)$.

The row for a vertex records everything that vertex touches. That is the pattern we want. In a moment the rows will be terms and the columns will be documents.

BINARY INCIDENCE MATRIX



The incidence matrix corresponding to the left is given below

	e_1	e_2	e_3	e_4
1	1	1	1	0
2	1	0	0	0
3	0	1	0	1
4	0	0	1	1

$$x_{ij} = \begin{cases} 1 & \text{if edge } e_j \text{ is incident on vertex } v_i \\ 0 & \text{otherwise} \end{cases}$$

TERM–DOCUMENT BINARY INCIDENCE MATRIX

Now let the rows be terms and the columns be documents. Shakespeare's plays are the classic example [1]

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello
antony	1	1	0	0	0
brutus	1	1	0	1	0
caesar	1	1	0	1	1
calpurnia	0	1	0	0	0
cleopatra	1	0	0	0	0
...	...	...	...	...	...
...	...	...	...	...	...

$$x_{td} = \begin{cases} 1, & \text{if the term } t \text{ occurs in document } d \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

This matrix carries no information about word order and none about how often a word occurs.

IR USING A BINARY INCIDENCE MATRIX

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello
antony	1	1	0	0	0
brutus	1	1	0	1	0
caesar	1	1	0	1	1
calpurnia	0	1	0	0	0
cleopatra	1	0	0	0	0
...	...	...	...	...	...
...	...	...	...	...	...

To answer the query *Brutus AND Caesar AND NOT Calpurnia*, take the rows for *brutus* and *caesar*, complement the row for *calpurnia*, and do a bitwise AND: 11010 AND 11011 AND 10111 = 10010. The answer is *Antony and Cleopatra* and *Hamlet*.

A query is now a Boolean expression evaluated on bit vectors. That is the entire retrieval model, and it was the state of the art for thirty years.

WHY THE MATRIX CANNOT BE STORED THE OBVIOUS WAY

The matrix has $|V|$ rows and N columns, so it holds $|V| \times \mathrm{N}$ bits. Take a modest collection:

Documents N	1,000,000
Distinct terms $\lvert V\rvert$	500,000
Cells $= \lvert V\rvert \times \mathrm{N}$	5×10^{11} bits
Stored as a dense bit matrix	62.5 GB
Distinct terms in an average document	$\approx 1{,}000$
Non-zero cells $\approx \mathrm{N} \times 1{,}000$	10^9
Density	0.2%

- 99.8% of the matrix is zero, and that is a consequence of Zipf's law. Most terms are rare, so most rows are nearly empty
- So we never store the matrix. We store, for each term, the list of documents that contain it. That is the **inverted index**
- The same sparsity argument returns for co-occurrence matrices in session 7 and for embedding tables later

WHAT THE BIT MATRIX CANNOT DO

- A document that mentions *caesar* once and one that mentions it forty times look identical
- Every matching document is equally good, so there is **no ranking**. A query returning 10,000 documents returns them in no order
- Boolean queries are precise and unforgiving. AND returns too little, OR returns too much
- Users do not write Boolean expressions. They write *climate policy 2026*

Replace the bit with a count, and ranking becomes possible. That is the next step.

The atomic unit is a ***word***

- Words are built from the alphabet of a language; we treat the ***word*** (or ***term***, which may span co-occurring words) as the atomic unit
- For computation, every word needs a numerical representation
- The vocabulary $V = \{w_1, w_2, w_3, \ldots, w_n\}$ is the set of the n unique words of a corpus
- A word of V may appear in a document of the collection $D = \{D_1, D_2, D_3, \ldots, D_m\}$ once, several times, or not at all

---

Write $tf_{t,d}$ for the number of times term t occurs in document d, and M_d for the total number of tokens in d. Those two symbols carry the rest of this session.

TERM-FREQUENCY WEIGHTING SCHEMES

$$\text{Boolean} \quad 0 \text{ or } 1 \tag{3}$$

$$\text{Raw count} \quad \mathrm{tf}_{t,d} \tag{4}$$

$$\text{Adjusted to document length} \quad \frac{\mathrm{tf}_{t,d}}{M_d} \tag{5}$$

$$\text{Log weighting} \quad \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

- **Boolean** throws away the count. This is the incidence matrix
- **Raw count** keeps everything, including the fact that long documents have larger counts for no interesting reason
- **Length adjustment** turns the count into a rate, so a 200-word abstract and a 20,000-word thesis become comparable
- **Log weighting** keeps the ordering but flattens the scale. The $+1$ makes a single occurrence score 1 rather than 0

WHY DAMP THE COUNT AT ALL?

Relevance does not grow linearly with occurrences. A document that says *cricket* twenty times is more about cricket than one that says it twice. It is not ten times more about cricket.

$\mathrm{tf}_{t,d}$	raw	tf/M_d ($\mathrm{M}_d = 1000$)	$1 + \log_{10} \mathrm{tf}$
1	1	0.001	1.000
2	2	0.002	1.301
10	10	0.010	2.000
100	100	0.100	3.000
1000	1000	1.000	4.000

- A thousandfold spread in raw counts becomes a fourfold spread in weight
- The logarithm turns *multiplicative* differences into *additive* ones, which is the same trick that made Zipf's law a straight line
- Length adjustment and log damping fix different problems and are often used together

DISADVANTAGES OF RAW FREQUENCY

- All terms are treated as equally important
- A common term such as ***the*** carries no information about the document, yet receives the highest score in almost every document
- Zipf's law guarantees this. The largest counts in any document belong to the least informative words
- Classification and retrieval both suffer when the score is dominated by words shared across the whole collection

---

Term frequency measures importance within a document. Nothing so far measures importance across the collection.

INVERSE DOCUMENT FREQUENCY

We want to scale down the weight of terms that occur everywhere and boost the weight of terms that occur in few places.
Let df_t be the **document frequency**, the number of documents that contain t at least once, out of N documents in the collection. Then

$$\mathrm{idf}_t = \log\left(\frac{N}{\mathrm{df}_t}\right) \tag{7}$$

- df_t counts *documents*, not occurrences. A term used 500 times in one document still has $\mathrm{df}_t = 1$
- $\mathrm{df}_t \leq N$ always, so the ratio is at least 1 and the idf is never negative
- If a term appears in every document, $\mathrm{df}_t = N$, so $\mathrm{idf}_t = \log 1 = 0$. The term cannot discriminate, and its weight is annihilated
- The rarer the term, the larger the weight. It is a measure of **informativeness**

READING THE EQUATION: WHY A LOGARITHM?

Pick a document at random. The chance it contains t is $P(t) = df_t/N$. Then

$$idf_t = \log \frac{N}{df_t} = -\log P(t) \tag{8}$$

which is exactly the **self-information** of the event "this document contains t" [2].

- idf is not an arbitrary formula. It is the number of bits of surprise in seeing the term, measured in whatever base the log uses
- A term you expected to see carries no information. A term you did not expect carries a lot
- The logarithm also tames the scale. Without it, N/df_t ranges over five orders of magnitude and one rare word would dominate every score
- The base only rescales every weight by a constant, so it never changes a ranking. Use $\log_{10}$ by hand and $\log_e$ in code

TWO PRACTICAL VARIANTS

$$\text{smoothed} \quad \mathrm{idf}_t = \log\left(\frac{N}{1+\mathrm{df}_t}\right)+1 \tag{9}$$

$$\text{probabilistic} \quad \mathrm{idf}_t = \log\left(\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\right) \tag{10}$$

- The $1+\mathrm{df}_t$ guards against a query term that appears in *no* document, where the plain formula divides by zero. The trailing $+1$ stops a term in every document from collapsing the whole vector to zero
- `scikit-learn` uses the smoothed form by default, so its numbers will not match a hand computation unless you say so
- The probabilistic form goes negative for terms in more than half the documents, and is the ancestor of the **BM25** weighting we meet in session 21
- All three agree on the ordering of terms. They differ in how hard they punish the head

TF-IDF

Multiplying the two factors gives a composite weight for each term in each document

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{11}$$

- $\text{tf}_{t,d}$ asks: is this term important *inside this document*?
- idf_t asks: is this term informative *across the collection*?
- The product is high only when both are high, which is exactly the profile of a word that characterises the document
- It is very low when a term appears in every document, whatever its count

---

At bottom, idf is a machine for undoing Zipf's law. It discounts the frequent words that the frequency distribution guarantees you will have.

WORKED EXAMPLE: ONE TERM IN A LARGE COLLECTION

IDF measures how rare the term t is across the collection:

$$\mathrm{idf}_t = \log_{10}\left(\frac{\text{total documents in the corpus}}{\text{documents containing t}}\right)$$

Take N = 100,000 documents. The word ***moon*** occurs in 100 of them, so

$$\mathrm{idf}(\textit{moon}) = \log_{10}\frac{100{,}000}{100} = \log_{10} 1000 = 3$$

In a 427-token document, *moon* appears 20 times:

$$\mathrm{tf} = \frac{20}{427} = 0.047 \qquad \text{tf-idf} = 0.047 \times 3 = \boxed{\mathbf{0.141}}$$

Now ***the***, in all 100,000 documents and 45 times in this one:

$$\mathrm{idf}(\textit{the}) = \log_{10} 1 = 0 \qquad \text{tf-idf} = \frac{45}{427} \times 0 = \boxed{\mathbf{0}}$$

The most frequent word in the language scores zero, whatever its count.

The same term *moon* in five documents of that collection. Rank by tf-idf:

Document	tf	tf-idf	Rank
d_1	$20/427 = 0.047$	0.14	3
d_2	$30/250 = 0.120$	0.36	1
d_3	$20/250 = 0.080$	0.24	2
d_9	$5/125 = 0.040$	0.12	4
d_{1000}	$20/1000 = 0.020$	0.06	5

Every document shares the same $\mathrm{idf} = 3$, so within one query term the ranking is decided by tf alone. Note d_1 and d_3: the same raw count of 20, different lengths, different ranks.

WORKED EXAMPLE, STAGE 1: COUNT

A collection of $N = 4$ documents.

d_1: `the moon orbits the earth` d_2: `the earth orbits the sun`
d_3: `the telescope shows the moon` d_4: `the sun is a star`

Rows are terms, columns are documents. Each cell is $tf_{t,d}$.

term	d_1	d_2	d_3	d_4
the	2	2	2	1
moon	1	0	1	0
orbits	1	1	0	0
earth	1	1	0	0
sun	0	1	0	1
telescope	0	0	1	0
shows	0	0	1	0
is	0	0	0	1
a	0	0	0	1
star	0	0	0	1

By raw count, *the* is the most important word in all four documents.

WORKED EXAMPLE, STAGE 2: COUNT DOCUMENTS

df_t counts documents, not occurrences.

term	df_t	N/df_t	$\mathrm{idf}_t = \log_{10}(\mathrm{N}/\mathrm{df}_t)$
the	4	1.00	**0.000**
moon	2	2.00	0.301
orbits	2	2.00	0.301
earth	2	2.00	0.301
sun	2	2.00	0.301
telescope	1	4.00	0.602
shows	1	4.00	0.602
is	1	4.00	0.602
a	1	4.00	0.602
star	1	4.00	0.602

- *the* is in every document, so $\mathrm{N}/\mathrm{df} = 1$ and its weight is 0
- Here *a* has $\mathrm{df} = 1$ and scores the *highest* idf. idf is a statistic of the collection, not a judgement about language, and with $\mathrm{N} = 4$ it measures almost nothing

WORKED EXAMPLE, STAGE 3: MULTIPLY

$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$, cell by cell.

term	d_1	d_2	d_3	d_4
the	0.000	0.000	0.000	0.000
moon	0.301	0.000	0.301	0.000
orbits	0.301	0.301	0.000	0.000
earth	0.301	0.301	0.000	0.000
sun	0.000	0.301	0.000	0.301
telescope	0.000	0.000	0.602	0.000
shows	0.000	0.000	0.602	0.000
is	0.000	0.000	0.000	0.602
a	0.000	0.000	0.000	0.602
star	0.000	0.000	0.000	0.602
$\|\vec{d}\|$	0.5214	0.5214	0.9031	1.0854

The whole first row is zero. **Two documents that share only *the* now share nothing at all.**

WORKED EXAMPLE, STAGE 4: RANK FOR A QUERY

Score a document by adding the tf-idf of the query terms:

$$\text{score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d} \tag{12}$$

query *"the moon"*

doc	raw count	tf-idf
d_1	3	**0.3010**
d_3	3	**0.3010**
d_2	2	0.0000
d_4	1	0.0000

query *"moon orbits"*

doc	raw count	tf-idf
d_1	2	**0.6021**
d_2	1	0.3010
d_3	1	0.3010
d_4	0	0.0000

- On *"the moon"*, raw counts put d_2 third with a score of 2, ahead of d_4. Neither mentions the moon. tf-idf sends both to exactly zero
- The word *the* contributed nothing to either query, so the user was free to type it

WHY TF-IDF WHEN LLM CAN DO A LOT?

- **Hybrid Search Standard:** Modern RAG pipelines and vector databases combine dense embeddings with sparse TF-IDF/BM25 for exact keyword matching.
- **Precision on Rare Terms:** Excels at out-of-vocabulary words, specialized technical jargon, product IDs, and code snippets.
- **High Efficiency:** Ultra-fast CPU execution with zero GPU requirements and minimal latency overhead.
- **Interpretability & Baselines:** Provides transparent feature weights and acts as a fast, reliable baseline model.
- **Pedagogical Foundation:** Essential for introducing core NLP concepts like document frequency weighting and vector space models.

OKAPI BM25 RANKING ALGORITHM

Relevance Score for Query $Q = \{q_1, \ldots, q_n\}$ and Document D:

$$\text{Score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Robertson-Spärck Jones IDF Variant

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

- $f(q_i, D)$: Raw term frequency of term q_i in document D.
- N & $n(q_i)$: Total documents in collection and number of docs containing q_i.
- $\text{avgdl} = \frac{\sum_{d \in D} |d|}{N}$. What is the importance of $\frac{|D|}{\text{avgdl}}$
- $|D|/\text{avgdl}$: Ratio of document length to average collection length.
- k_1 (default $\approx 1.2 - 2.0$): Non-linear term frequency saturation parameter.
- b (default ≈ 0.75): Document length normalization weighting parameter.

BAG OF WORDS

Representing a document by its term weights alone, $[w_1, w_2, w_3, \ldots, w_n]$, is the *bag-of-words* model

- Word order is ignored. Only the weights matter, so *"dog bites man"* and *"man bites dog"* get the same vector
- Two documents with similar bags are assumed to be similar in content
- Every term is its own dimension, orthogonal to every other. So *cat* and *dog* are exactly as unrelated as *cat* and *the*
- That last point is the flaw the rest of the course repairs. tf-idf weights the axes; it does not make similar words share them

---

Contrast: transformers restore word order through position information, and embeddings give related words nearby directions.

OPEN-CLASS AND STOP WORDS

Open-class words: Carry the main semantic content of a sentence

- Contribute to the overall meaning of a sentence
- Include nouns, verbs, adjectives, and adverbs
- **Expansive**: New words can be added to this category over time (e.g., "vlog," "emoji", “Mulligatawny”)
- **Adaptability**: New terms to express emerging concepts, technologies, and innovation

Closed-class (function) words, the usual stop words, serve grammatical purposes

- Include
  - prepositions (in, on, with, ...)
  - conjunctions (and, but, or..)
  - articles (a, an, the)
  - pronouns(he, she, they...)
- Essential for the grammatical integrity of sentences
- Contribute less to the overall meaning of the sentence

DOCUMENTS ARE NOW POINTS IN A SPACE

Each document is a vector of $|V|$ weights. "How similar are these two?" becomes a geometric question.

The obvious first answer is the **dot product**

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{|V|} u_i v_i \tag{13}$$

- It rewards terms that are heavy in both vectors, which is what we want
- It grows with the *lengths* of the vectors, which is not
- Double every count in a document and its dot product with everything doubles, though the document says the same thing

---

A long document would be similar to everything, purely by being long.

COSINE SIMILARITY

Divide the magnitudes out. Compare *direction* instead of length.

$$\cos(\vec{u},\vec{v}) = \frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\,\|\vec{v}\|} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2}\,\sqrt{\sum_i v_i^2}} \tag{14}$$

- $\cos = 1$: the vectors point the same way, so the documents have identical term *proportions*
- $\cos = 0$: orthogonal, so the documents share no weighted term at all
- $\cos < 0$ cannot happen for tf-idf vectors, whose entries are never negative. It does happen for embeddings, where it means opposed directions
- Dividing by $\|\vec{u}\|$ is **length normalisation**. It is why a 200-word abstract can be compared with a 20,000-word thesis
- If both vectors are already unit length, the denominator is 1 and **cosine is just the dot product**

COSINE, DOT PRODUCT, EUCLIDEAN DISTANCE

Measure	Formula	Sensitive to
Dot product	$\vec{u} \cdot \vec{v}$	direction *and* length
Cosine	$\frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|\|\vec{v}\|}$	direction only
Euclidean	$\|\vec{u} - \vec{v}\|$	direction *and* length

For unit-length vectors the three collapse into one relationship:

$$\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\vec{u} \cdot \vec{v} = 2 - 2\cos(\vec{u}, \vec{v}) \tag{15}$$

- So on normalised vectors, ranking by cosine and ranking by Euclidean distance give the *same order*. Many vector databases exploit exactly this
- On unnormalised vectors they disagree, and Euclidean is usually the wrong choice
- **Normalise once, then use whichever is faster**

- **Session 6–7.** Word vectors from co-occurrence counts, compared by cosine
- **Session 10–11.** word2vec and GloVe embeddings. The measuring stick stays cosine; only the space improves
- **Session 17.** Self-attention scores a query against every key with a *raw dot product*, scaled by $\sqrt{d_k}$. The unnormalised form is not a mistake there. Magnitude carries signal
- **Session 21.** Retrieval-augmented generation ranks passages by cosine over dense embeddings, and by BM25 over sparse tf-idf-like weights

The limitation to carry forward

Under a bag of words, a document about *cats* and one about *dogs* have cosine 0. They are as unrelated as *cat* and *the*. Fixing that means changing the space, not the measure.

WORKED EXAMPLE, STAGE 5: COMPARE BY ANGLE

Take $\vec{d_1}$ and $\vec{d_2}$ from the tf-idf table. Each has three non-zero entries of 0.301, and they share two of them, *orbits* and *earth*.

$$\vec{d_1} \cdot \vec{d_2} = (0.301)(0.301) + (0.301)(0.301) = 0.1812$$

$$\|\vec{d_1}\| = \|\vec{d_2}\| = \sqrt{3 \times 0.301^2} = 0.301\sqrt{3} = 0.5214$$

$$\cos(\vec{d_1}, \vec{d_2}) = \frac{0.1812}{0.5214 \times 0.5214} = \frac{0.1812}{0.2719} = \boxed{\mathbf{0.6667}}$$

That is exactly $2/3$: two shared terms out of three equally weighted terms.

Now $\vec{d_1}$ and $\vec{d_3}$, which share only *moon*:

$$\cos(\vec{d_1}, \vec{d_3}) = \frac{(0.301)(0.301)}{0.5214 \times 0.9031} = \frac{0.0906}{0.4709} = \boxed{\mathbf{0.1925}}$$

WORKED EXAMPLE, STAGE 6: WHAT THE WEIGHTING BOUGHT

The same six pairs, scored twice. Once on raw counts, once on tf-idf.

pair	cos on raw tf	cos on tf-idf
d_1, d_2	0.8571	**0.6667**
d_1, d_3	0.7143	0.1925
d_2, d_4	0.5071	0.1601
d_1, d_4	0.3381	**0.0000**
d_2, d_3	0.5714	**0.0000**
d_3, d_4	0.3381	**0.0000**

- Read d_2, d_3. Raw counts call them 0.57 similar. The only word they share is *the*. Under tf-idf their cosine is exactly 0, which is the right answer
- Read d_1, d_2 against d_1, d_3. Raw counts separate them by a factor of 1.2. tf-idf separates them by a factor of 3.5
- The ranking was already right. The weighting made it **decisive**, and that is what a retrieval system needs

SUMMARY

- A bit matrix gives Boolean retrieval and no ranking. Counts give ranking
- tf measures importance inside a document. idf measures informativeness across the collection
- $\mathrm{idf}_t = -\log \mathrm{P}(t)$, so the weight is a count of bits of surprise
- tf-idf is high only when a term is frequent *here* and rare *elsewhere*
- Cosine compares direction, so document length stops mattering
- Every term is still its own axis, so *cat* and *dog* remain orthogonal. That is what the next sessions repair

---

Every number in this session is reproduced by
`python3 weighting.py corpus`

EXERCISES

1. In a collection of $\mathrm{N} = 10{,}000$, term A is in 10 documents, B in 1,000, C in all 10,000. Compute idf in base 10 and rank them. Explain C's value in one sentence.
2. A 500-token document has *data* 15 times ($\mathrm{df} = 50$) and *the* 40 times ($\mathrm{df} = 10{,}000$). Compute both length-normalised tf-idf weights. Which word is the document about?
3. Let $\vec{u} = (3, 0, 4)$ and $\vec{v} = (6, 0, 8)$. Compute the dot product and the cosine. The two documents have identical proportions and different lengths. Which measure notices?
4. Rebuild the four-document example with the smoothed idf, $\log(\mathrm{N}/(1 + \mathrm{df})) + 1$. Which rankings change, and which do not?
5. Add a fifth document that mentions *the moon* once. Recompute $\mathrm{idf}(\textit{moon})$ and explain why every existing weight moved.
6. Build the term–document matrix for the four documents as an inverted index instead. Compare the number of stored values with the 10×4 dense matrix.

REFERENCES I

[1] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0-262-13360-1.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge UP, 2009. Chap. 6, pp. 109–133.