CHAPTER 1,2,3

- **Significance of IR:**
  - Provides solution for fetching relevant, user-specific and accurate information from large database collection.
  - Providing accurate information through use of algorithms

Primary tasks of IR:
- Get the query from the user.
- Process the query by using certain algorithms.
- Provide result to the user.

The objectives of Information Retrieval System are:

- Query a web search engine to show a document of matching patterns based on a ranking system. The top ranked outputs are shown to user.
- Output will always be unstructured data. IR focuses on collecting data from such unstructured document.
- Understand the data corpus (dataset).
- Compression techniques to compress index of dictionary and its posting list.
- Apply retrieval model to construct IR system.
- Apply text clustering and classification techniques for IR.

Steps to process a Boolean query using Inverted index:

Consider of query i.e. AND of terms
For each of the ‘t’ terms, get its posting list.
Process terms in the order of increased document frequency.
Intersecting the two smallest list by AND-ing them together.
If a query contains both AND and OR operators
  Get frequency of all terms.
  Estimate the size of each OR by sum of frequency of its disjunction.
  Process he query in increasing order of size of each disjunctive term.

- **Algorithm for intersection of 2 posting lists:**

```
Intersection(P1,P2)
      Answer← <>                          #Not yet traversed, thus empty
      while (P1!=null)AND(P2!=null)       #end of posting list not reached
       if(docID(P1)==docID(P2))           #matching element in both lists
        ADD(answer,docID(P1))             #put docID in answer
        P1 ← next(P1)
        P2 ← next(p2)                           #Increment both pointers
      Else if(docID(P1)<docID(P2))
            P1←next(p1)
      Else
            P2←next(p2)
Return answer
```

- **The difference between IR system and DBMS are as follows:**

IR System:

- No syntax/semantic for querying (unstructured formal English).
- There exists a ranking technique to extract document wrt web DB.
- Elated documents may be PDF/Img/PPT i.e. unstructured terms and documents.
- Querying happens wrt pattern matching(approx search probabilities)
- Inverted Index Data Structure.

DBMS:

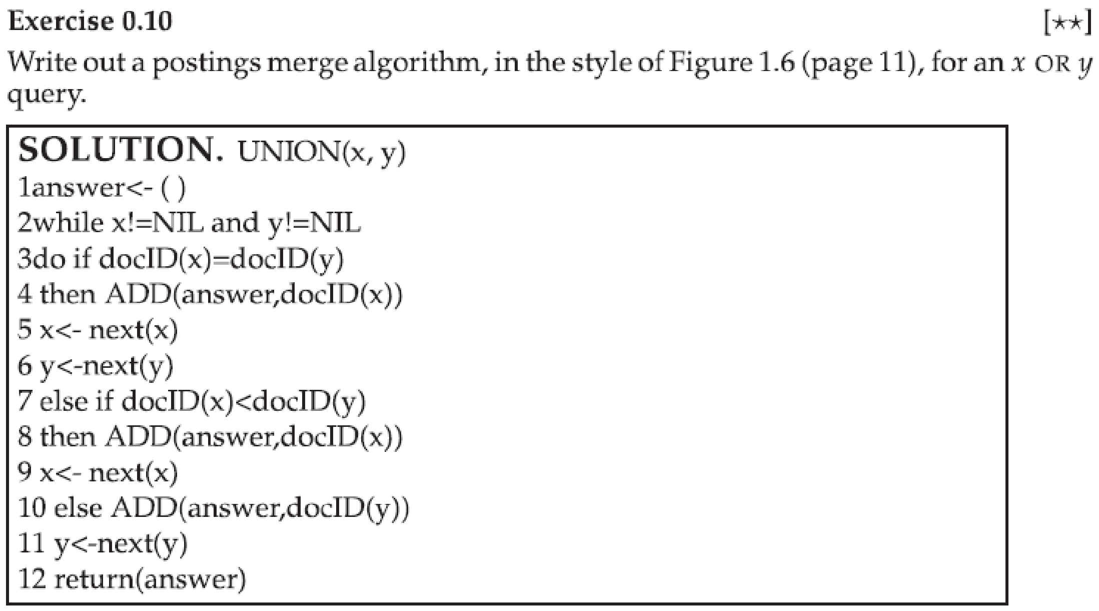
- There exists predefined syntax for querying(structured query).
- No ranking. Record inserted first is displayed first.
- Structured output: attribute value in column, record in row, concerned with specific domain.
- Deterministic output(same output for same query)due to exact searching process.
- Table data structure.

- **Tokenization**: It is the process of chopping a character sequence into pieces called tokens.

  Stemming : It is process that chops off the end of the words usually derivational suffixes.

Normalization: It is the process of canonicalization of tokens so that matches occur despite of superficial difference of character sequence in the term.

QUESTION:



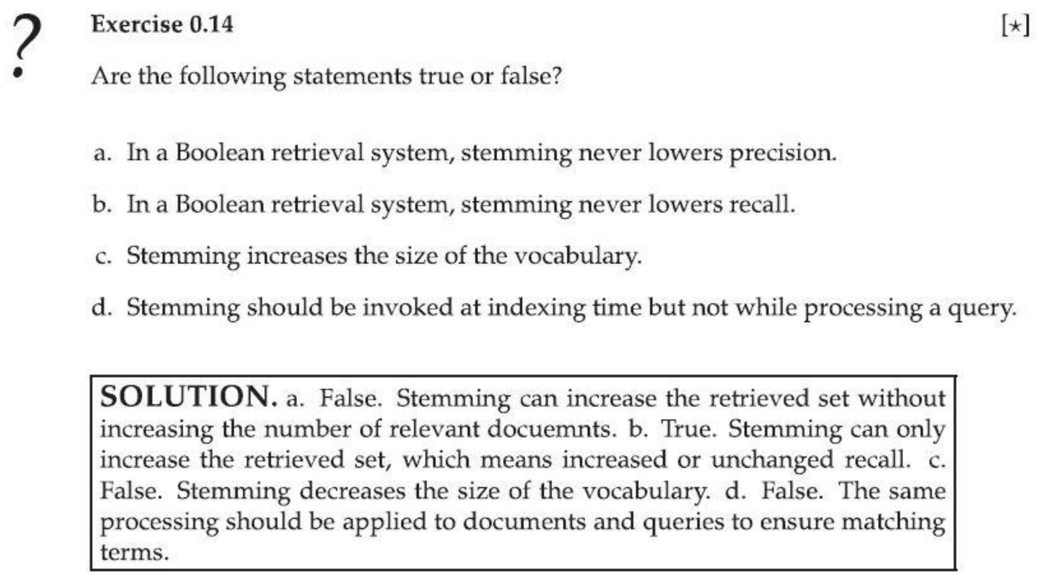
Exercise 0.10 [⋆⋆]

Write out a postings merge algorithm, in the style of Figure 1.6 (page 11), for an x OR y query.

SOLUTION. UNION(x, y)

```
1answer<- ( )
2while x!=NIL and y!=NIL
3do if docID(x)=docID(y)
4 then ADD(answer,docID(x))
5 x<- next(x)
6 y<-next(y)
7 else if docID(x)<docID(y)
8 then ADD(answer,docID(x))
9 x<- next(x)
10 else ADD(answer,docID(y))
11 y<-next(y)
12 return(answer)
```

QUESTION:



?

Exercise 0.14 [⋆]

Are the following statements true or false?

a. In a Boolean retrieval system, stemming never lowers precision.

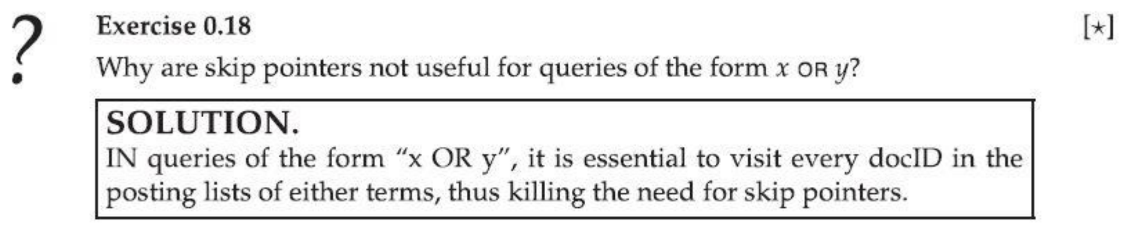
b. In a Boolean retrieval system, stemming never lowers recall.

c. Stemming increases the size of the vocabulary.

d. Stemming should be invoked at indexing time but not while processing a query.

SOLUTION. a. False. Stemming can increase the retrieved set without increasing the number of relevant docuemnts. b. True. Stemming can only increase the retrieved set, which means increased or unchanged recall. c. False. Stemming decreases the size of the vocabulary. d. False. The same processing should be applied to documents and queries to ensure matching terms.

QUESTION:



? **Exercise 0.18** [⋆]

Why are skip pointers not useful for queries of the form x OR y?

SOLUTION.
IN queries of the form "x OR y", it is essential to visit every docID in the posting lists of either terms, thus killing the need for skip pointers.

- ***More skips:***

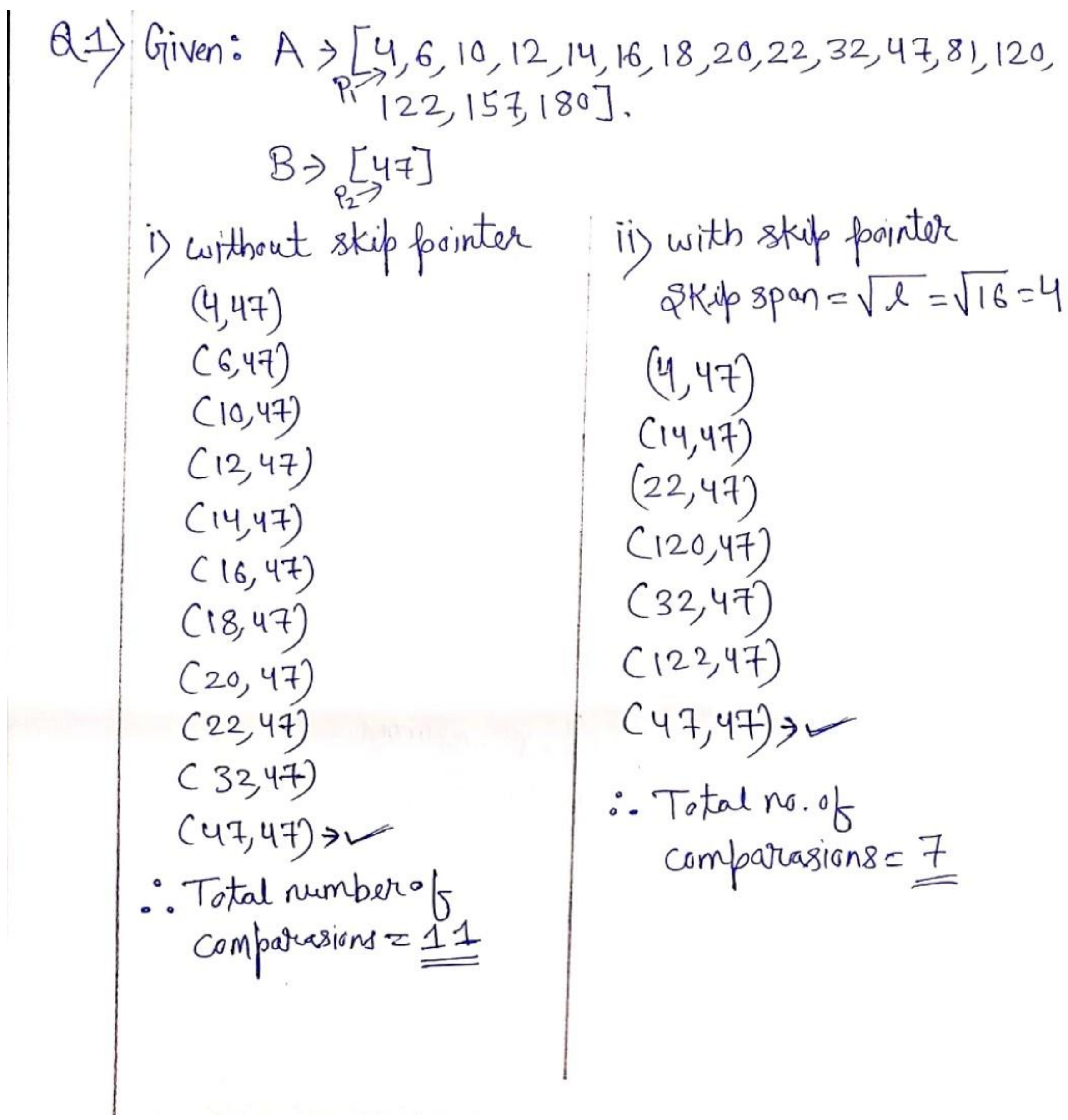
  More skips means shorter skip spans, and that we are more likely to skip. But it also means lots of comparisons to skip pointers, and lots of space storing skip pointers.

  Few Skips:

  Fewer skips means few pointer comparisons, but then long skip spans which means that there will be fewer opportunities to skip.

- ***Advantages of skip pointer over normal pointers:***

1. Faster posting list intersection
2. Efficient and optimal use of posting list
3. If the sizes of our two posting lists are **x** and **y**, the intersection using normal pointer takes **x+y** operations. But, Skip pointers allow us to potentially complete the intersection in less than **x+y** operations.


Q1) Given: A → [4, 6, 10, 12, 14, 16, 18, 20, 22, 32, 47, 81, 120, 122, 157, 180].

$P_1 \rightarrow$

B → [47]

$P_2 \rightarrow$

i) without skip pointer

(4,47)

(6,47)

(10,47)

(12,47)

(14,47)

(16,47)

(18,47)

(20,47)

(22,47)

(32,47)

(47,47) → ✓

∴ Total number of comparasions = $\underline{\underline{11}}$

ii) with skip pointer

Skip span = $\sqrt{l} = \sqrt{16} = 4$

(4,47)

(14,47)

(22,47)

(120,47)

(32,47)

(122,47)

(47,47) → ✓

∴ Total no. of comparasions = $\underline{\underline{7}}$

- ***Major steps involved in Inverted index construction:***

Collect the documents to be indexed.
Tokenize the text
Do linguistic pre processing of tokens.
Index the documents that each term occurs in.

- ***Data Structures used for Inverted Index:***

Fixed Size array, single LinkedList, Variable Sized array

- ***Difference between Inverted index and positional index:***

  Inverted Index:
  This index contains the terms with its posting list (contains the documents).
  Eg.

  Positional Index:
  This index contains the term, no. of occurance of the term along with docID and the respective positions where they are present in the document.
  Eg.

- ***Find two differently spelled proper nouns whose soundex codes are same.***

  1. Odisha and Odisa
     For Odisha, O30200
     Soundex code: O320

     For Odisa, O3020
     Soundex code: O320
  2. Bharun and Barun
     For Bharun, B00605
     Soundex code: B650
     For Barun, B0605
     Soundex code: B650

QUESTION:

? **Exercise 0.21** [⋆]

Assume a biword index. Give an example of a document which will be returned for a query of New York University but is actually a false positive which should not be returned.

SOLUTION.
Document="Some alumni had arrived from New York. University faculty said that Stanford is the best place to study....".

QUESTION:

? **Exercise 0.41**

Find two differently spelled proper nouns whose soundex codes are the same.

SOLUTION. Mary, Nira (Soundex code = 5600).

Exercise 0.42

Find two phonetically similar proper nouns whose soundex codes are different.

SOLUTION. Chebyshev, Tchebycheff.

QUESTION:

? **Exercise 0.28**

In the permuterm index, each permuterm vocabulary term points to the original vocabulary term(s) from which it was derived. How many original vocabulary terms can there be in the postings list of a permuterm vocabulary term?

SOLUTION.
One. (If it weren't for the $ terminal symbol, we could have multiple original vocabulary terms resulting from rotations of the same permuterm vocabulary term, e.g., leaf and flea.)

QUESTION:

Exercise 0.33

Give an example of a sentence that falsely matches the wildcard query mon*h if the search were to simply use a conjunction of bigrams.

SOLUTION. His personality is *moonish*.

Soundex Algorithm

The variations in different soundex algorithms have to do with the conversion of terms to 4-character forms. A commonly used conversion results in a 4-character code, with the first character being a letter of the alphabet and the other three being digits between 0 and 9.

- Retain the first letter of the term.
- Change all occurrences of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
- Change letters to digits as follows: B, F, P, V to 1. C, G, J, K, Q, S, X, Z to 2. D,T to 3. L to 4. M, N to 5. R to 6.
- Repeatedly remove one out of each pair of consecutive identical digits.

Contd..

- Remove all zeros from the resulting string. Pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits.
- Change letters to digits as follows: B, F, P, V to 1. C, G, J, K, Q, S, X, Z to 2. D,T to 3. L to 4. M, N to 5. R to 6.
- Repeatedly remove one out of each pair of consecutive identical digits.
- Remove all zeros from the resulting string. Pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits.

Example

Soundex of HERMAN with HERMANN

- Retain H
- *ERMAN* → *0RM0N*
- *0RM0N* → *06505*
- *06505* → *06505*
- *06505* → *655*
- Return *H655*
- Note: *HERMANN* will generate the same code

Jaccard Coefficient

To find the k-gram overlap between two postings list, we use the Jaccard coefficient. Here, A and B are two sets (postings lists), A for the misspelt word and B for the corrected word.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

CHAPTER 4 (BSBI, SPIMI, DISTRIBUTED INDEXING, MAP REDUCE)

Hardware basics

- Access to data is much faster in memory than on disk. (roughly a factor of 10)
- Disk seeks are "idle" time: No data is transferred from disk while the disk head is being positioned.
- To optimize transfer time from disk to memory: one large chunk is faster than many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks). Block sizes: 8KB to 256 KB
- Servers used in IR systems typically have many GBs of main memory and TBs of disk space.
- Fault tolerance is expensive: It's cheaper to use many regular machines than one fault tolerant machine.

Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?

•No: Sorting very large sets of records on disk is too slow–too many disk seeks.

•We need an external sorting algorithm.

Index construction

- **Blocked Sort-Based Indexing**
  - implement a term to term-ID mapping
  - very slow index construction
- **Single-pass in-memory indexing**
  - Generate separate dictionaries for each block – no need to maintain term-termID mapping
  - Don't sort

- ***Blocked Sort Based Indexing Algorithm:***

BSBI (i) segments the collection into parts of equal size, (ii) sorts the termID–docID pairs of each part in memory, (iii) stores intermediate sorted results on disk, and (iv) merges all intermediate results into the final index.

```
BSBINDEXCONSTRUCTION()
1 n ← 0
2 while (all documents have not been processed)
3 do n ← n +1
4 block ← PARSENEXTBLOCK ()
5 BSBI-INVERT(block)
6 WRITEBLOCKTODISK(block, fn)
7   MERGEBLOCKS( f1, . . . , fn; fmerged)
```

Problem with BSBI algorithm

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.
- Actually, we could work with term,docID postings instead of termID,docID postings . . .
- . . . but then intermediate files become very large. (We would end up with a scalable, but very slow index construction method.)

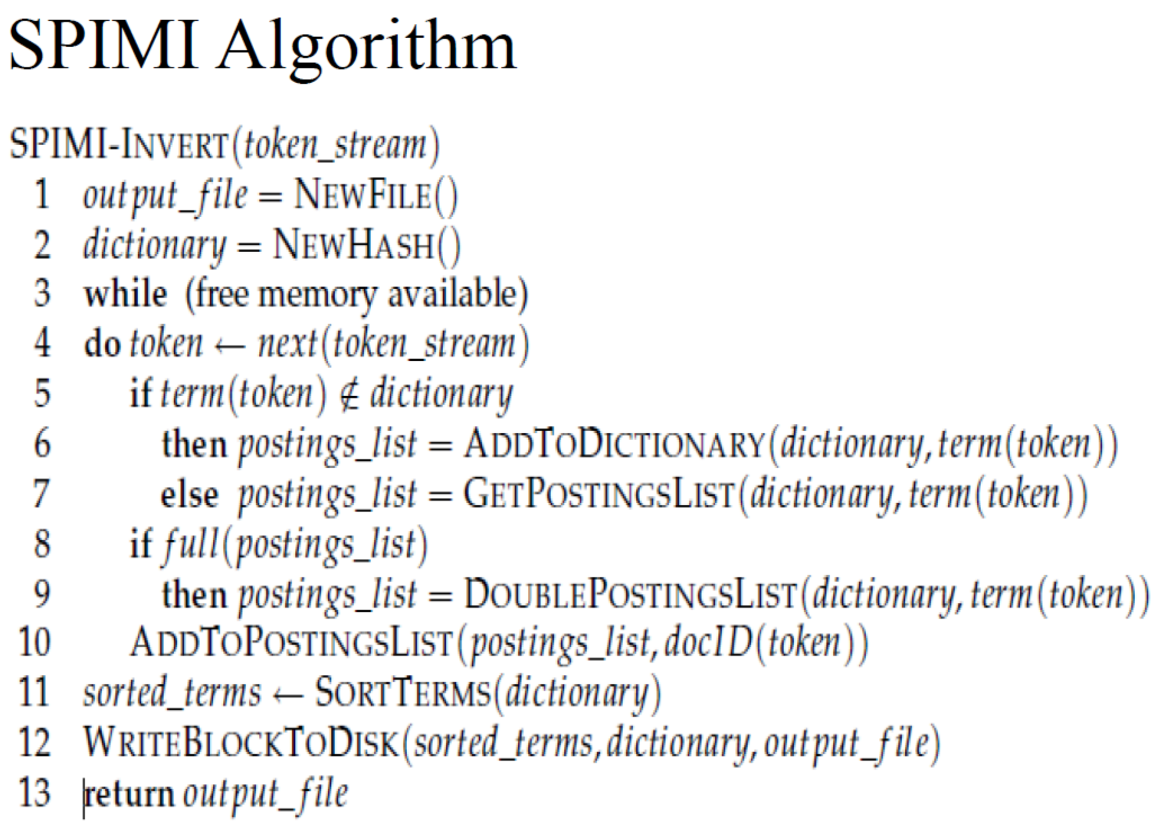
Blocked sort-based indexing has excellent scaling properties, but it needs a data structure for mapping terms to termIDs. For very large collections, this data structure does not fit into memory.

Single-pass in-memory indexing

A more scalable alternative is *single-pass in-memory indexing* or *SPIMI*. SPIMI uses terms instead of termIDs, writes each block's dictionary to disk, and then starts a new dictionary for the next block. SPIMI can index collections of any size as long as there is enough disk space available.

Single-pass in-memory indexing (SPIMI)

- Key idea 1: Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.



SPIMI Algorithm

```
SPIMI-INVERT($token\_stream$)
output_file = NEWFILE()
dictionary = NEWHASH()
while (free memory available)
do token ← next(token_stream)
    if term(token) ∉ dictionary
        then postings_list = ADDTODICTIONARY(dictionary, term(token))
        else postings_list = GETPOSTINGSLIST(dictionary, term(token))
    if full(postings_list)
        then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
    ADDTOPOSTINGSLIST(postings_list, docID(token))
sorted_terms ← SORTTERMS(dictionary)
WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
return output_file
```

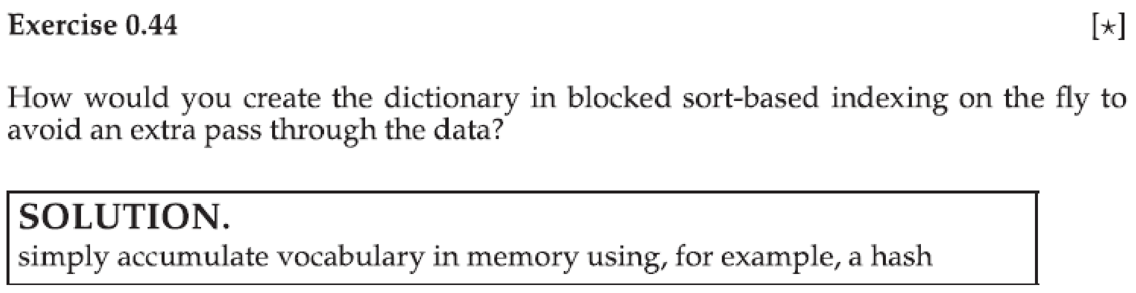
A ***difference between BSBI and SPIMI*** is that SPIMI adds a posting directly to its postings list. Instead of first collecting all termID–docID pairs and then sorting them (as we did in BSBI), each postings list is dynamic (i.e., its size is adjusted as it grows) and it is immediately available to collect postings.

Advantages of SPIMI:

1. Generate separate dictionaries for each block –no need to maintain term-termID mapping.

2. It is faster because there is no sorting required, and it saves memory because we keep track of the term a postings list belongs to, so the termIDs of postings need not be stored.

QUESTION:



Exercise 0.44 [⋆]

How would you create the dictionary in blocked sort-based indexing on the fly to avoid an extra pass through the data?

SOLUTION.
simply accumulate vocabulary in memory using, for example, a hash

Distributed indexing

Maintain a master machine directing the indexing job – considered "safe"

Break up indexing into sets of parallel tasks

Master machine assigns each task to an idle machine from a pool.

Parallel tasks

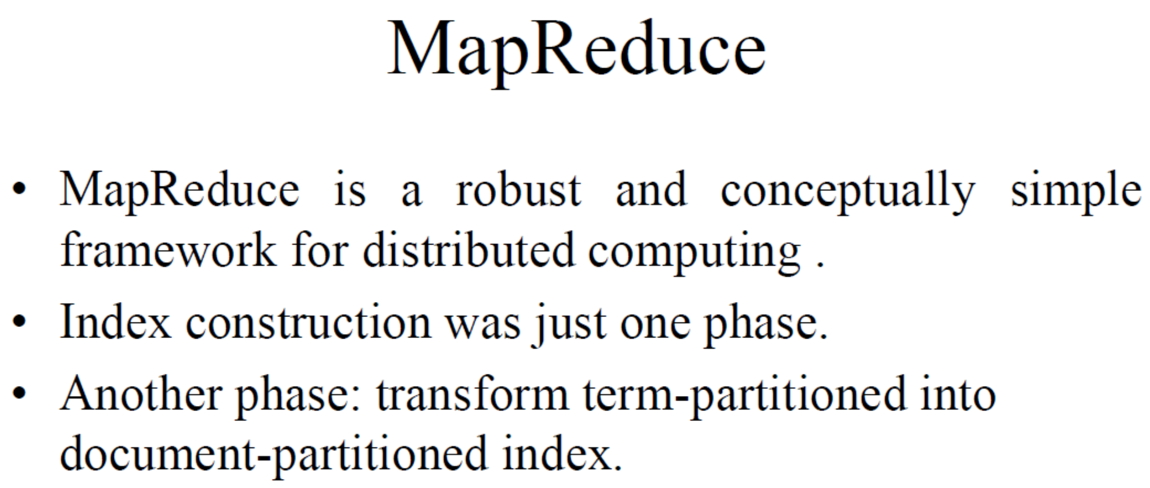
- We will define two sets of parallel tasks and deploy two types of machines to solve them:
  - Parsers
  - Inverters
- Break the input document collection into splits (corresponding to blocks in BSBI/SPIMI)
- Each split is a subset of documents.

Parsers

- Master assigns a split to an idle parser machine.
- Parser reads a document at a time and emits (term,docID)-pairs.
- Parser writes pairs into j term-partitions.
- Each for a range of terms' first letters
  e.g., a-f, g-p, q-z (here: j = 3)

Inverters

- An inverter collects all (term,docID) pairs (= postings) for one term-partition (e.g., for a-f)
- Sorts and writes to postings lists



MapReduce

- MapReduce is a robust and conceptually simple framework for distributed computing .
- Index construction was just one phase.
- Another phase: transform term-partitioned into document-partitioned index.

Advantages of MapReduce:

1. MapReduce offers a robust and conceptually simple framework for implementing index construction in a distributed environment.

2. By the help of semiautomatic method for splitting index construction into smaller tasks, it can scale large collections, given computer clusters of sufficient size.

CHAPTER 5 (HEAP'S LAW, ZIFF'S LAW, DICTIONARY COMPRESSION)

Why compression? (in general)

- Use less disk space (saves money)
- Keep more stuff in memory (increases speed)
- Increase speed of transferring data from disk to memory (again, increases speed)
  - [read compressed data and decompress in memory] is faster than [read uncompressed data]
- Decompression algorithms are fast.

Why compression in information retrieval?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk

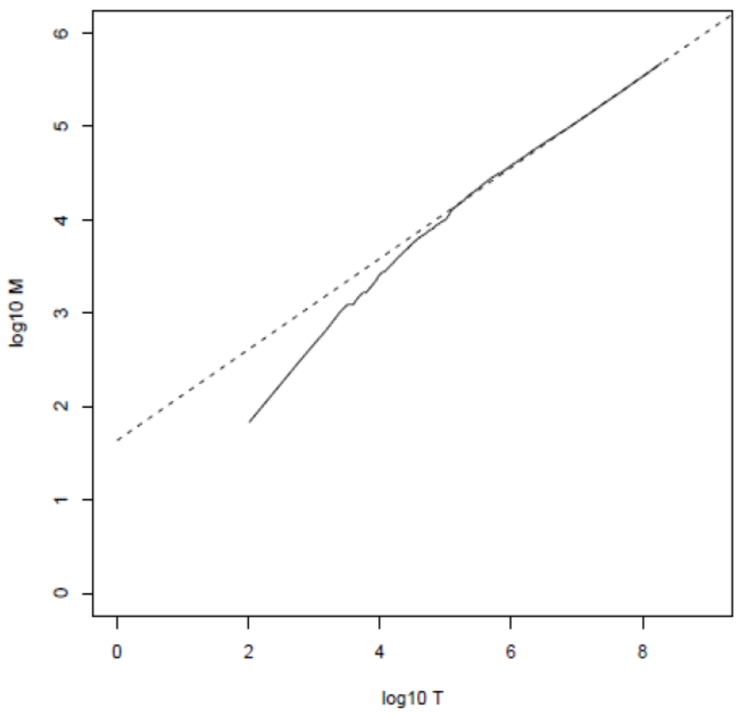
Lossy vs. lossless compression

- Lossy compression: Discard some information.
- Lossless compression: All information is preserved.

Heaps' law

- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$.
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - Empirical law

Heaps' law for Reuters



Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

- Regardless of the values of the parameters for a particular collection, ***Heaps' law suggests that*** (i) the dictionary size continues to increase with more documents in the collection, rather than a maximum vocabulary size being reached, and (ii) the size of the dictionary is quite large for large collections. These two hypotheses have been empirically shown to be true of large text collections . So dictionary compression is important for an effective information retrieval system.

Zipf's law

Zipf's law: The i^{th} most frequent term has frequency proportional to $1/i$.

$\mathrm{cf}_i \propto \frac{1}{i}$

cf is collection frequency: the number of occurrences of the term in the collection.

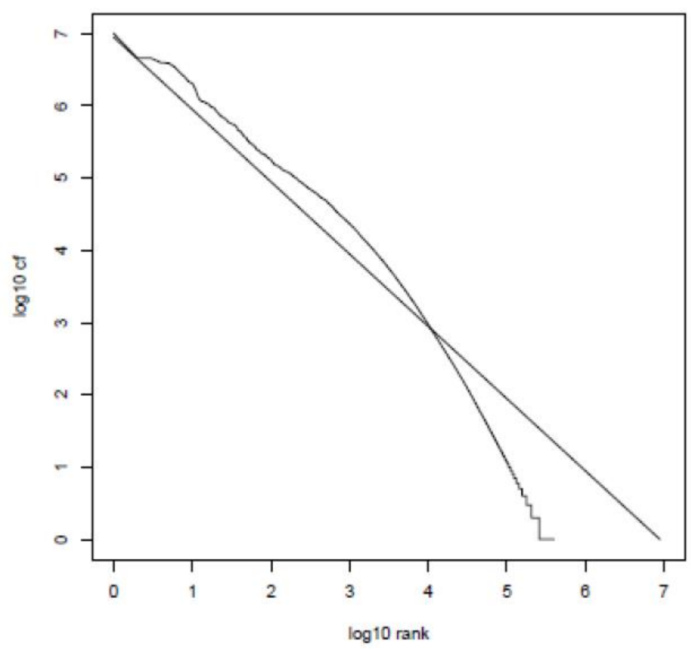
So if the most frequent term (*the*) occurs cf_1 times, then the second most frequent term (*of*) has half as many occurrences $\mathrm{cf}_2 = \frac{1}{2}\mathrm{cf}_1 \ \ldots$

. . . and the third most frequent term (*and*) has a third as many occurrences $\mathrm{cf}_3 = \frac{1}{3}\mathrm{cf}_1$ etc.

Equivalent: $\mathrm{cf}_i = ci^k$ and $\log \mathrm{cf}_i = \log c + k \log i$ (for $k = -1$)

Example of a power law

Zipf's law for Reuters

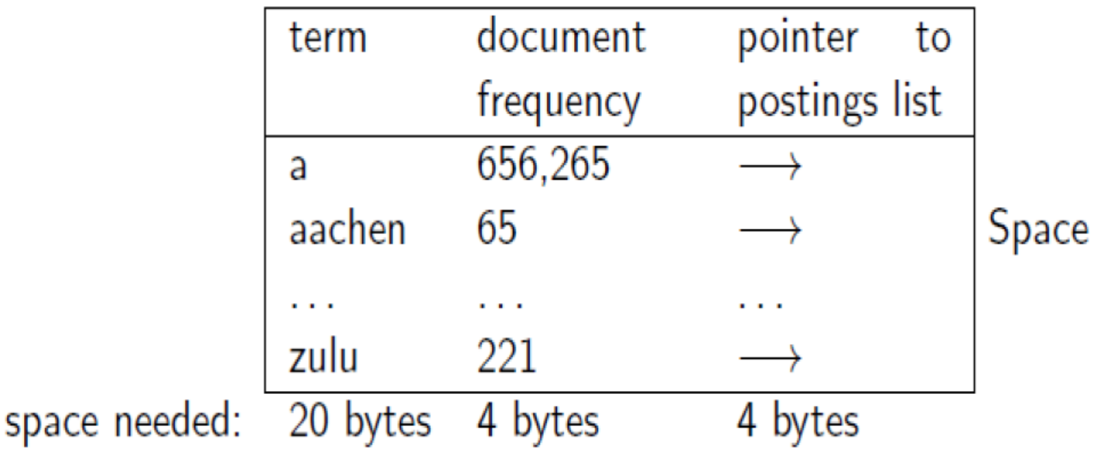


Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.

Dictionary compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones, onboard computers, fast startup time
- So compressing the dictionary is important.

Recall: Dictionary as array of fixed-width entries



	term	document frequency	pointer to postings list	
	a	656,265	$\longrightarrow$	
	aachen	65	$\longrightarrow$	Space
	...	...	...	
	zulu	221	$\longrightarrow$	
space needed:	20 bytes	4 bytes	4 bytes	

- for Reuters: (20+4+4)*400,000 = 11.2 MB

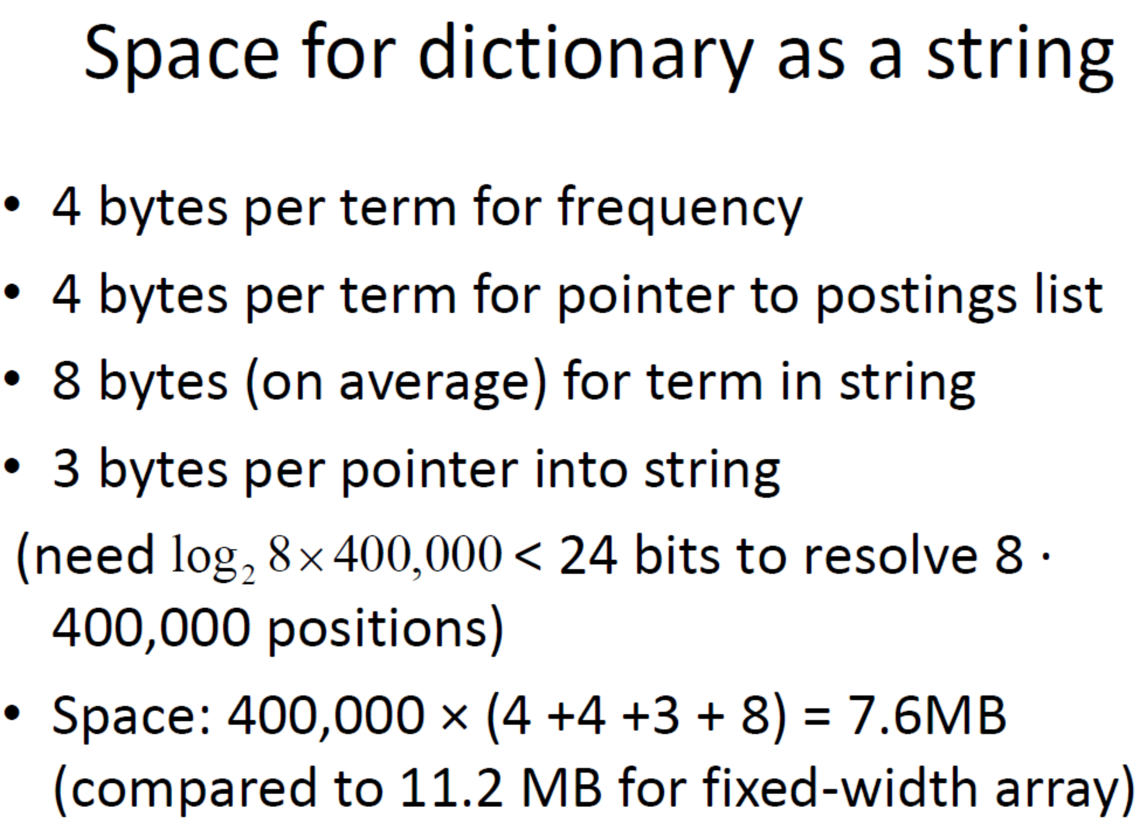
Limitation of Fixed-width entries

- Most of the bytes in the term column are wasted.
  - We allot 20 bytes for terms of length 1.
- We can’t handle hydrochlorofluorocarbons and supercalifragilisticexpialidocious
- Average length of a term in English: 8 characters (or a little bit less)
- How can we use on average 8 characters per term?

Dictionary as a string

- It store the dictionary terms as one long string of characters.
- Term pointers mark the end of the preceding term and the beginning of the next.

  For example, the first three terms in this example are systile, syzygetic, and syzygial



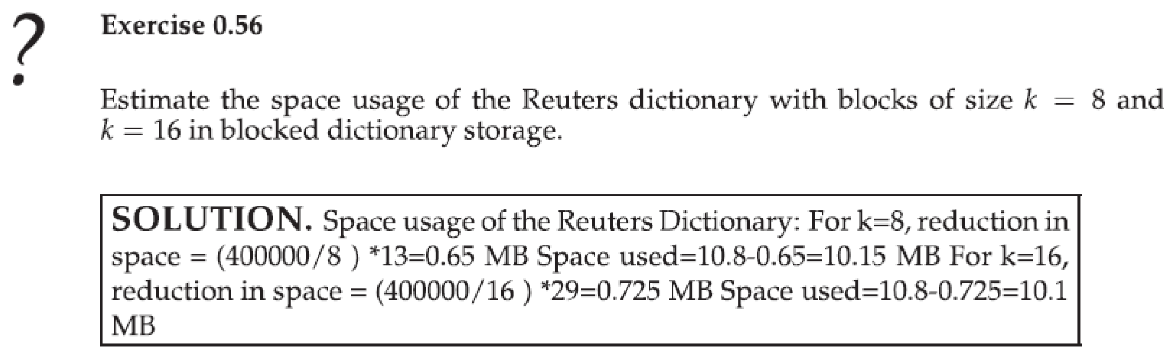
Space for dictionary as a string

- 4 bytes per term for frequency
- 4 bytes per term for pointer to postings list
- 8 bytes (on average) for term in string
- 3 bytes per pointer into string

(need $\log_2 8 \times 400{,}000 < 24$ bits to resolve 8 · 400,000 positions)

- Space: 400,000 × (4 +4 +3 + 8) = 7.6MB (compared to 11.2 MB for fixed-width array)

*** QUESTION:**



? **Exercise 0.56**

Estimate the space usage of the Reuters dictionary with blocks of size $k = 8$ and $k = 16$ in blocked dictionary storage.

SOLUTION. Space usage of the Reuters Dictionary: For k=8, reduction in space = (400000/8) *13=0.65 MB Space used=10.8-0.65=10.15 MB For k=16, reduction in space = (400000/16) *29=0.725 MB Space used=10.8-0.725=10.1 MB

CHAPTER 6 (TERM FREQUENCY AND WEIGHTING, VECTOR MODEL FOR SCORING)

Term frequency (tf)

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use *tf* when computing query-document match scores.
  - But how?
- Raw term frequency is not what we want because:
- A document with *tf* = 10 occurrences of the term is more relevant than a document with *tf* = 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Instead of raw frequency: Log frequency weighting

- Score for a document-query pair: sum over terms t in both q

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

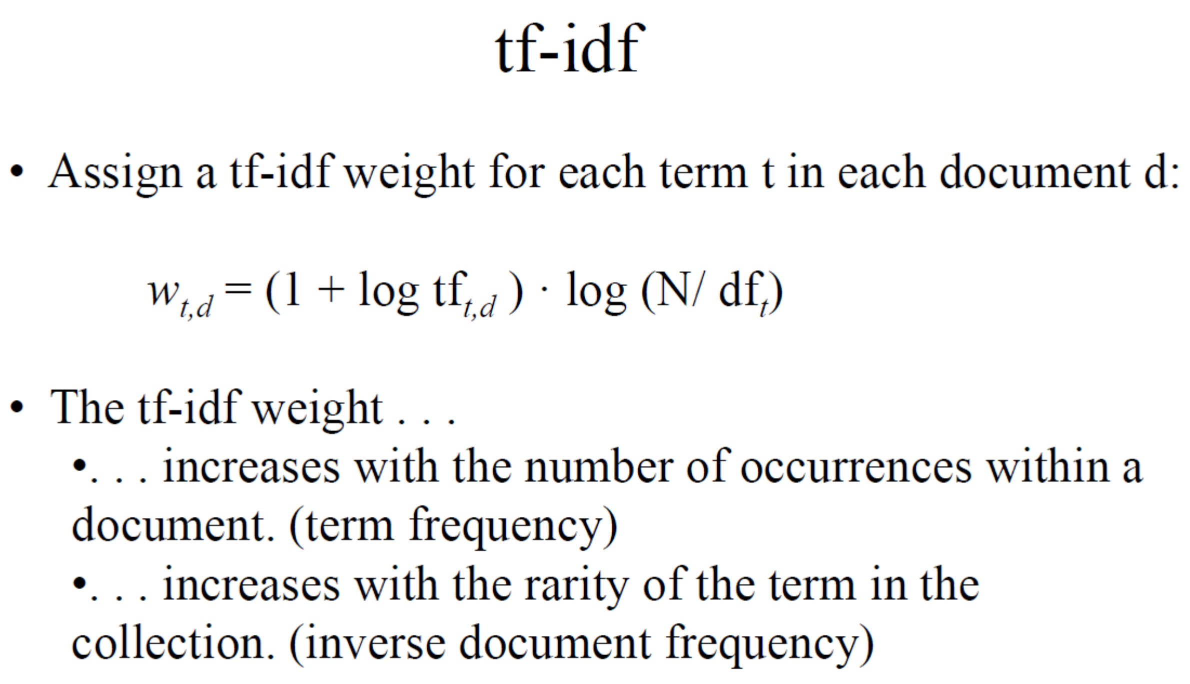
- $tf_{t,d} \to w_{t,d}$:
- $0 \to 0$, $1 \to 1$, $2 \to 1.3$, $10 \to 2$, $1000 \to 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d:
- tf-matching-score(*q*, *d*) = $\sum_{t \in q \cap d} (1 + \log tf_{t,d})$
- The score is 0 if none of the query terms is present in the document.

idf weight

- df_t is the document frequency, the number of documents that *t* occurs in.
- df_t is an inverse measure of the informativeness of term *t*.
- We define the idf weight of term *t* as follows:
- $\mathrm{idf}_t = \log_{10} (N/\mathrm{df}_t)$

  (N is the number of documents in the collection.)
- idf_t is a measure of the informativeness of the term.
- [log N/df_t] instead of [N/df_t] to “dampen” the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

*** Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low.**



tf-idf

- Assign a tf-idf weight for each term t in each document d:

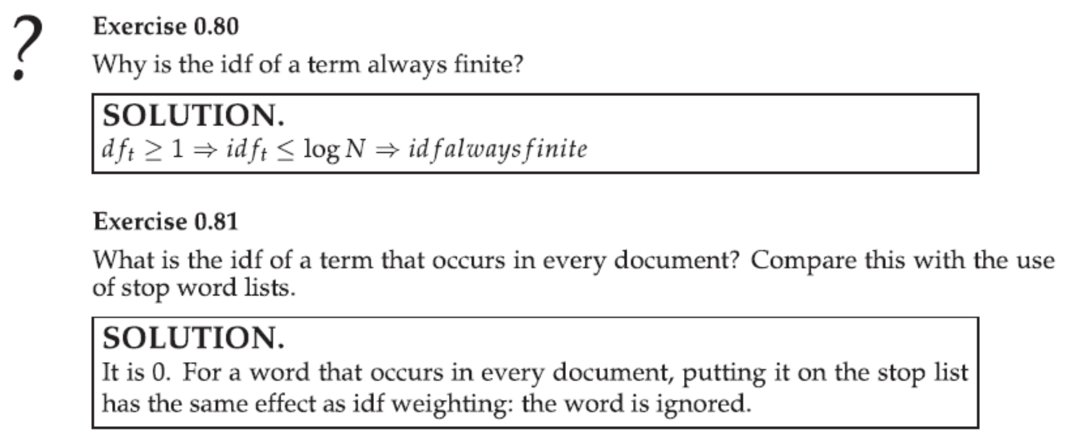
$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log (N/\, \mathrm{df}_t)$$

- The tf-idf weight . . .
  - . . . increases with the number of occurrences within a document. (term frequency)
  - . . . increases with the rarity of the term in the collection. (inverse document frequency)

*** The tf-idf weight is :**

1. highest when *t* occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

* QUESTION:



?

Exercise 0.80

Why is the idf of a term always finite?

SOLUTION.
$df_t \geq 1 \Rightarrow idf_t \leq \log N \Rightarrow idfalwaysfinite$

Exercise 0.81

What is the idf of a term that occurs in every document? Compare this with the use of stop word lists.

SOLUTION.
It is 0. For a word that occurs in every document, putting it on the stop list has the same effect as idf weighting: the word is ignored.

* QUESTION:

?

Exercise 0.86

If we were to stem jealous and jealousy to a common stem before setting up the vector space, detail how the definitions of tf and idf should be modified.

SOLUTION. If jealousy and jealous are stemmed to a common term t, then their tf's and their df's would be added together, in computing the tf-idf weights.

* QUESTION:

Example 6.4: We now consider the query best car insurance on a fictitious collection with $N = 1{,}000{,}000$ documents where the document frequencies of auto, best, car and insurance are respectively 5000, 50000, 10000 and 1000.

term	query				document			product
	tf	df	idf	$w_{t,q}$	tf	wf	$w_{t,d}$	
auto	0	5000	2.3	0	1	1	0.41	0
best	1	50000	1.3	1.3	0	0	0	0
car	1	10000	2.0	2.0	1	1	0.41	0.82
insurance	1	1000	3.0	3.0	2	2	0.82	2.46

In this example the weight of a term in the query is simply the idf (and zero for a term not in the query, such as auto); this is reflected in the column header $w_{t,q}$ (the entry for auto is zero because the query does not contain the termauto). For documents, we use tf weighting with no use of idf but with Euclidean normalization. The former is shown under the column headed wf, while the latter is shown under the column headed $w_{t,d}$. Invoking (6.9) now gives a net score of $0 + 0 + 0.82 + 2.46 = 3.28$.

* QUESTION:

Exercise 0.91

Compute the vector space similarity between the query "digital cameras" and the document "digital cameras and video cameras" by filling out the empty columns in Table 6.1. Assume $N = 10{,}000{,}000$, logarithmic term weighting (wf columns) for query and document, idf weighting for the query only and cosine normalization for the document only. Treat and as a stop word. Enter term counts in the tf columns. What is the final similarity score?

SOLUTION.

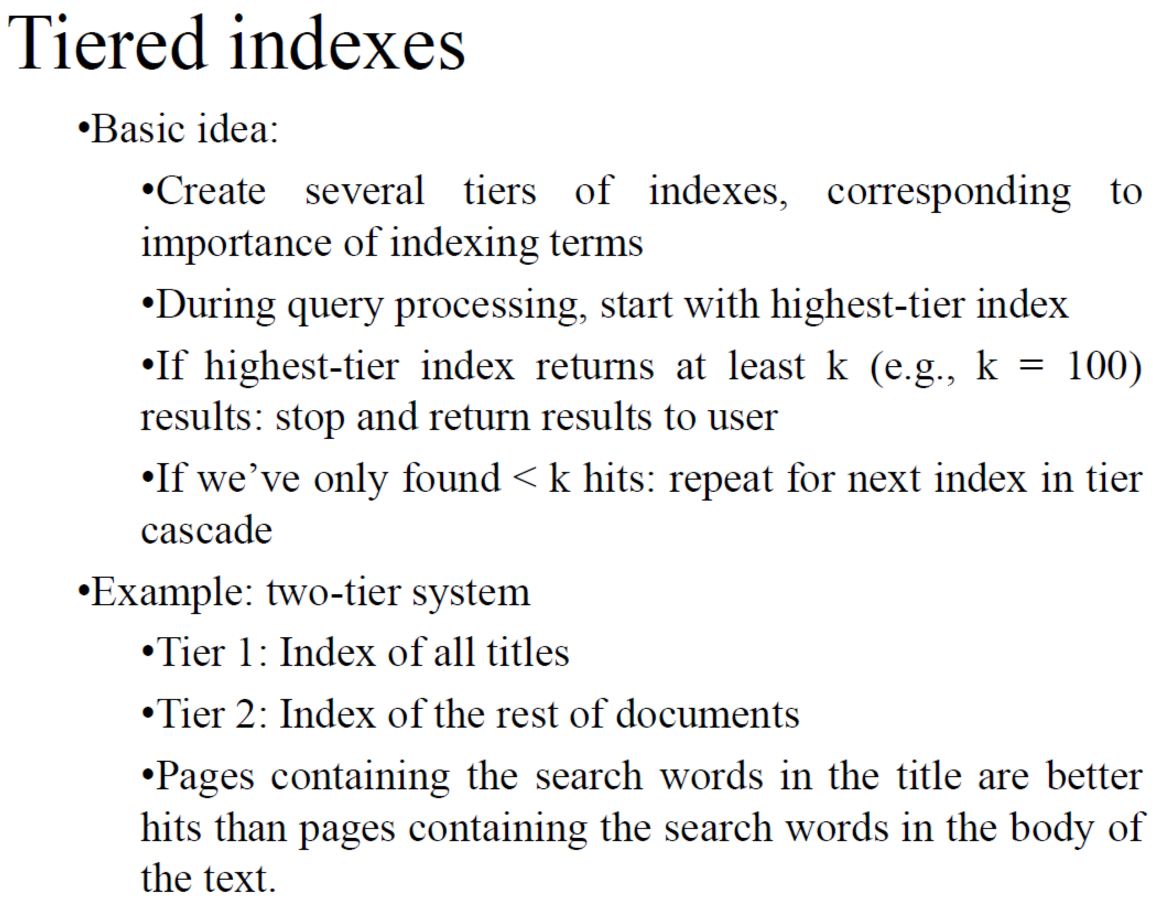
	query					document			
word	tf	wf	df	idf	q_i = wf-idf	tf	wf	d_i = normalized wf	$q_i \cdot d_i$
digital	1	1	10,000	3	3	1	1	0.52	1.56
video	0	0	100,000	2	0	1	1	0.52	0
cameras	1	1	50,000	2.3	2.3	2	1.3	0.68	1.56

Similarity score: $1.56 + 1.56 = 3.12$.

Normalized similarity score is also correct: 3.12/length(query) = 3.12/3.78 = 0.825

CHAPTER 7 (COMPONENTS OF AN IR SYSTEM)

- ***Components of an information retrieval system***



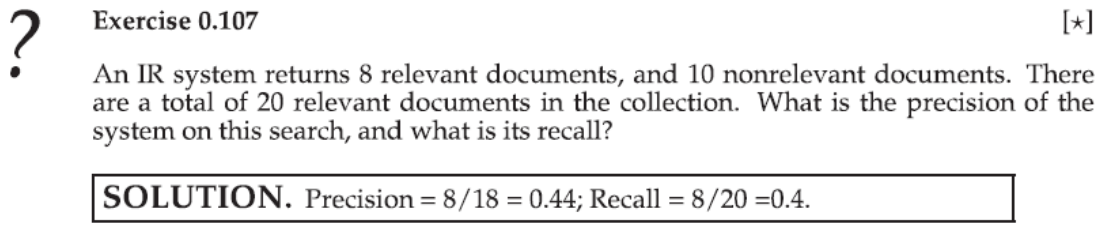
Tiered indexes

- Basic idea:
  - Create several tiers of indexes, corresponding to importance of indexing terms
  - During query processing, start with highest-tier index
  - If highest-tier index returns at least k (e.g., k = 100) results: stop and return results to user
  - If we've only found < k hits: repeat for next index in tier cascade
- Example: two-tier system
  - Tier 1: Index of all titles
  - Tier 2: Index of the rest of documents
  - Pages containing the search words in the title are better hits than pages containing the search words in the body of the text.

CHAPTER 8 (STANDARD TEST , PRECISSION, RECALL)

- ***Standard test collections***
  - CRANFIELD The *Cranfield* collection
  - TREC *Text Retrieval Conference (TREC)*
  - CLEF Cross Language Evaluation Forum (*CLEF*).

* QUESTION:

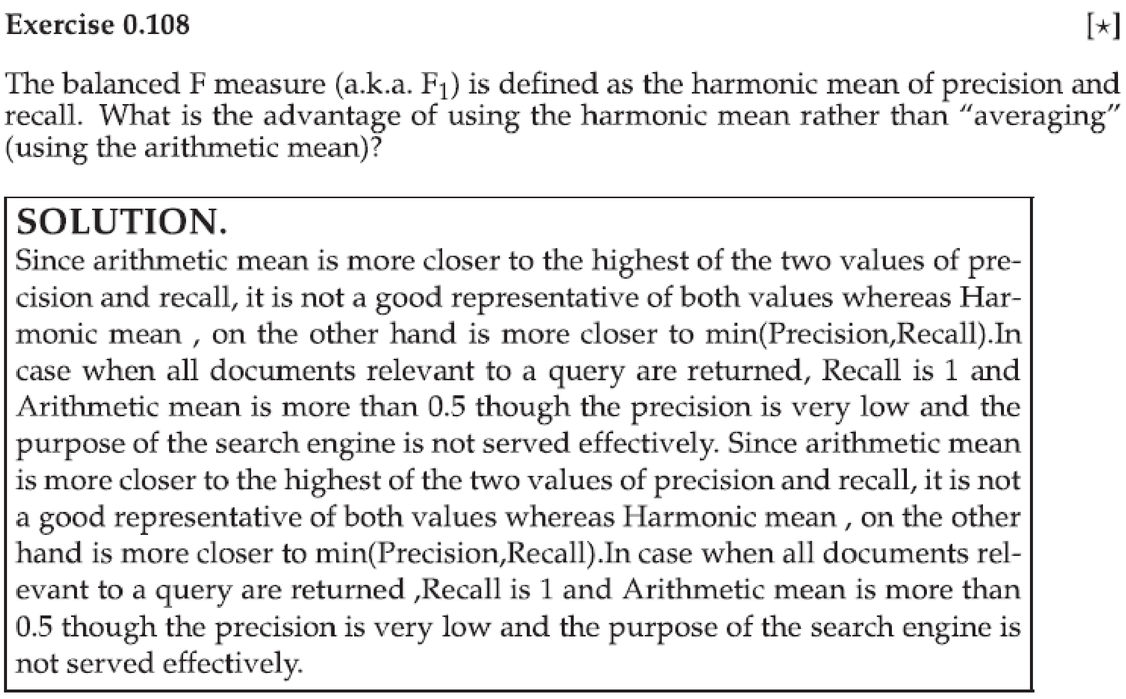


Exercise 0.107 [⋆]

An IR system returns 8 relevant documents, and 10 nonrelevant documents. There are a total of 20 relevant documents in the collection. What is the precision of the system on this search, and what is its recall?

SOLUTION. Precision = 8/18 = 0.44; Recall = 8/20 =0.4.

* QUESTION:



Exercise 0.108 [⋆]

The balanced F measure (a.k.a. F_1) is defined as the harmonic mean of precision and recall. What is the advantage of using the harmonic mean rather than "averaging" (using the arithmetic mean)?

SOLUTION.
Since arithmetic mean is more closer to the highest of the two values of precision and recall, it is not a good representative of both values whereas Harmonic mean , on the other hand is more closer to min(Precision,Recall).In case when all documents relevant to a query are returned, Recall is 1 and Arithmetic mean is more than 0.5 though the precision is very low and the purpose of the search engine is not served effectively. Since arithmetic mean is more closer to the highest of the two values of precision and recall, it is not a good representative of both values whereas Harmonic mean , on the other hand is more closer to min(Precision,Recall).In case when all documents relevant to a query are returned ,Recall is 1 and Arithmetic mean is more than 0.5 though the precision is very low and the purpose of the search engine is not served effectively.

CHAPTER 9 (RELEVANCE FEEDBACK)

RELEVANCE FEEDBACK

The idea of *relevance feedback* (RF) is to involve the user in the retrieval process so as to improve the final result set. In particular, the user gives feedback on the relevance of documents in an initial set of results. The basic procedure is:

- The user issues a (short, simple) query.
- The system returns an initial set of retrieval results.
- The user marks some returned documents as relevant or nonrelevant.
- The system computes a better representation of the information need based on the user feedback.
- The system displays a revised set of retrieval results.

When does relevance feedback work?

The success of relevance feedback depends on certain assumptions. Firstly, the user has to have sufficient knowledge to be able to make an initial query which is at least somewhere close to the documents they desire. This is needed anyhow for successful information retrieval in the basic case, but it is important to see the kinds of problems that relevance feedback cannot solve alone.

Cases where relevance feedback alone is not sufficient include:

- Misspellings. If the user spells a term in a different way to the way it is spelled in any document in the collection, then relevance feedback is unlikely to be effective.
- Cross-language information retrieval. Documents in another language are not nearby in a vector space based on term distribution. Rather, documents in the same language cluster more closely together.
- Mismatch of searcher’s vocabulary versus collection vocabulary. If the user searches for laptop but all the documents use the term notebook computer, then the query will fail, and relevance feedback is again most likely ineffective.

* QUESTION:

Exercise 0.122 [⋆]

Give three reasons why relevance feedback has been little used in web search.

SOLUTION. i. RF slows down returning results as you need to run two sequential queries, the second of which is slower to compute than the first. Web users hate to be kept waiting. ii. RF is mainly used to increase recall, but web users are mainly concerned about the precision of the top few results. iii. RF is one way of dealing with alternate ways to express an idea (synonymy), but indexing anchor text is commonly already a good way to solve this problem. iv. RF is an HCI failure: it's too complicated for the great unwashed to understand.

* QUESTION:

SOLUTION. The vectors are:

	q	d1	d2	d3
Ariolimax	0	1	0	0
banana	1	1	1	0
Campus	0	0	0	1
columbianus	0	1	0	0
Cruz	0	0	1	1
Mascot	0	0	0	1
mountains	0	0	1	0
Santa	0	0	1	1
slug	1	1	1	0

Using Rocchio relevance feedback mechanism, with $\alpha = \beta = \gamma = 1$, and after changing negative components back to 0:

$$\vec{q}_r = [\frac{1}{2}, 2, 0, \frac{1}{2}, 0, 0, \frac{1}{2}, 0, 2]$$

?

Exercise 0.121

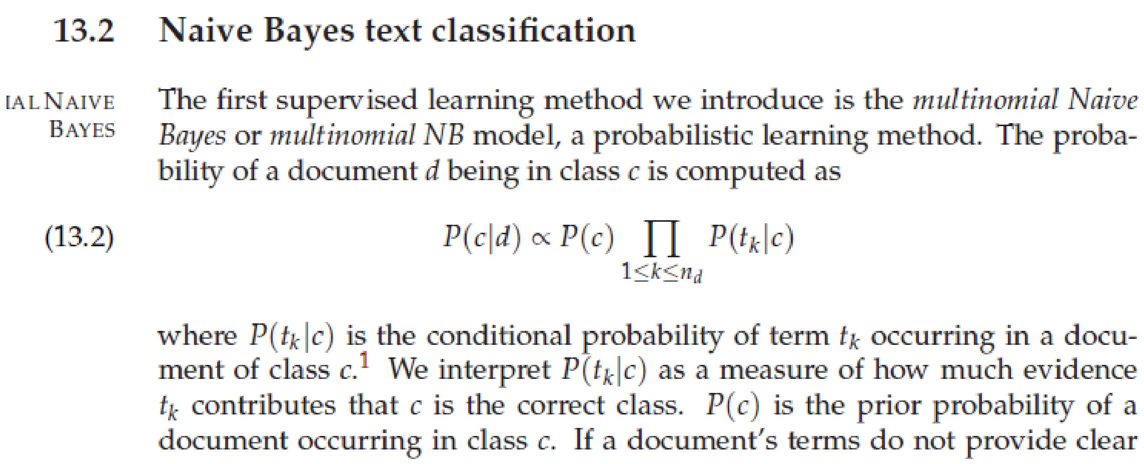
In Rocchio's algorithm, what weight setting for $\alpha/\beta/\gamma$ does a "Find pages like this one" search correspond to?

SOLUTION. "Find pages like this one" ignores the query and no negative judgments are used. Hence $\alpha = \gamma = 0$. This implies $\beta = 1$.

CHAPTER 10 (XML)

An **XML document** is an ordered, labeled tree. Each node of the tree is an XML Element and is written with an opening and closing *tag*. An element can have one or more *XML attributes*.

CHAPTER 13 (NAIVE BAYES, CLUSTERING)



13.2 Naive Bayes text classification

IAL NAIVE BAYES

The first supervised learning method we introduce is the *multinomial Naive Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document d being in class c is computed as

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{13.2}$$

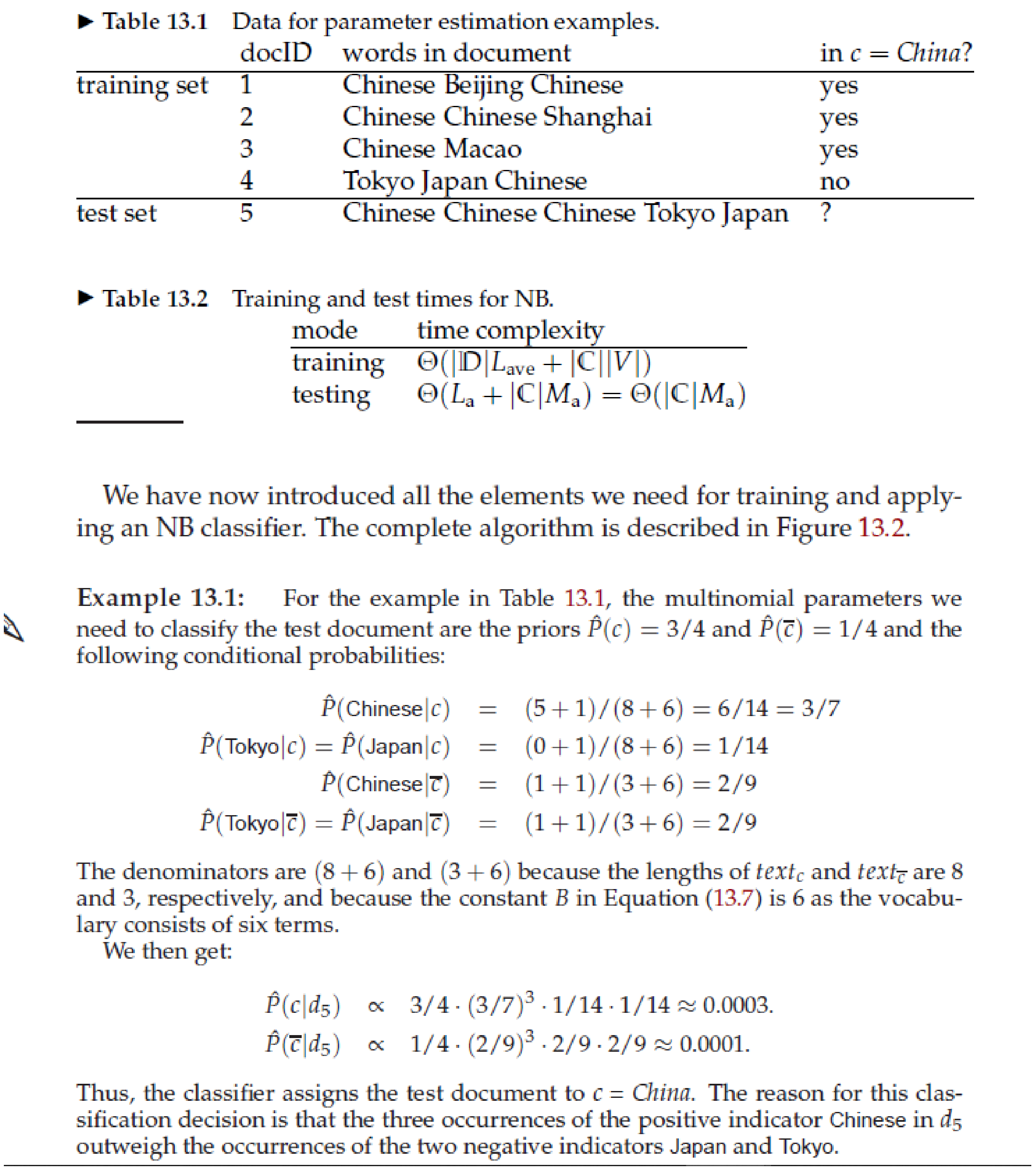
where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c.[1] We interpret $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class. $P(c)$ is the prior probability of a document occurring in class c. If a document's terms do not provide clear

- ***Hard clustering*** computes a *hard assignment* – each document is a member of exactly one cluster. The assignment of ***soft clustering*** *algorithms* is *soft* – a document's assignment is a distribution over all clusters. In a soft assignment, a document has fractional membership in several clusters.

- ***The main applications of clustering in information retrieval***:

  query expansion, document grouping, document indexing, and visualization of search results.

* QUESTION:



▶ **Table 13.1** Data for parameter estimation examples.

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

▶ **Table 13.2** Training and test times for NB.

mode	time complexity
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}} + \lvert\mathbb{C}\rvert\lvert V\rvert)$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{C}\rvert M_{\text{a}}) = \Theta(\lvert\mathbb{C}\rvert M_{\text{a}})$

We have now introduced all the elements we need for training and applying an NB classifier. The complete algorithm is described in Figure 13.2.

Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}
\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms.

We then get:

$$\begin{aligned}
\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003. \\
\hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.
\end{aligned}$$

Thus, the classifier assigns the test document to c = *China*. The reason for this classification decision is that the three occurrences of the positive indicator Chinese in d_5 outweigh the occurrences of the two negative indicators Japan and Tokyo.

CHAPTER 16 (CLUSTERING , K-MEANS)

16.4 *K*-means

CENTROID

K-means is the most important flat clustering algorithm. Its objective is to minimize the average squared Euclidean distance (Chapter 6, page 131) of documents from their cluster centers where a cluster center is defined as the mean or *centroid* $\vec{\mu}$ of the documents in a cluster ω:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

* QUESTION:

Exercise 0.194

Why are documents that do not use the same term for the concept *car* likely to end up in the same cluster in *K*-means clustering?

SOLUTION.
Documents which fall into the same cluster are similar, where the similarity is defined by the dot product of the Euclidean distance in most cases. The document with the same concept like car but not using the same term(ie car) are likely to have a lot of other common terms,similar to the term car, which will put them into the same cluster.

Exercise 0.195

Two of the possible termination conditions for *K*-means were (1) assignment does not change, (2) centroids do not change (page 361). Do these two conditions imply each other?

SOLUTION.
Let each document x(n) be assigned to a particular cluster w(k) for two consecutive interations. This implies that the assignment did not change. Also, each cluster w(k) has a fixed number of same documents belonging to it for both iterations. This implies that the centroids of clusters did not change. So termination conditions 1 and 2 imply each other.

Exercise 0.191

Define two documents as similar if they have at least two proper names like Clinton or Sarkozy in common. Give an example of an information need and two documents, for which the cluster hypothesis does *not* hold for this notion of similarity.

SOLUTION.
Document 1 is about Stanford University and Document 2 is about major search engines. Both documents contain terms Google and Yahoo. For the information need Search engine technology, document 2 is very relevant whereas document 1 is not.