Chapter III: Ranking Principles

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Wintersemester 2013/14

Chapter III: Ranking Principles

III.1 Boolean Retrieval & Document Processing
Boolean Retrieval, Tokenization, Stemming, Lemmatization

III.2 Basic Ranking & Evaluation Measures
TF*IDF, Vector Space Model, Precision/Recall, F-Measure, etc.

III.3 Probabilistic Retrieval Models
Probabilistic Ranking Principle, Binary Independence Model, BM25

III.4 Statistical Language Models
Unigram Language Models, Smoothing, Extended Language Models

III.5 Latent Topic Models
(Probabilistic) Latent Semantic Indexing, Latent Dirichlet Allocation

III.6 Advanced Query Types
Relevance Feedback, Query Expansion, Novelty & Diversity

III.1 Boolean Retrieval & Document Processing

1. **Definition of Information Retrieval**
2. **Boolean Retrieval**
3. **Document Processing**
4. **Spelling Correction and Edit Distances**

Based on **MRS Chapters 1 & 3**

Shakespeare…

- Which plays of Shakespeare mention *Brutus* and *Caesar* but not *Calpurnia*?

  (i) Get all of Shakespeare's plays from Project Gutenberg in plain text

  (ii) Use UNIX utility `grep` to determine files that match *Brutus* and *Caesar* but not *Calpurnia*

William Shakespeare

```
— bash — 61×10
grep --files-with-matches 'Brutus' * |
xargs grep --files-with-matches 'Caesar' | xargs -- files-wit
hout-match 'Calpurnia'
```

1. Definition of Information Retrieval

*Information retrieval is **finding material** (usually documents)*
*of an **unstructured nature** (usually text)*
*that satisfies an **information need***
*from within **large collections** (usually stored on computers).*

- **Finding documents** (e.g., articles, web pages, e-mails, user profiles) as opposed to creating additional data (e.g., statistics)
- **Unstructured data** (e.g., text) w/o easy-for-computer structure as opposed to structured data (e.g., relational database)
- **Information need** of a *user,* usually expressed through a *query*, needs to be satisfied which implies *effectiveness* of methods
- **Large collections** (e.g., Web, e-mails, company documents) demand *scalability & efficiency* of methods

2. Boolean Retrieval Model

- **Boolean variables** indicate presence of words in documents
- **Boolean operators** AND, OR, and NOT
- **Boolean queries** are arbitrarily complex compositions of those
  - *Brutus* AND *Caesar* AND NOT *Calpurnia*
  - NOT ((*Duncan* AND *Macbeth*) OR (*Capulet* AND *Montague*))
  - …
- **Query result** is (unordered) set of documents satisfying the query

Incidence Matrix

- Binary word-by-document matrix indicating presence of words
  - Each **column** is a binary vector: which **document** contains which words?
  - Each **row** is a binary vector: which **word** occurs in which documents?
  - To answer a Boolean query, we take the rows corresponding to the query words and apply the Boolean operators column-wise

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

Extended Boolean Retrieval Model

- Boolean retrieval used to be the standard and is still common in certain domains (e.g., library systems, patent search)
- Plain Boolean queries are too restricted
  - Queries look for words anywhere in the document
  - Words have to be exactly as specified in the query
- Extensions of the Boolean retrieval model
  - **Proximity operators** to demand that words occur close to each other (e.g., with at most *k* words or sentences between them)
  - **Wildcards** (e.g., *Ital**) for a more flexible matching
  - **Fields/Zones** (e.g., title, abstract, body) for more fine-grained matching
  - …

Boolean Ranking

- Boolean query can be satisfied by many zones of a document
- Results can be **ranked** based on how many zones satisfy query
  - **Zones** are given weights (that sum to 1)
  - **Score** is the sum of weights of those fields that satisfy the query
  - <u>Example</u>: Query *Shakespeare* in title, author, and body
    - Title with weight 0.3, author with weight 0.2, body with weight 0.5
    - Document that contains *Shakespeare* in title and body but not in title gets score 0.8

3. Document Processing

- How to convert natural language documents into an easy-for-computer format?
- Words can be simply **misspelled** or in **various forms**
  - **plural/singular** (e.g., *car*, *cars*, *foot*, *feet*, *mouse*, *mice*)
  - **tense** (e.g., *go*, *went*, *say*, *said*)
  - **adjective/adverb** (e.g., *active*, *actively*, *rapid*, *rapidly*)
  - …
- Issues and solutions are often **highly language-specific** (e.g., diacritics and inflection in German, accents in French)
- Important first step in IR

What is a Document?

- If data is not in **linear plain-text format** (e.g., ASCII, UTF-8), it needs to be converted (e.g., from PDF, Word, HTML)
- Data has to be divided into **documents** as retrievable units
  - Should the book "*Complete Works of Shakespeare*" be considered a single document? Or, should each act of each play be a document?
  - UNIX `mbox` format stores all e-mails in a single file. Separate them?
  - Should one-page-per-section HTML pages be concatenated?

Tokenization

- Tokenization splits a text into **tokens**

Two households, both alike in dignity, in fair Verona, where

Two	*households*	*both*	*alike*	*in*	*dignity*	*in*	*fair*	*Verona*	*where*

- A **type** is a class of all tokens with the same character sequence

- A **term** is a (possibly normalized) type that is included into an IR system's dictionary and thus indexed by the system

- **Basic tokenization**
  (i) Remove punctuation (e.g., commas, fullstops)
  (ii) Split at white spaces (e.g., spaces, tabulators, newlines)

Issues with Tokenization

- Language- and content-dependent
  - *Boys'* => *Boys* vs. *can't* => *can t*
  - http://www.mpi-inf.mpg.de and support@ebay.com
  - *co-ordinates* vs. *good-looking man*
  - *straight forward*, *white space*, *Los Angeles*
  - *l'ensemble* and *un ensemble*
  - Compounds: *Lebensversicherungsgesellschaftsangestellter*
  - No spaces at all (e.g., major East Asian languages)

Stopwords

- **Stopwords** are very frequent words that carry no information and are thus excluded from the system's dictionary (e.g., *a*, *the*, *and*, *are*, *as*, *be*, *by*, *for*, *from*)
- Can be defined **explicitly** (e.g., with a list) or **implicitly** (e.g., as the *k* most frequent terms in the collection)
- Do not seem to help with ranking documents
- Removing them **saves significant space** but can cause problems
  - *to be or not to be*, *the who*, etc.
  - "*president of the united states*", "*with or without you*", etc.
- Current trend towards **shorter or no stopword lists**

Stemming

- Variations of words could be grouped together (e.g., plurals, adverbial forms, verb tenses)
- A crude heuristic is to cut the ends of words (e.g., *ponies* => *poni*, *individual* => *individu*)
- **Word stem** is not necessarily a proper word
- Variations of the same word ideally map to same unique stem
- Popular stemming algorithms for English
  - Porter (http://tartarus.org/martin/PorterStemmer/)
  - Krovetz
- For English stemming has little impact on retrieval effectiveness

Porter Stemming Example

Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
From forth the fatal loins of these two foes

↓

Two household, both alik in digniti,
In fair Verona, where we lay our scene,
From ancient grudg break to new mutini,
Where civil blood make civil hand unclean.
From forth the fatal loin of these two foe

Lemmatization

- **Lemmatizer** conducts full **morphological analysis** of the word to identify the **lemma** (i.e., dictionary form) of the word
- Example: For the word *saw*, a stemmer may return *s* or *saw*, whereas a lemmatizer tries to find out whether the word is a noun (return *saw*) or a verb (return *to see*)
- For English lemmatization does not achieve considerable improvements over stemming in terms of retrieval effectiveness

Other Ideas

- **Diacritics** (e.g., ü, ø, à, ð)
  - Remove/normalize diacritics: ü => u, å => a, ø => o
  - Queries often do not include diacritics (e.g., *les miserables*)
  - Diacritics are sometimes typed using multiple characters: *für* => *fuer*
- **Lower/upper-casing**
  - Discard case information (e.g., *United States* => *united states*)
- **n-grams** as sequences of *n* characters (inter- or intra-word) are useful for Asian (CJK) languages without clear word spaces

What’s the Effect?

- Depends on the language; effect is typically limited with English
- Results for 8 European languages [Hollink et al. 2004]
  - **Diacritic removal** helped with Finnish, French, and Swedish
  - **Stemming** helped with Finnish (30% improvement) but only little with English (0-5% improvement and even less with lemmatization)
  - **Compound splitting** helped with Swedish (25%) and German (5%)
  - **Intra-word 4-grams** helped with Finnish (32%), Swedish (27%), and German (20%)
- Larger benefits for morphologically richer languages

Zipf's Law (after George Kingsley Zipf)

- The **collection frequency** cf_i of the i-th most frequent word in the document collection is **inversely proportional** to the rank i

$$cf_i \propto \frac{1}{i}$$

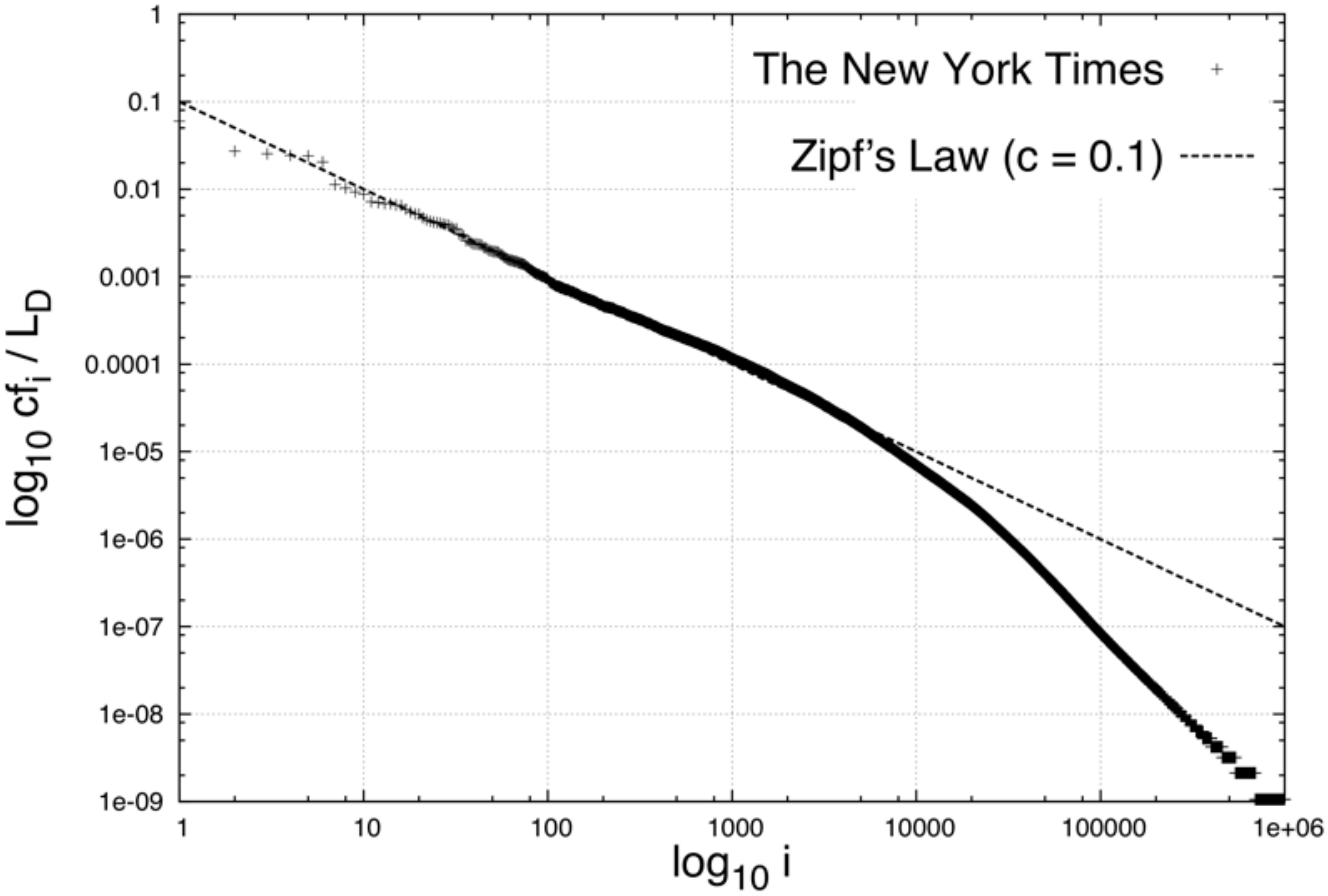
- For the relative collection frequency with **language-specific constant** c (for English $c \approx 0.1$) we obtain

$$\frac{cf_i}{\sum_j cf_j} \propto \frac{c}{i}$$

- In an English document collection,
we can thus expect the most frequent word
to account for 10% of all term occurrences

George Kingsley Zipf

Zipf's Law (cont'd)



Heaps' Law (after Harold Stanley Heaps)

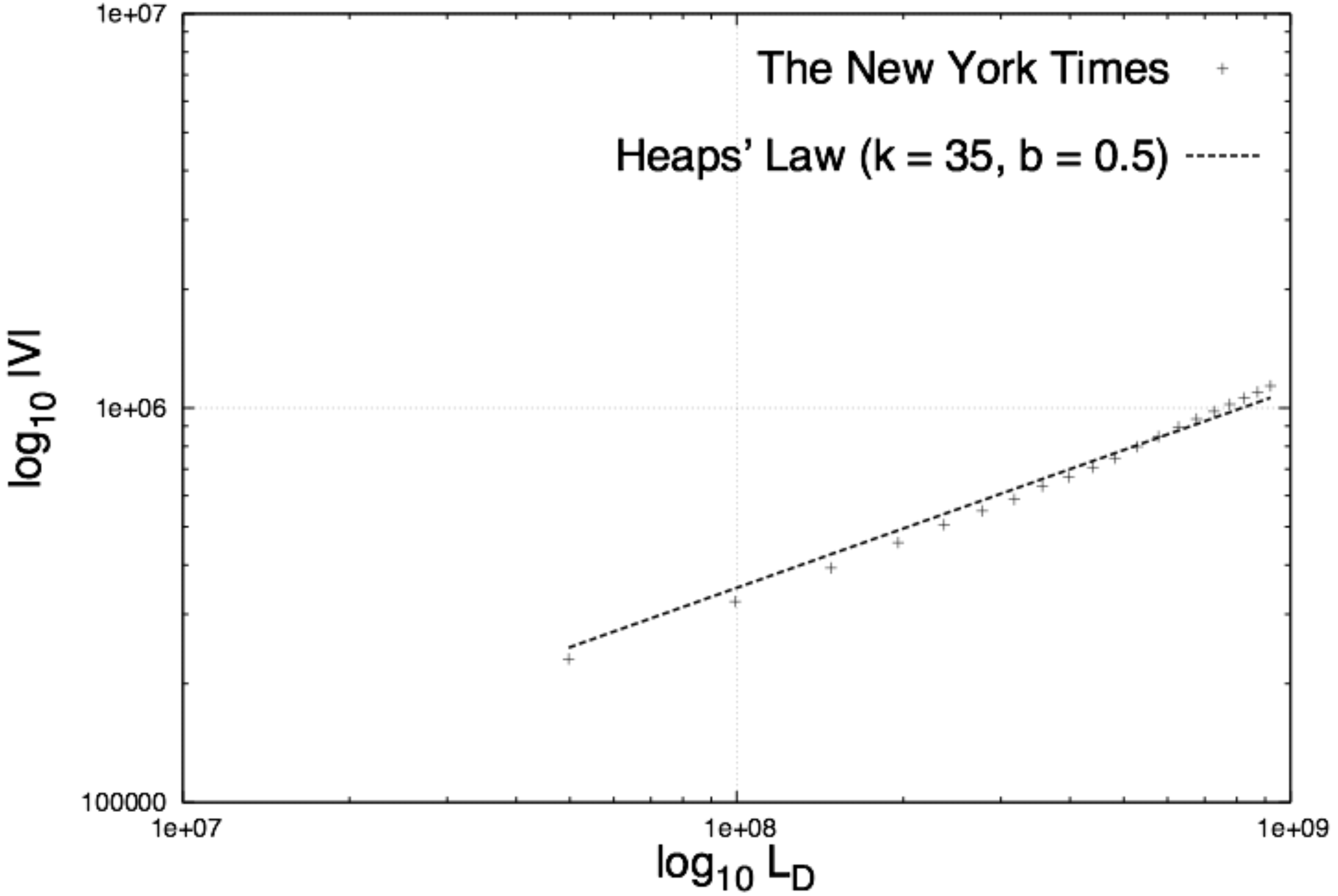
- The **number of distinct words** $|V|$ in a document collection (i.e., the size of the vocabulary) relates to the **total number of word occurrences** as

$$|V| \propto k \left(\sum_{v \in V} cf(v) \right)^b$$

  with **collection-specific constants** k and b

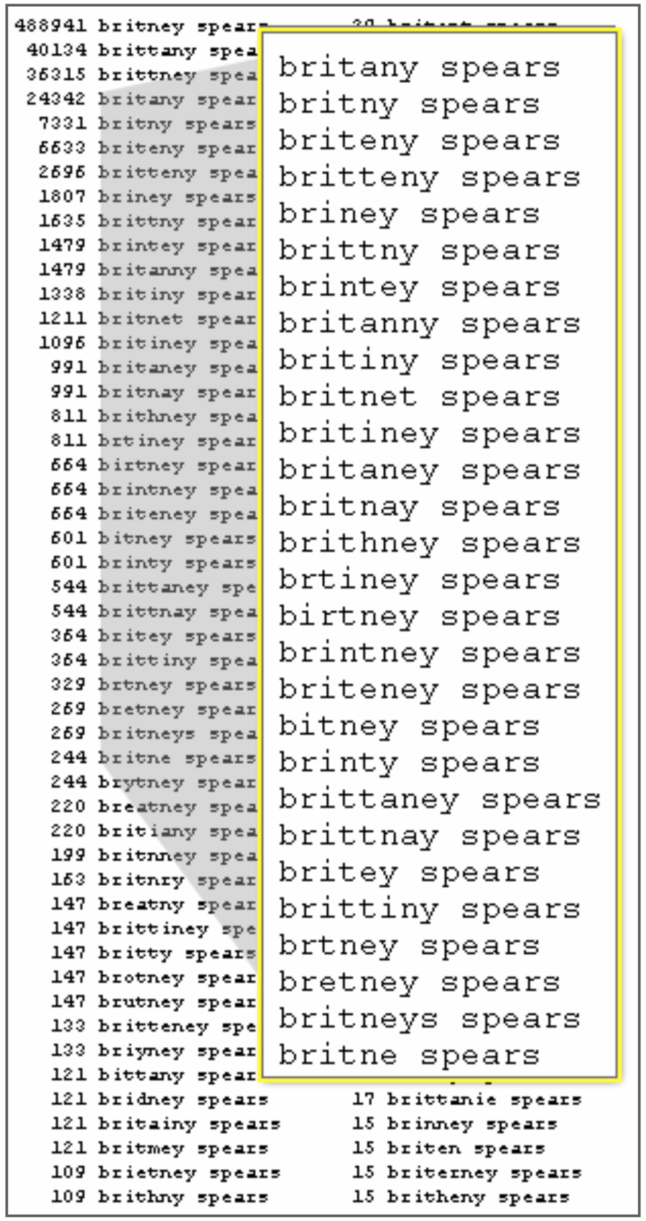
- We can thus expect the **size of the vocabulary** to **grow with** the **size of the document collection** – with ramifications on the implementation of IR systems

Heaps' Law (cont'd)



4. Spelling Correction and Edit Distances

- Users don't know how to spell!
- When the user types in an unknown, potentially misspelled word, we can try to map it to the "closest" term in our dictionary
- We need a notion of **distance between terms**
  - adding extra character (e.g., *hoouse* vs. *house*)
  - omitting character (e.g., *huse*)
  - using wrong character (e.g., *hiuse*)
  - as-heard spelling (e.g., *kwia* vs. *choir*)



Amit Singhal: SIGIR '05 Keynote

Hamming Edit Distance

- Distances should satisfy **triangle inequality**
  - $d(x, z) \leqq d(x,y) + d(y, z)$ for strings x, y, z and distance d
- **Hamming edit distance** is the number of positions at which the two strings x and y are different
- Strings of **different lengths** are compared by padding the shorter one with null characters (e.g., *house* vs. *hot* => *house* vs. hot _ _)
- Hamming edit distance counts **wrong characters**
- <u>Examples</u>:
  - $d(car, cat) = 1$
  - $d(house, hot) = 3$
  - $d(house, hoouse) = 4$

Longest Common Subsequence

- A **subsequence** of two strings x and y is a string s such that all characters from s occur in x and y in the same order but not necessarily contiguously

- **Longest common subsequence (LCS) distance** defined as

$$d(x,y) = max(|x|,|y|) - \max_{s \in S(x,y)} |s|$$

  with $S(x, y)$ as the set of all subsequences of x and y and string lengths $|x|$, $|y|$, and $|s|$

- LCS distance counts **omitted characters**

- Examples:

  - $d(house, huse) = 1$

  - $d(banana, atana) = 2$

Levenshtein Edit Distance

- **Levenshtein edit distance** between two strings x and y is the minimal number of edit operations (*insert*, *replace*, *delete*) required to transform x into y

- The minimal number of operations $m[i, j]$ to transform the **prefix substring** $x[1{:}i]$ into $y[1{:}j]$ is defined via the **recurrence**

$$m[i,j] = \min \begin{cases} m[i-1, j-1] & + \;\; (x[i] = y[j] \;?\; 0 \;:\; 1) & \text{(replace } x[i]\text{?)} \\ m[i-1, j] & + \;\; 1 & \text{(delete } x[i]\text{)} \\ m[i, j-1] & + \;\; 1 & \text{(insert } y[j]\text{)} \end{cases}$$

- Examples:
  - $d(hoouse, house) = 1$
  - $d(house, rose) = 2$
  - $d(house, hot) = 3$

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \; + \; 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \; + \; 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- Example: *cat* vs. *kate*

$$
m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \,?\, 0 \;:\; 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \quad\quad\;\; + \; 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \quad\quad\;\; + \; 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}
$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$
m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \ ?\ 0 \ : \ 1) & \text{(replace } x[i]?\text{)} \ \nwarrow \\ m[i-1, j] \qquad\quad + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \qquad\quad + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}
$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1, j] \ + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \ + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1,j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1,j] \qquad\qquad + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i,j-1] \qquad\qquad + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- Example: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \;+\; (x[i] = y[j]\ ?\ 0 \ :\ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\quad +\; 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\quad +\; 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \ ? \ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\qquad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\qquad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\qquad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\qquad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \left\{ \begin{array}{llll} m[i-1, j-1] & + \ (x[i] = y[j] \ ? \ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} & \nwarrow \\ m[i-1, j] & + \ 1 & \text{(delete } x[i]\text{)} & \uparrow \\ m[i, j-1] & + \ 1 & \text{(insert } y[j]\text{)} & \leftarrow \end{array} \right.$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \, ? \, 0 \ : \ 1) & (\text{replace } x[i]?) \ \nwarrow \\ m[i-1, j] \qquad\quad + \ 1 & (\text{delete } x[i]) \ \uparrow \\ m[i, j-1] \qquad\quad + \ 1 & (\text{insert } y[j]) \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1, j] \qquad\quad + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \qquad\quad + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1, j] \ + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \ + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont’d)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$
m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1, j] \qquad\qquad + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \qquad\qquad + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}
$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- Example: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \ ? \ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\qquad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\qquad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \ ? \ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\quad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\quad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \mathbin{?} 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \ \nwarrow \\ m[i-1, j] \qquad\quad + \ 1 & \text{(delete } x[i]\text{)} \ \uparrow \\ m[i, j-1] \qquad\quad + \ 1 & \text{(insert } y[j]\text{)} \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\,|y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] + (x[i] = y[j]\ ?\ 0 : 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] + 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] + 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- Example: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j] \ ? \ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \quad\quad\quad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \quad\quad\quad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & (\text{replace } x[i]?) \ \nwarrow \\ m[i-1, j] \qquad + \ 1 & (\text{delete } x[i]) \ \uparrow \\ m[i, j-1] \qquad + \ 1 & (\text{insert } y[j]) \ \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings x and y corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \; ? \; 0 \; : \; 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \qquad\quad + \; 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \qquad\quad + \; 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \; ? \; 0 \; : \; 1) & \text{(replace } x[i]\text{?)} \; \nwarrow \\ m[i-1, j] \; + \; 1 & \text{(delete } x[i]\text{)} \; \uparrow \\ m[i, j-1] \; + \; 1 & \text{(insert } y[j]\text{)} \; \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] + (x[i] = y[j]\ ?\ 0 : 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] + 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] + 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	*k*	*a*	*t*	*e*
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- <u>Example</u>: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \; ? \; 0 \; : \; 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \; + \; 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \; + \; 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\ |y|)$

- Example: *cat* vs. *kate*

$$m[i,j] = \min \begin{cases} m[i-1, j-1] \ + \ (x[i] = y[j]\ ?\ 0 \ : \ 1) & \text{(replace } x[i]\text{?)} \quad \nwarrow \\ m[i-1, j] \quad + \ 1 & \text{(delete } x[i]\text{)} \quad \uparrow \\ m[i, j-1] \quad + \ 1 & \text{(insert } y[j]\text{)} \quad \leftarrow \end{cases}$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	2

Levenshtein Edit Distance (cont'd)

- Levenshtein edit distance between two strings *x* and *y* corresponds to $m[|x|, |y|]$ and can be computed using **dynamic programming** in time and space $O(|x|\,|y|)$

- <u>Example</u>: *cat* vs. *kate*

$$
m[i,j] = \min \begin{cases} m[i-1, j-1] \; + \; (x[i] = y[j] \,?\, 0 \,:\, 1) & (\text{replace } x[i]?) \; \nwarrow \\ m[i-1, j] \; + \; 1 & (\text{delete } x[i]) \; \uparrow \\ m[i, j-1] \; + \; 1 & (\text{insert } y[j]) \; \leftarrow \end{cases}
$$

- Edit operations can be weighted (e.g., based on letter frequency)

	_	***k***	***a***	***t***	***e***
_	0	1	2	3	4
c	1	1	2	3	4
a	2	2	1	2	3
t	3	3	2	1	**2**

Soundex

- **Soundex algorithm** tries to map **homophones** (i.e., words with same pronunciation) to canonical representation based on rules
  - Keep the first letter
  - Replace letters *A*, *E*, *I*, *O*, *U*, *H*, *W*, and *Y* by number *0*
  - Replace letters *B*, *F*, and *P* by number *1*
  - Replace letters *C*, *G*, *J*, *K*, *Q*, *S*, *X*, and *Z* by number *2*
  - Replace letters *D* and *T* by number *3*
  - Replace letter *L* by number *4*
  - Replace letters *M* and *N* by number *5*
  - Replace letter *R* by number *6*
  - Coalesce sequences of the same number (e.g., *33311* => *31*)
  - Remove all *0s* and append *000*
  - Keep the first four characters as canonical representation

Soundex (Examples)

- <u>Examples</u>:
  - ***lightening*** *=> L0g0t0n0ng => L020t0n0n2 => L02030n0n2 => L020305052 => L23552000 =>* ***L235***
  - ***lightning*** *=> L0g0tn0ng => L020tn0n2 => L0203n0n2 => L02035052 => L23552000 =>* ***L235***

Summary of III.1

- **Boolean retrieval**
  supports precise-yet-limited querying of document collections
- **Stemming and lemmatization**
  to deal with syntactic diversity (e.g., inflection, plural/singular)
- **Zipf's law**
  about the frequency distribution of terms
- **Heaps' law**
  about the number of distinct terms
- **Edit distances**
  to handle spelling errors and allow for vagueness

Additional Literature for III.1

- **V. Hollink, J. Kamps, C. Monz, and M. de Rijke**: *Monolingual document retrieval for European languages*, IR 7(1):33-52, 2004

- **A. Singhal**: *Challenges in running a commercial search engine*, SIGIR 2005

III.2 Basic Ranking & Evaluation

1. **Vector Space Model**
2. **TF*IDF**
3. **IR Evaluation**

Based on **MRS Chapters 6 & 8**

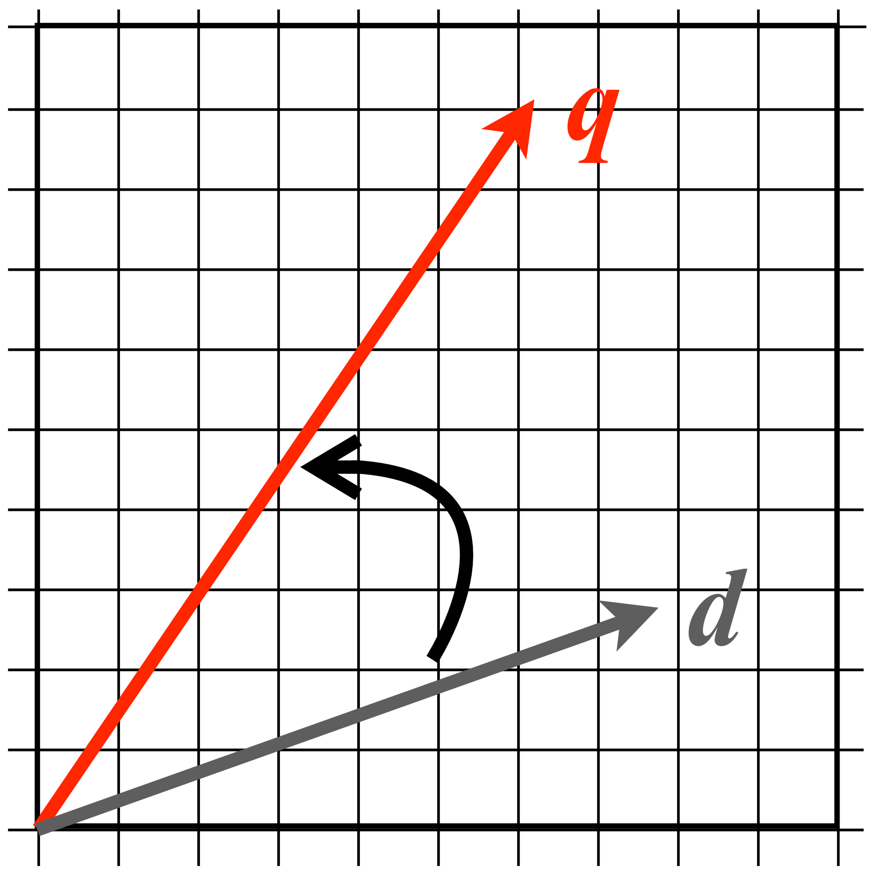
1. Vector Space Model (VSM)

- Boolean retrieval model provides **no (or only rudimentary) ranking of results** – severe limitation for large result sets
- Vector space model views **documents and queries as vectors** in a $|V|$-dimensional vector space (i.e., one dimension per term)
- **Cosine similarity** between two vectors ***q*** and ***d*** is the cosine of the angle between them

$$sim(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \, \|\mathbf{d}\|}$$

$$= \frac{\sum_{i=1}^{|V|} \mathbf{q}_i \, \mathbf{d}_i}{\sqrt{\sum_{i=1}^{|V|} \mathbf{q}_i^2} \sqrt{\sum_{i=1}^{|V|} \mathbf{d}_i^2}}$$

$$= \frac{\mathbf{q}}{\|\mathbf{q}\|} \frac{\mathbf{d}}{\|\mathbf{d}\|}$$



2. TF*IDF

- How to set the vector components $\mathbf{d}_t$ and $\mathbf{q}_t$?
- Incidence matrix from Boolean retrieval model suggests 0/1
  - documents should be favored if they **contain a query term often**
  - **query terms should be weighted** (e.g., *edward snowden movie*)
- **Term frequency** $tf_{t,d}$ as
  the number of times the term t occurs in document d
- **Document frequency** df_t as
  the number of documents that contain the term t
- **Inverse document frequency** idf_t as

$$idf_t = \frac{|D|}{df_t}$$

  with $|D|$ as the **number of documents** in the collection

TF*IDF (cont'd)

- The tf.idf weight of term *t* in document *d* is then defined as

$$tf.idf_{t,d} = tf_{t,d} \times idf_t$$

- The weight $tf.idf_{t,d}$ is…
  - **larger** when *t* occurs **often in d** and/or **not in many documents**
  - **smaller** when *t* occurs **not often in d** and/or **in many documents**
- When using the VSM, we can set the vector components as

$$\mathbf{d}_t = tf.idf_{t,d} \qquad \mathbf{q}_t = tf.idf_{t,q}$$

- Slightly simpler scoring of documents for query **q** as

$$score(\mathbf{q}, \mathbf{d}) = \sum_{t \in \mathbf{q}} tf.idf_{t,d}$$

Dampening, Length Normalization, etc.

- **Many variations** of the basic TF*IDF weighting scheme exist
  - **Logarithmic dampening** of inverse document frequency

$$idf_t = log \frac{|D|}{df_t}$$

  avoids putting too much weight on **exotic terms**

  - **Sublinear scaling** of term frequency

$$wtf_{t,d} = \begin{cases} 1 + log\ tf_{t,d} & : \quad tf_{t,d} > 0 \\ 0 & : \quad \text{otherwise} \end{cases}$$

  - **Length normalization** and **max-tf normalization**

$$rtf_{t,d} = \frac{tf_{t,d}}{\sum_{v \in \mathbf{d}} tf_{v,d}} \qquad ntf_{t,d} = \frac{tf_{t,d}}{\max_{v \in \mathbf{d}} tf_{v,d}}$$

  avoids favoring **long documents**

3. IR Evaluation

- How to **systematically evaluate/compare** different IR methods
  - which variant of TF*IDF performs best?
  - does stemming help? How about stopword removal?
- We need a **document collection**, a set of **topics** and **relevance assessments**, and **effectiveness measures**
- IR evaluation has been driven a lot by **benchmark initiatives**
  - **TREC** (http://trec.nist.gov) – diverse & changing tasks
  - **CLEF** (http://www.clef-initiative.eu) – original focus: cross-lingual IR
  - **NTCIR** (http://research.nii.ac.jp/ntcir) – original focus: Asian languages
  - **INEX** (https://inex.mmci.uni-saarland.de) – original focus: XML-IR

Documents, Topics, and Relevance Assessments

- **Document collection** (e.g., a collection of newspaper articles)
- **Topics** are descriptions of concrete **information needs**

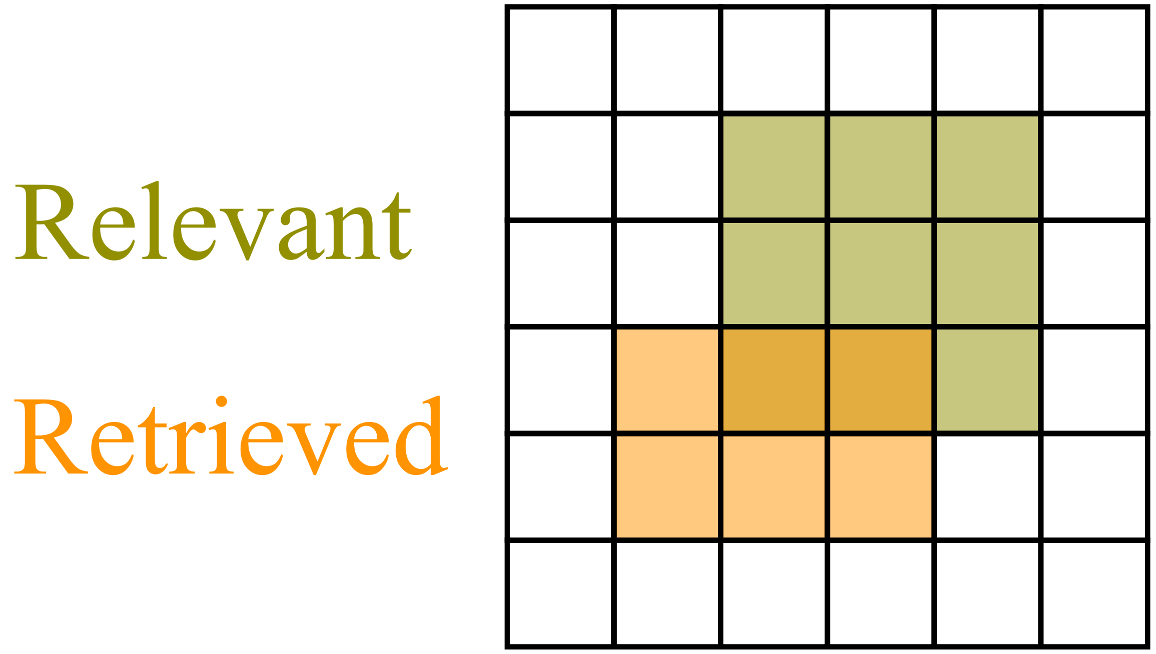
```
<num> Number: 310
<title> Radio Waves and Brain Cancer

<desc> Description:
Evidence that radio waves from radio towers or car phones affect
brain cancer occurrence.

<narr> Narrative:
Persons living near radio towers and more recently persons using
car phones have been diagnosed with brain cancer.  The argument
rages regarding the direct association of one with the other.
The incidence of cancer among the groups cited is considered…
```

- **Queries** are derived from topics (e.g., using only the title)
- **Relevance assessments** are (*topic*, *document*, *label*) tuples with binary (1 : relevant, 0 : irrelevant) or graded labels often determined by trained experts
- **Parameter tuning** mandates splitting into training & test topics

Classifying the Results

- We can classify documents for a given information need as

	relevant	**irrelevant**
retrieved	true positives (**tp**)	false positives (**fp**)
not retrieved	false negatives (**fn**)	true negatives (**tn**)



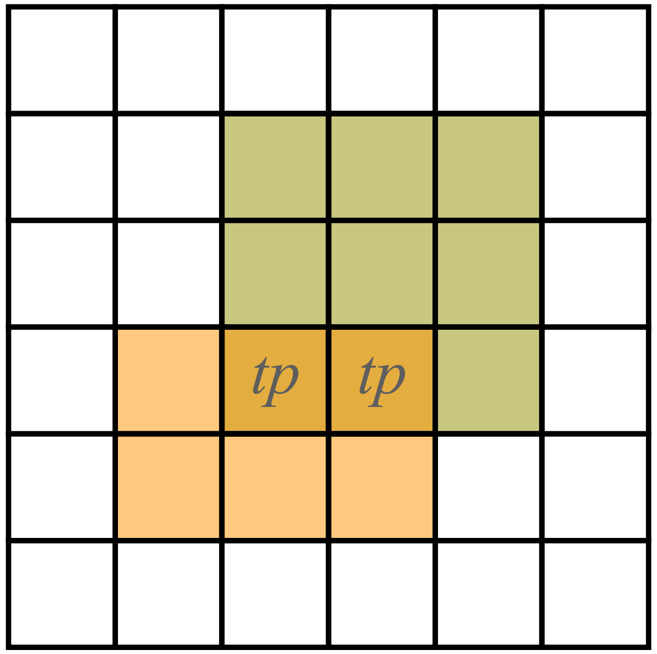
Classifying the Results

- We can classify documents for a given information need as

	relevant	**irrelevant**
retrieved	true positives (**tp**)	false positives (**fp**)
not retrieved	false negatives (**fn**)	true negatives (**tn**)

Relevant

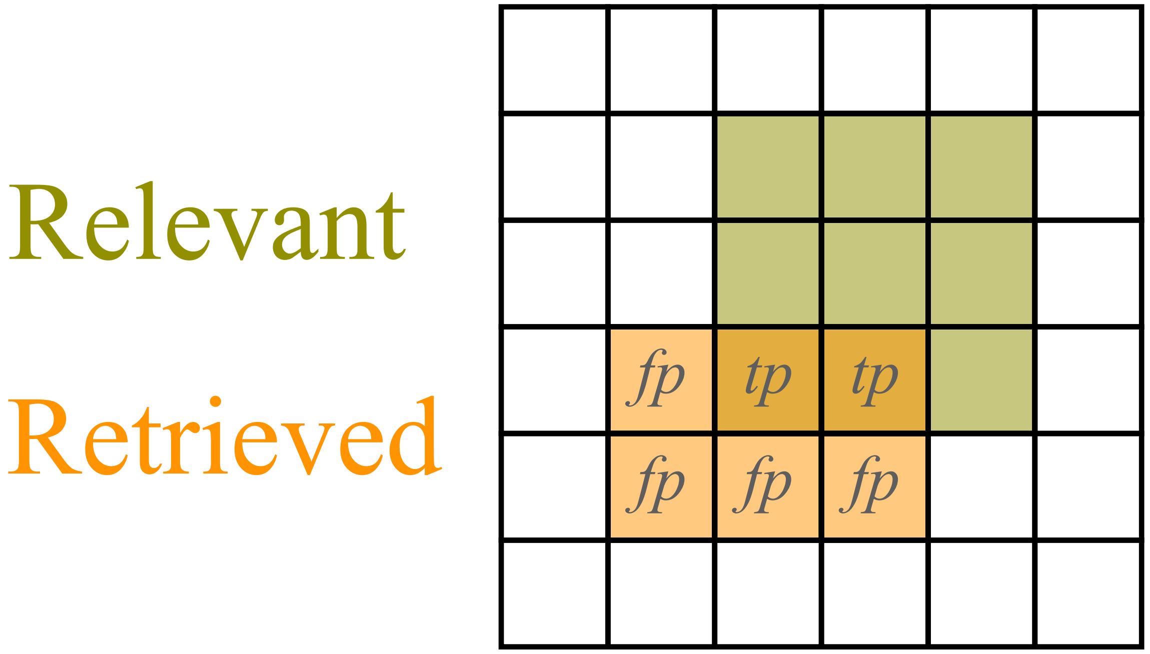
Retrieved



Classifying the Results

- We can classify documents for a given information need as

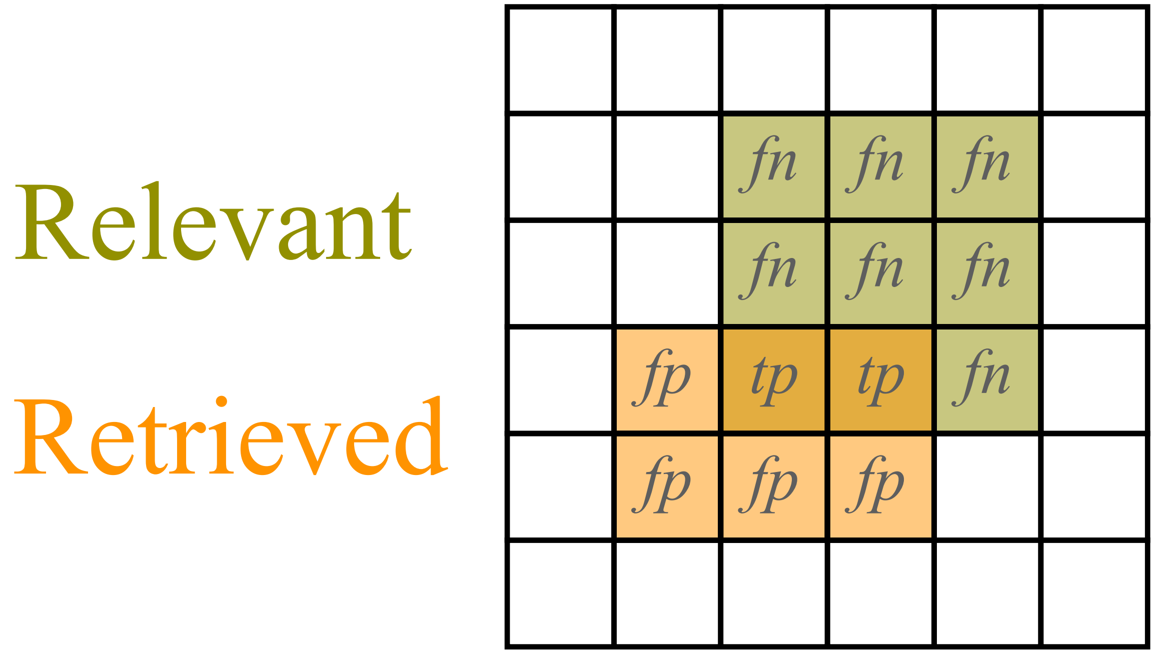
	<u>**relevant**</u>	<u>**irrelevant**</u>
<u>**retrieved**</u>	true positives (**tp**)	false positives (**fp**)
<u>**not retrieved**</u>	false negatives (**fn**)	true negatives (**tn**)



Classifying the Results

- We can classify documents for a given information need as

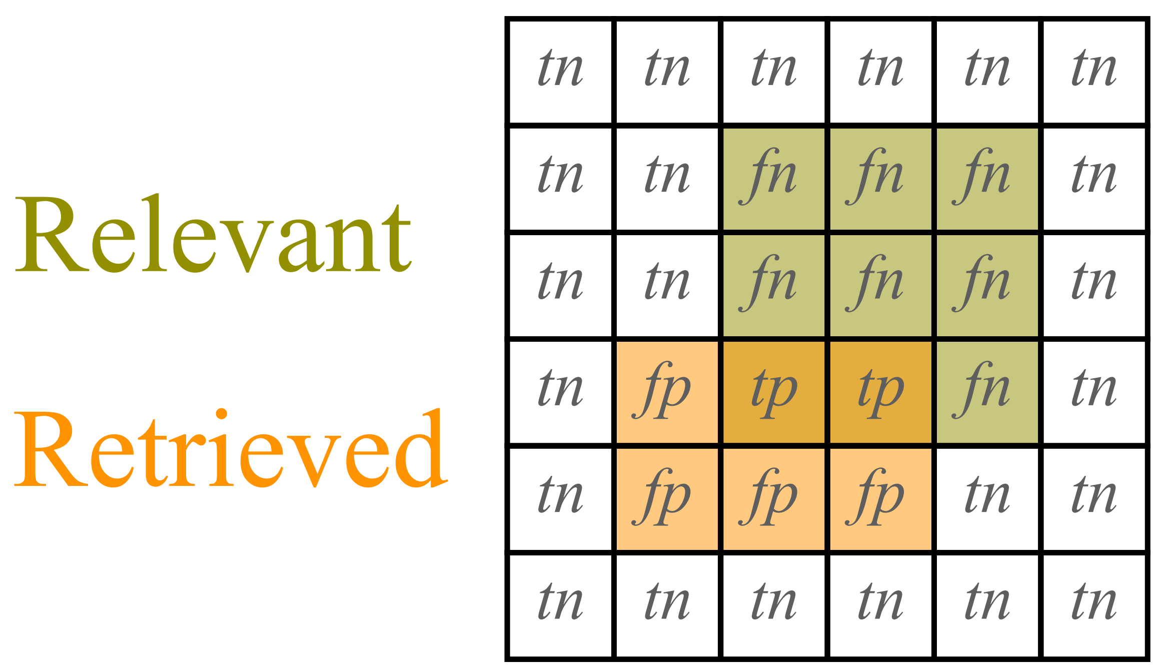
	relevant	**irrelevant**
retrieved	true positives (**tp**)	false positives (**fp**)
not retrieved	false negatives (**fn**)	true negatives (**tn**)



Classifying the Results

- We can classify documents for a given information need as

	relevant	**irrelevant**
retrieved	true positives (**tp**)	false positives (**fp**)
not retrieved	false negatives (**fn**)	true negatives (**tn**)



Precision, Recall, and Accuracy

- **Precision** *P* is the fraction of retrieved documents that is relevant

$$P = \frac{tp}{tp + fp}$$

- **Recall** *R* is the fraction of relevant results that is retrieved

$$R = \frac{tp}{tp + fn}$$

- **Accuracy** *A* is the fraction of correctly classified documents

$$A = \frac{tp + tn}{tp + fp + tn + fn}$$

Precision, Recall, and Accuracy

- **Precision** P is the fraction of retrieved documents that is relevant

$$P = \frac{tp}{tp + fp}$$

- **Recall** R is the fraction of relevant results that is retrieved

$$R = \frac{tp}{tp + fn}$$

- **Accuracy** A is the fraction of correctly classified documents

$$A = \frac{tp + tn}{tp + fp + tn + fn}$$

Not appropriate for IR

F-Measure

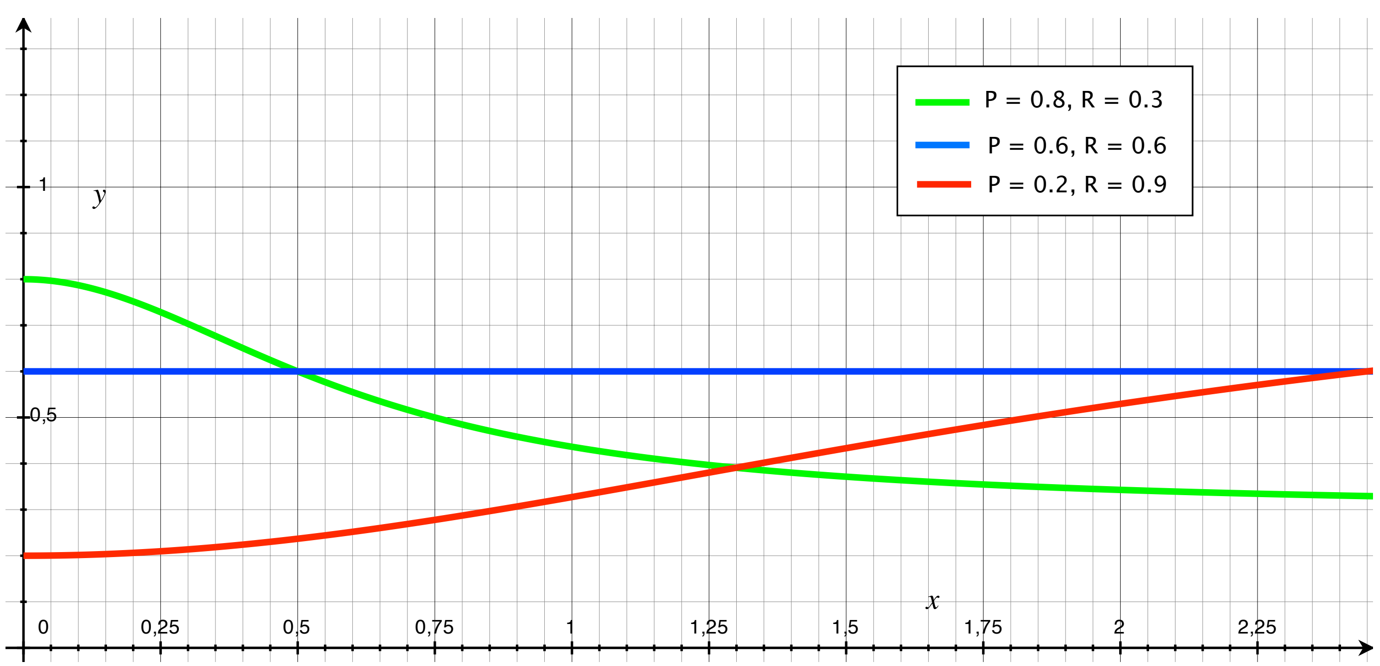
- Some tasks focus on **precision** (e.g., web search), others only on **recall** (e.g., library search), but usually a **balance** between the two is sought

- **F-measure** combines precision and recall in a single measure

$$F_\beta = \frac{(\beta^2 + 1)\, P\, R}{\beta^2\, P + R}$$

with β as **trade-off parameter**

- $\beta = 1$ is balanced
- $\beta < 1$ emphasizes precision
- $\beta > 1$ emphasizes recall



(Mean) Average Precision

- Precision, recall, and F-measure ignore the order of results
- **Average precision** (AP) averages over retrieved relevant results
  - Let $\{d_1, \ldots, d_{mj}\}$ be the set of relevant results for the query q_j
  - Let R_{jk} be the set of ranked retrieval results for the query q_j from top until you get to the relevant result d_k

$$\mathrm{AP}(q_j) = \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk})$$

- **Mean average precision** (MAP) averages over multiple queries

$$\mathrm{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \mathrm{AP}(q_j)$$

Precision@*k*

- It is unrealistic to assume that users inspect the entire query result
- Often (e.g., in web search) users would only look at **top-*k* results**
- **Precision@*k*** (P@*k*) is the precision achieved by the top-*k* results
- **Typical values** of *k* are 5, 10, 20

(Normalized) Discounted Cumulative Gain

- What if we have **graded labels** as relevance assessments? (e.g., 0 : not relevant, 1 : marginally relevant, 2 : relevant)
- **Discounted cumulative gain** (DCG) for query *q*

$$DCG(q,k) = \sum_{m=1}^{k} \frac{2^{R(q,m)} - 1}{log(1+m)}$$

  with $R(q, m) \in \{0, \ldots, 2\}$ as label of m-th retrieved result

- **Normalized discounted cumulative gain** (NDCG)

$$NDCG(q,k) = \frac{DCG(q,k)}{IDCG(q,k)}$$

  normalized by **idealized discounted cumulative gain** (IDCG)

(Normalized) Discounted Cumulative Gain (cont'd)

- $IDCG(q, k)$ is the **best-possible** value $DCG(q, k)$ achievable for the query q on the document collection at hand
- Example: Let $R(q, m) \in \{0, \ldots, 2\}$ and assume that two documents have been labeled with 2, two with 1, all others with 0. The best-possible top-5 result thus has labels <2, 2, 1, 1, 0> and determines the value of $IDCG(q, k)$ for this query
- NDCG **also considers rank** at which relevant results are retrieved
- NDCG is typically average over **multiple queries**

$$NDCG(Q, k) = \frac{1}{|Q|} \sum_{q \in Q} NDCG(q, k)$$

Summary of III.2

- **Vector space model**
  maps queries and documents into a common vector space
- **Cosine similarity**
  to compare query vectors and document vectors
- **TF*IDF**
  weights terms based on term frequency and document frequency
- **Documents, queries, relevance assessments**
  as essential building blocks of IR evaluation
- **Effectiveness measures** (Precision, Recall, MAP, nDCG, etc.)
  assess the quality of results taking into account order, labels, etc.