Building an inverted index

Stanislav Protasov

Agenda

1. Languages
    a. Formal approach
    b. Tokenization
    c. Stemming and lemmatization
2. Building an inverted index

Languages

Languages: syntax, semantics, pragmatics

- **Pragmatics:** `new_var = map(lambda x: x - 2, [4, 5, 6])`
- **Semantics:**
    - This is a valid sentence in English.
    - The worst part and clumsy looking for whoever heard light.
    - Twas brillig, and the slithy toves did gyre and gimble in the wabe.
    - Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor …
- **Grammar (syntax):**
    - I can has cheezburger?
    - I nevr mkae tipos and erors in my sentencs.
    - I’m chuffed to bits seeing you! Do ya wanna watch some telly together, bro?
    - I’ll txt w/my ETA 2U.

Questions to discuss?

Where does semantics hide in the language?

What is the purpose to have a language syntax?

Syntax

Definitions

syntax guards word order — initially linguistic term

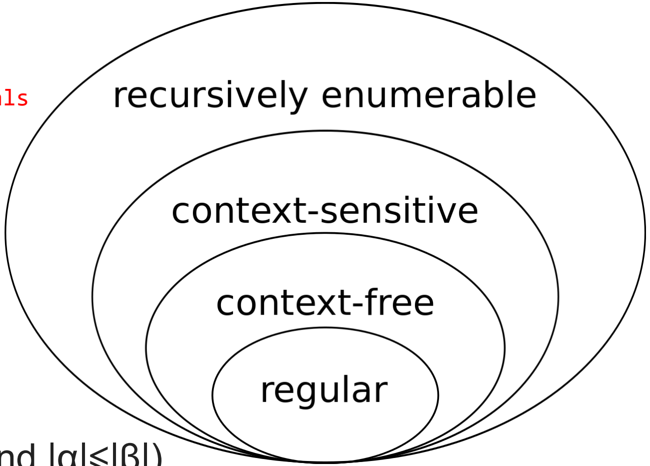
[formal] **grammar** - describes *how to* form strings from a language's *alphabet* that are valid according to the language's *syntax*. = set of **rules**, way to express syntax

formal language - *set* of all strings *allowed* by a grammar. = **satisfy** rules

Grammar is a <**Σ** - terminals, **N** - nonterminals, **P** - productions, **S** - start symbol>

Chomsky Normal Form (CNF) and Chomsky hierarchy

ABC – nonterminals
abc – terminals
αβ – seq of terminals/non-terminals
γ – non-empty seq



0. **Recursively enumerable** (almost any productions)

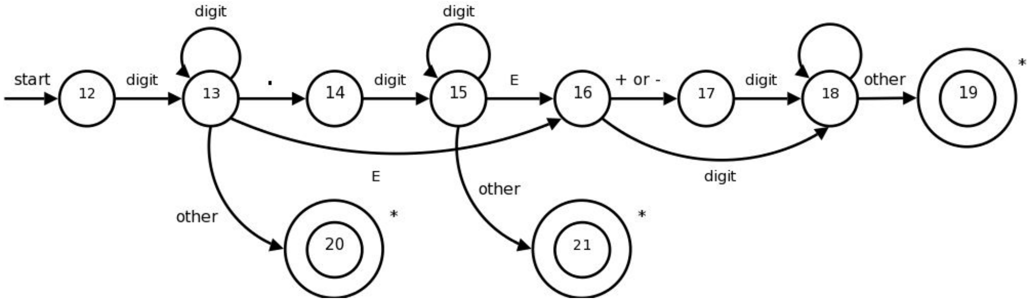
1. **Context-sensitive** $\alpha A\beta \rightarrow \alpha\gamma\beta$

    Also noncontracting grammar ($\alpha \rightarrow \beta$, where $\alpha,\beta \in \{\Sigma \cup N\}^+$ and $|\alpha| \leqslant |\beta|$)

    E.g. **Professor** (α) **Brown** (proper noun) vs **Brown** (adjective) **bear** (β)

2. **Context-free** $A \rightarrow \alpha$

3. **Regular** A → a or A → aB | A → Ba (exceptionally)



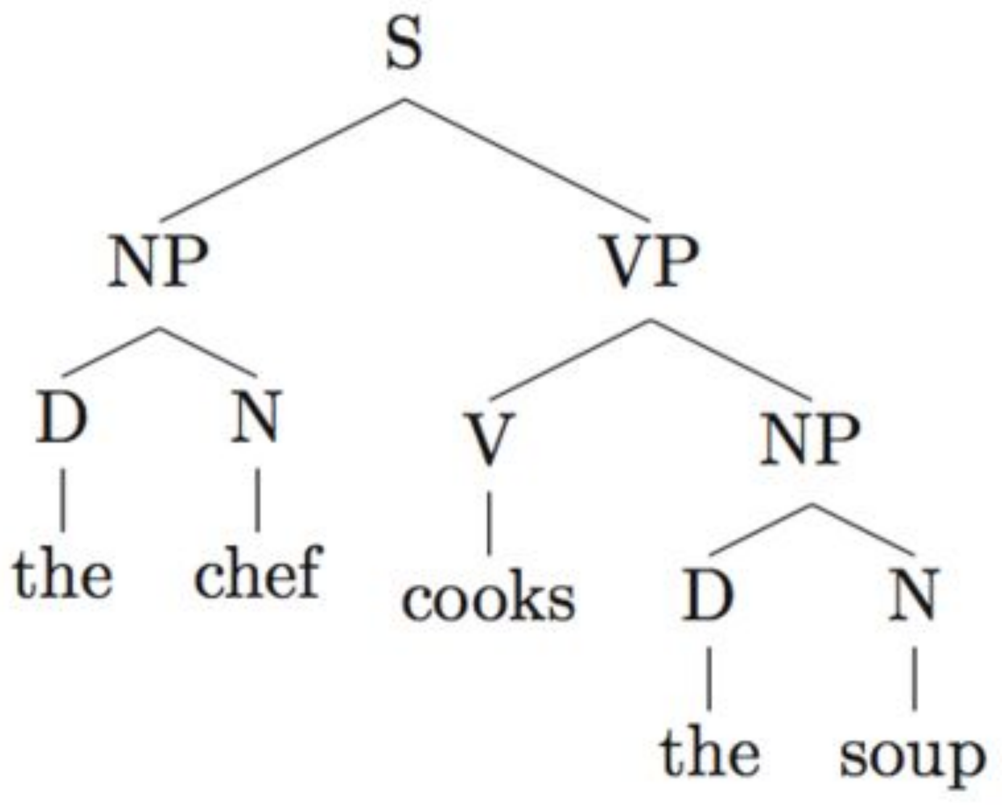
— e.g. numbers.

Extended Backus-Naur Form (EBNF)

```
A = B,C.            # concat

A = B|C|D.          # one of

A = [B].            # 0/1

A = {B}.            # 0+

A = B{B}.           # 1+

A = (B|C)(D|E).     # grouping (to avoid service NT)
```

Syntax tree

```
D  =  the | a

N  = chef | soup

V  = cooks

NP  = D N | ...

VP  = V N | V NP

S  = NP VP
```



<**Σ** - terminals, **N** - nonterminals, **P** - productions, **S** - start symbol>

Tokenization

Lexemes (distinct objects of the language) are produced by **scanner**.

`token = (lexeme, token_type ~ PoS)`

Program, converting stream of characters into a stream of **tokens** is called **lexical analyzer, lexer, tokenizer**.

```
i like to read science fiction.
[('i', 'PRP'), ('like', 'VBP'), ('to', 'TO'),
('read', 'VB'), ('science', 'NN'), ('fiction'
, 'NN'), ('.', '.')]
```

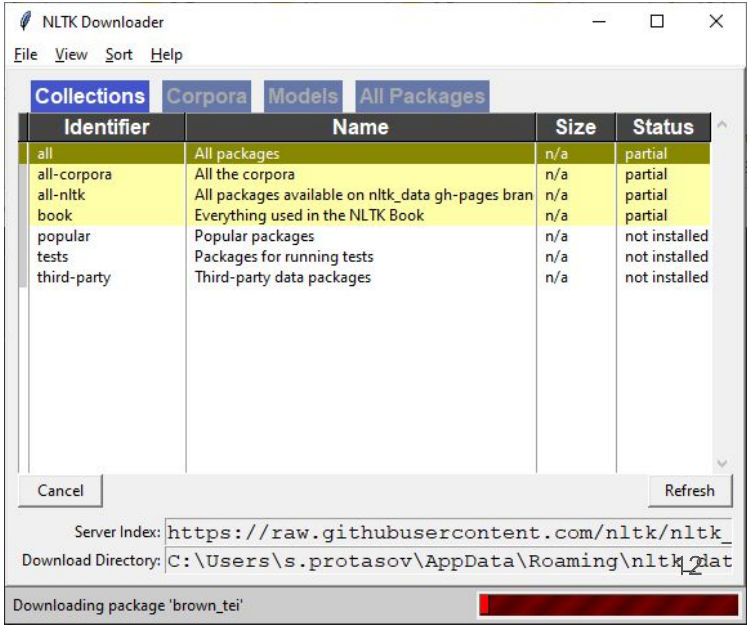
Syntax analysis helps proper tokenization

L'ensemble □ one token or two? ***L*** ? ***L’*** ? ***Le*** ?

莎拉波娃现在居住在美国东南部的佛罗里达

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

```
# language-specific punctuation
import nltk
nltk.download()
st = nltk.data.load('tokenizers/punkt/english.pickle')
st.tokenize(text)     # sentence splitting

nltk.download('punkt')
nltk.word_tokenize()  # language specific

# grammar based tokenization
simple_grammar = nltk.parse_cfg(...)
parser = nltk.ChartParser(simple_grammar)
trees = parser.nbest_parse("A car has a door")
```



… or use statistics from corpora!

This is a sentence that we will use to test the magic tool

*nltk.**UnigramTagger**(brown_tagged_sents).tag(tokens)*

```
[('This', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('sentence', 'NOUN'), ('that',
'ADP'), ('we', 'PRON'), ('will', 'VERB'), ('use', 'NOUN'), ('to', 'PRT'),
('test', 'NOUN'), ('the', 'DET'), ('magic', 'ADJ'), ('tool', 'NOUN')]
```

*nltk.**BigramTagger**(brown_tagged_sents).tag(tokens)*

```
[('This', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('sentence', 'NOUN'), ('that',
'PRON'), ('we', 'PRON'), ('will', 'VERB'), ('use', 'VERB'), ('to', 'PRT'),
('test', 'VERB'), ('the', 'DET'), ('magic', 'ADJ'), ('tool', 'NOUN')]
```

We are almost ready to build an index of tokens. Anything left?

Compression techniques used across methods

- **case folding**: London = london; Лев = лев
- **stemmer vs lemmer**:
  stemming: *compress* = *compress*ion = un*compress*ed
  бегу = бег
  lemmatization: better = good
  бегу = бегать
- ignore **stop words**: to, the, it, be, or, ...
  - Problems arise when search on “To be or not to be” or “the month of May”
- **Thesaurus**: fast = rapid; лев = лёвушка
  - handbuilt clustering

Stop here!

1. Language, Syntax, Grammar
2. Formal languages, grammars and types
3. Tokenization and syntax analysis
4. Stems and stemming

Boolean retrieval

BIR is based on **Boolean logic** and classical set theory in that both the documents to be searched and the user's query are conceived as sets of terms (a **bag-of-words model**). Retrieval is based on **whether or not the documents contain the query terms**.

(Wikipedia), [The Book]

- **Boolean query**
  - E.g., `(“obama” AND “healthcare” AND NOT “news”)`

Search and recommender systems idea

Boolean retrieval, exact nearest neighbour search or exact range queries can be too **expensive.** *Can we do faster?*

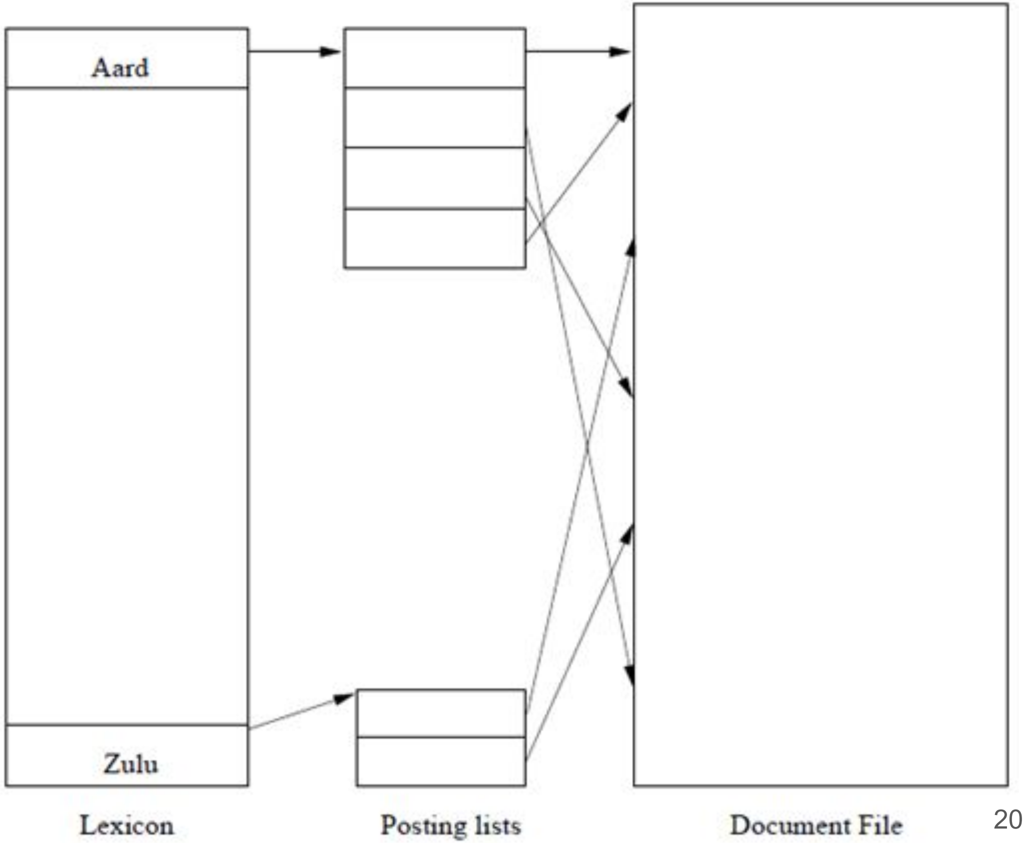
Pre-select (*pre-ranking sets*, approximate NN …) which is done fast (e.g. O(log(N)), selects enough to catch [almost] all relevant elements. Requires data structures: ***indices***.

Select (*ranking*, exact match) is done on a smaller set (PRS).

Text indexing: inverted index

Inverted index

1) Build a **lexicon** for the whole database
2) For each word of lexicon build a **posting list** (set of pointers)
3) [optional] persist this structure as a sparse matrix

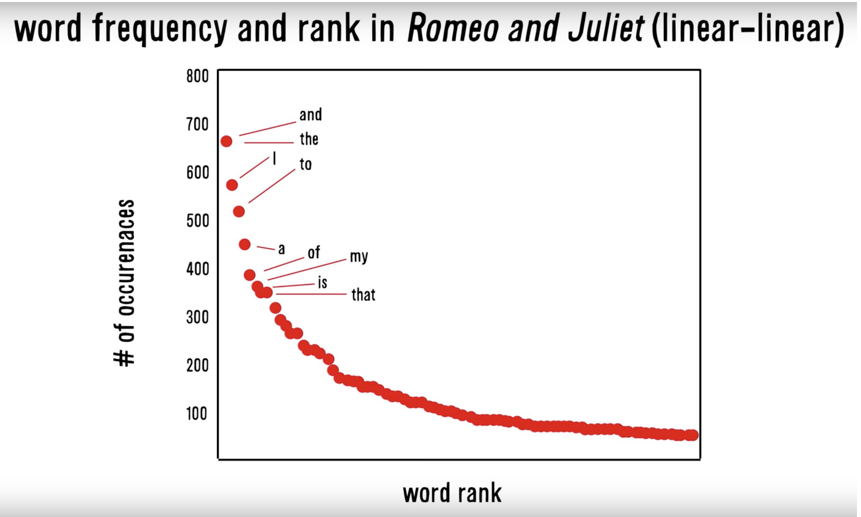


Remarks on lexicon

- Languages have lots of names and phrasal forms of 2+ words (e.g. New York) → use **bigrams** and alternative tokenizations (e.g. subword tokenization)
- Stopwords should be selected carefully

Document frequency

Idea: a term is more discriminative if it occurs only in fewer documents



https://medium.com/datadriveninvestor/zipfs-law-breakdown-application-in-app-development-5e9cda70cdc8

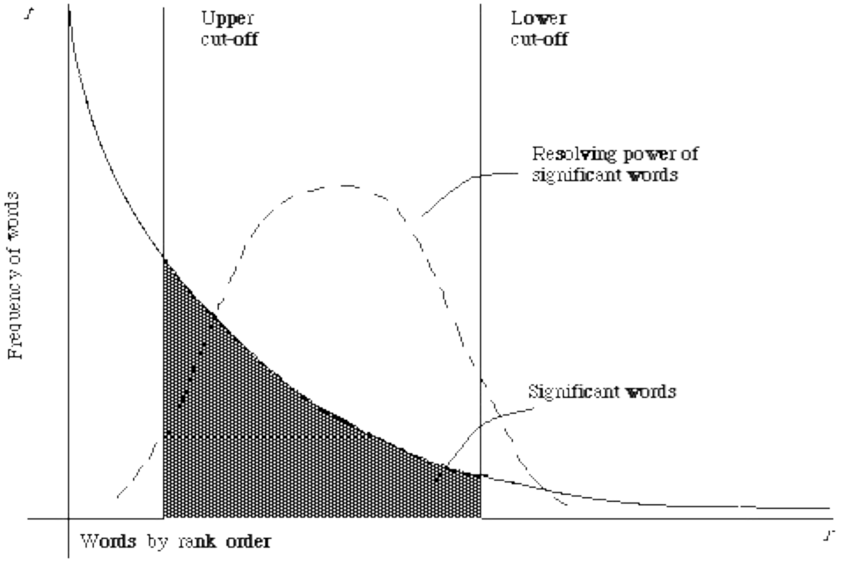


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

Subjective remarks on dictionary construction

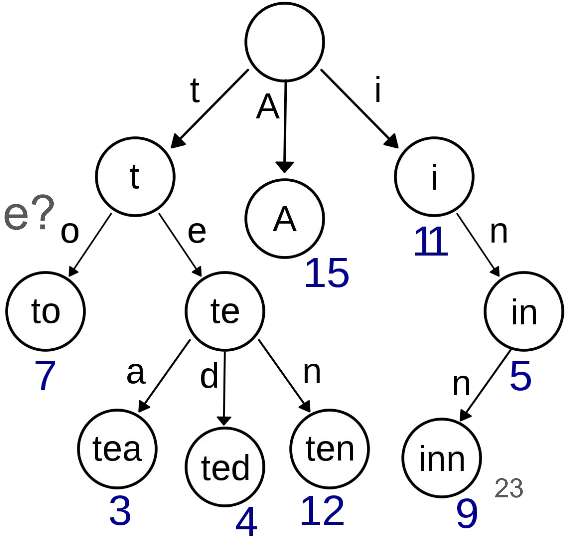
External memory dictionaries are almost useless* for real-time applications. Files are used for persistence (`mmap`)

Index **building** requires more memory, than index storage itself. Map-Reduce is widely used for this. (Book, 4.4)

Indices are usually **static**. For dynamic read (Book, 4.5)

What if total number of documents is big for a single machine?

- Use Trie
- Use Sharding: *hash* based, *lexicographical*



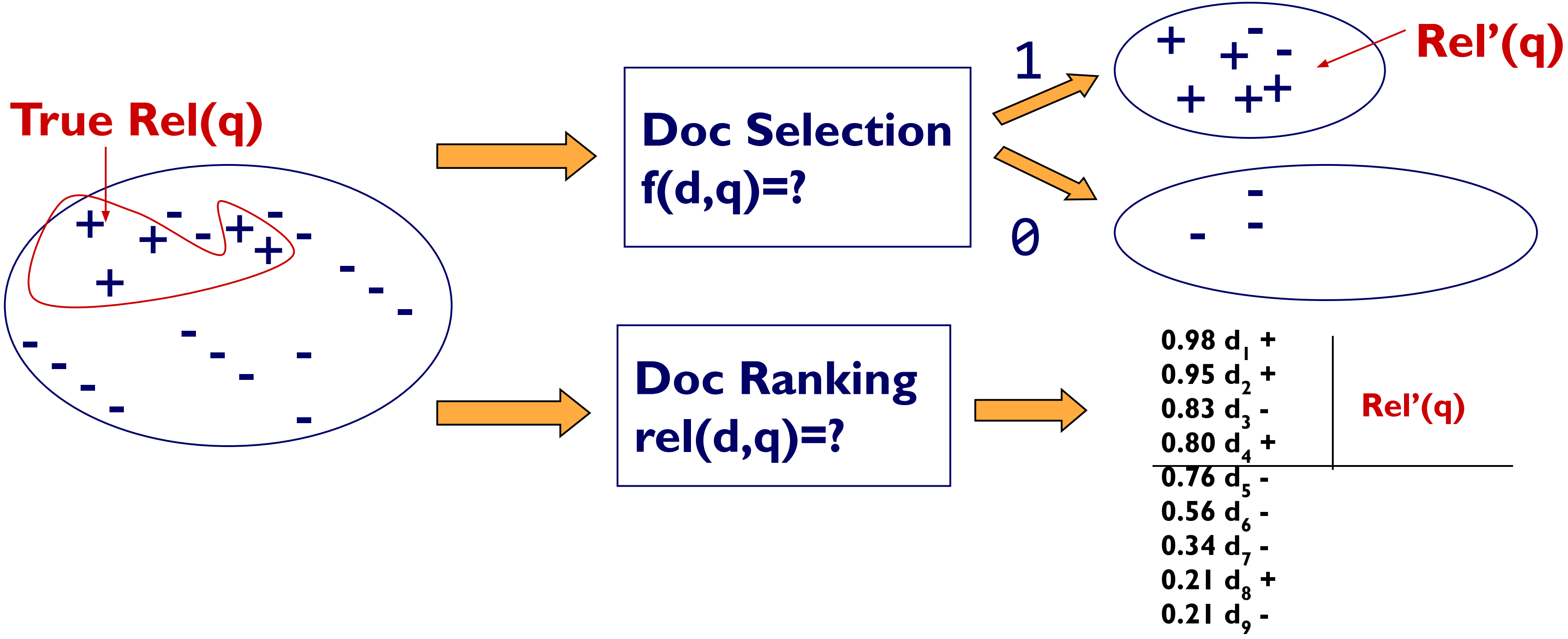
Search with Boolean query

- **Boolean query**
  - E.g., `“obama” AND “healthcare” AND NOT “news”`
- **Pre-ranking set**
  - Lookup query term in the dictionary
  - Retrieve the posting lists
  - Operation
    - AND: intersect the posting lists (skip-lists can help to intersect in O(m+n))
    - OR: union the posting list
    - NOT: diff the posting list
- **“Ranking”: Last step**
  - *Re-check* selected documents hold **expected substring** (for query search)

Deficiency of Boolean model

- The query is unlikely precise
  - "**Over-constrained**" **query** (terms are too specific): no relevant documents can be found
  - "**Under-constrained**" **query** (terms are too general): over delivery
  - It is hard to find the right position between these two extremes (hard for users to specify constraints)
- Even if it is accurate
  - Not all users would like to use such queries
  - All relevant documents are **not equally** important
    - No one would go through all the matched results
- Relevance is a matter of degree!

Document Selection vs. Ranking



Ranking is often preferred

- Relevance is a matter of degree
  - Easier for users to **find** appropriate **queries**
- A user can stop browsing anywhere, so the **boundary** is **controlled by the user**
  - Users prefer *coverage* would view more items
  - Users prefer *precision* would view only a few
- Theoretical justification: Probability Ranking Principle
  - relevance has a probabilistic interpretation. According to this principle documents are ranked by a probability ***p***(*Rel*|*d*, *q*), where *Rel* denotes the event of a document *d* being relevant to a query *q*

Retrieval procedure in modern IR

- Boolean model provides <u>all</u> the ranking candidates
  - Locate documents satisfying (somehow) Boolean condition
    - E.g., “obama healthcare” -> “obama” **OR** “healthcare”
- Rank candidates by **relevance**
  - Important: the definition of relevance
- Efficiency consideration
  - Top-k retrieval (Google example of page 110)

Intuitive understanding of relevance

	information	retrieval	retrieved	is	helpful	for	you	everyone
Doc1	1	1	0	1	1	1	0	1
Doc2	1	0	1	1	1	1	1	0
Query	1	1	0	0	0	0	0	0

E.g., 0/1 for Boolean models,
***probabilities** for probabilistic models*

Ranking over Inverted Index: TF-IDF

Term frequency **tf**(*t,d*), count of a term *t* in a document *d*.

- Boolean "frequencies": $\mathrm{tf}(t,d) = 1$ if t occurs in d and 0 otherwise;
- **term frequency adjusted** for document length : $f_{t,d}$ ÷ (number of words in *d*)
- **logarithmically scaled** frequency: $\mathbf{tf}(t,d) = \log(1 + f_{t,d})$
- augmented frequency

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

Inverse document frequency $\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

Home reading

The book, chapter 4 (index construction)