Chapter III: Ranking Principles

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Winter Semester 2011/12

Chapter III: Ranking Principles*

III.1 Document Processing & Boolean Retrieval

Tokenization, Stemming, Lemmatization, Boolean Retrieval Models

III.2 Basic Ranking & Evaluation Measures

TF*IDF & Vector Space Model, Precision/Recall, F-Measure, MAP, etc.

III.3 Probabilistic Retrieval Models

Binary/Multivariate Models, 2-Poisson Model, BM25, Relevance Feedback

III.4 Statistical Language Models (LMs)

Basic LMs, Smoothing, Extended LMs, Cross-Lingual IR

III.5 Advanced Query Types

Query Expansion, Proximity Ranking, Fuzzy Retrieval, XML-IR

*Mostly following **Manning/Raghavan/Schütze,** with additions from other sources

Chapter III.1: Document processing & Boolean Retrieval

1. First Example
2. Boolean retrieval model
    2.1. Basic and extended Boolean retrieval
    2.2. Boolean ranking
3. Document processing
    3.1. Basic ideas and tokenization
    3.2. Stemming & lemmatization
4. Edit distances and spelling correction

Based on **Manning/Raghavan/Schütze,** Chapters 1.1, 1.4, 2.1, 2.2, 3.3, and 6.1

First example: Shakespeare

- Which plays of Shakespeare contain words *Brutus* and *Caesar* but do not contain the word *Calpurnia*?
- Get each play of Shakespeare from Project Gutenberg in plain text
- Use Unix utility grep to go thru the plays and select the ones that mach to *Brutus* AND *Caesar* AND NOT *Calpurnia*
  - grep --files-with-matches ‘Brutus’ * | \
    xargs grep --files-with-matches ‘Caesar’ | \
    xargs grep --files-without-match ‘Calpurnia’

Definition of Information Retrieval

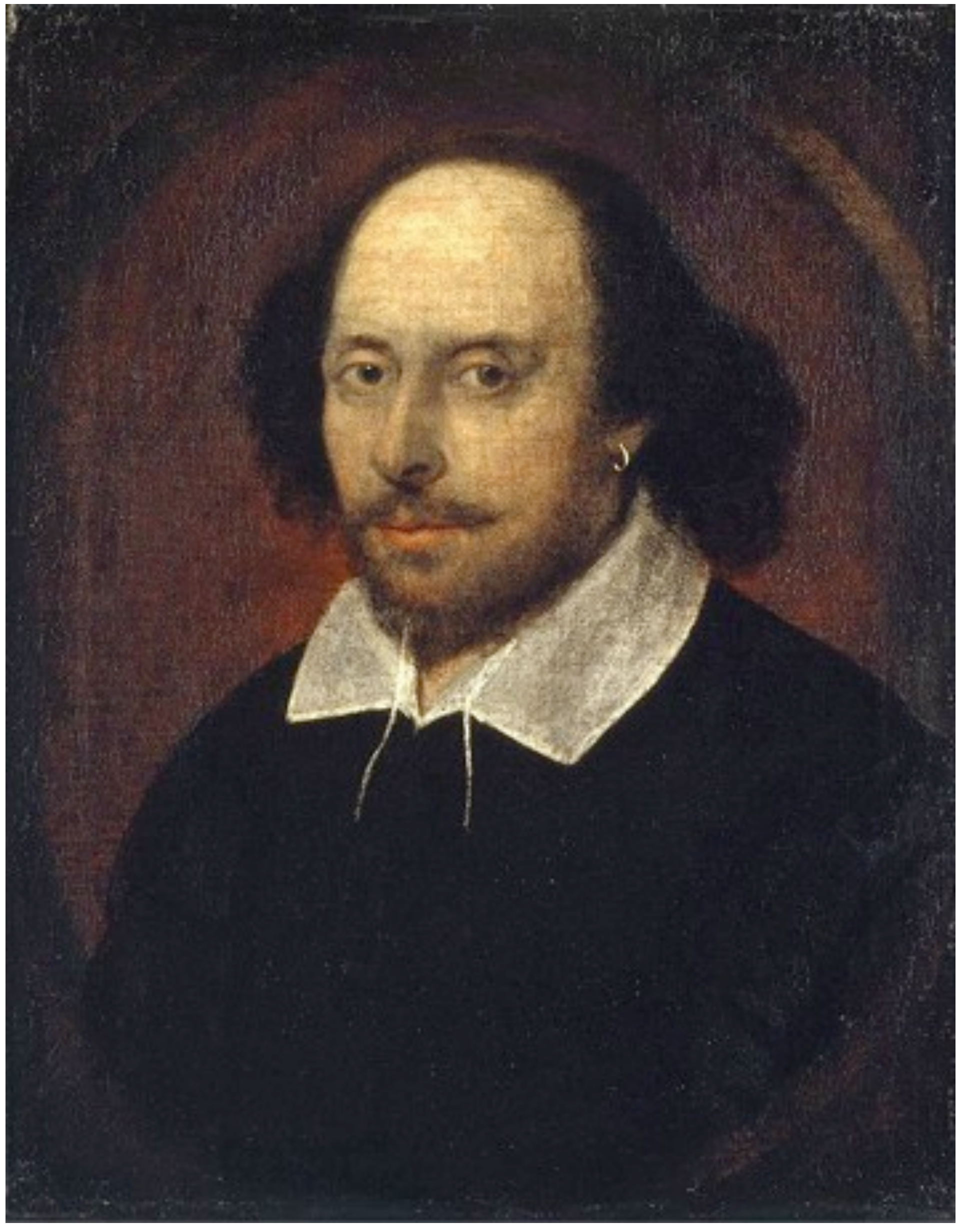
- Per Manning/Raghavan/Schütze:

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

  - **Unstructured data**: data without clear and easy-for-computer structure
    - e.g. text
  - **Structured data**: data with such structure
    - e.g. relational database
  - Large collection: the web
    - But also your computer: e-mails, documents, programs, etc.

Boolean Retrieval Model

- We want to find Shakespeare's plays with words *Caesar* and *Brutus*, but not *Calpurnia*
  - Boolean query
    `Caesar AND Brutus AND NOT Calpurnia`
  - Answer is all the plays that satisfy the query
- We can construct arbitrarily complex queries
- Result is an unordered set of plays with that satisfy the query



Incidence matrix

- Binary terms-by-documents matrix
  - Each column is a binary vector describing which terms appear in the corresponding documents
  - Each row is a binary vector describing which documents have the corresponding term
  - To answer to the Boolean query, we take the rows corresponding to the query terms and apply the Boolean operators element-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 queries

- Boolean queries used to be the standard
  - Still common with e.g. library systems
- Plain Boolean queries are too restricted
  - Queries look terms anywhere in the document
  - Terms have to be exact
- Extensions to plain Boolean queries
  - *Proximity operator* requires two terms to appear close to each other
    - Distance is usually defined using either words appearing between the terms or structural units such as sentences
  - *Wildcards* avoid the need for stemming/lemmatization

Boolean ranking

- Many documents have zones
  - Author, title, body, abstract, etc.
- A query can be satisfied by many zones
- Results can be ranked based on how many zones the article satisfies
  - Fields are given weights (that sum to 1)
  - The score is the sum of weights of those fields that satisfy the query
  - Example: query *Shakespeare* in author, title, and body
    - Author weight = 0.2, title = 0.3, and body = 0.5
    - Article with *Shakespeare* in title and body but not in author would obtain score 0.8

Document processing

- From natural language documents to easy-for-computer format
- Query term can be misspelled or be in wrong form
  - plural, past tense, adverbial form, etc.
- Before we can do IR, we must define how we handle these issues
  - ‘Correct’ handling is very much language-dependent

What is a document?

- If data are not in some linear plain-text format (ASCII, UTF-8, etc.), it needs to be converted
  - Escape sequences (e.g. `&`); compressed files; PDFs, etc.
- Data has to be divided into *documents*
  - A document is a basic unit of answer
    - Should *Complete Works of Shakespeare* be considered as a single document? Or should each act of each play be a document?
    - Unix `mbox`-format stored each e-mail into one file, should they be separated?
    - Should one-page-per-section HTML-pages be concatenated into one document?

Tokenization

- Tokenization splits text into **tokens**

Friends, Romans, Countrymen, lend me your ears;

Friends	Romans	Countrymen	lend	me	your	ears

- A **type** is a class of all tokens with same character sequence
- A **term** is a (possibly normalized) type that is included into IR system's dictionary
- Basic tokenization
  - Split at white space
  - Throw away punctuation

Issues with tokenization

- Language- and content-dependent

Issues with tokenization

- Language- and content-dependent
  - Boys’ $\Rightarrow$ Boys vs. can’t $\Rightarrow$ can t

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man

Issues with tokenization

- Language- and content-dependent
  - Boys' ⇒ Boys vs. can't ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l’ensemble* and *un ensemble*

Issues with tokenization

- Language- and content-dependent
  - Boys' $\Rightarrow$ Boys vs. can't $\Rightarrow$ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l'ensemble* and *un ensemble*
  - Compound nouns

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l’ensemble* and *un ensemble*
  - Compound nouns
    - *Lebensversicherungsgesellschaftsangestellter*

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l’ensemble* and *un ensemble*
  - Compound nouns
    - *Lebensversicherungsgesellschaftsangestellter*
  - Noun cases

Issues with tokenization

- Language- and content-dependent
  - Boys’ ⇒ Boys vs. can’t ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l’ensemble* and *un ensemble*
  - Compound nouns
    - *Lebensversicherungsgesellschaftsangestellter*
  - Noun cases
    - *Talo* (a house) vs. *talossa* (in a house), *lammas* (a sheep) vs. *lampaan* (sheep’s)

Issues with tokenization

- Language- and content-dependent
  - Boys' ⇒ Boys vs. can't ⇒ can t
  - http://www.mpi-inf.mpg.de and pauli.miettinen@mpi-inf.mpg.de
  - co-ordinates vs. a good-looking man
  - straight forward, white space, Los Angeles
  - *l'ensemble* and *un ensemble*
  - Compound nouns
    - *Lebensversicherungsgesellschaftsangestellter*
  - Noun cases
    - *Talo* (a house) vs. *talossa* (in a house), *lammas* (a sheep) vs. *lampaan* (sheep's)
  - No spaces at all (major East Asian languages)

Stop words

- **Stop words** are extremely common words that are excluded from the system's vocabulary
  - *a, an, and, are, as, at, be, by, for, from, has, he, in, is, …*
- Do not seem to help and removing saves space
- Removing can cause problems
  - President of the United States vs. President United States
  - *Let it be; to be or not to be;* etc.
- Current trend towards shorter or no stop word lists

Stemming

- Variations of words could be grouped together
  - E.g. plurals, adverbial forms, verb tenses
- A crude heuristic to cut the ends of the words
  - *ponies* ⇒ *poni*; *individual* ⇒ *individu*
- Exact stem does not need to be a proper word
  - variations of same word should have unique stem
- Most popular one in English is *Porter Stemmer*
  - http://tartarus.org/martin/PorterStemmer/

Example of stemming

Original: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lemmatization

- A *lemmatizer* produces full morphological analysis of the word to identify the *lemma* of the word
  - *Lemma* is the dictionary form of the word
- With input *saw* stemmer might return either *s* or *saw*, whereas lemmatizer tries to define if the word is noun (return *saw*) or verb (return *see*)
- With English lemmatizers do not produce considerable improvements over stemmers
  - But stemmers do not help that much, either

Other ideas

- Diacritic removal
  - Remove diacritics, e.g. ü ⇒ u, å ⇒ a, ø ⇒ o
  - Many queries do not include diacritics
  - Sometimes diacritics are typed using multiple characters
    - für ⇒ fuer
- ***n*-grams** are sequences of *n* characters (inter- or intra-word)
  - Very useful with Asian languages without clear word spaces
- Lower-casing words
  - Truecasing tries to use the correct capitalization
  - But users rarely use correct capitalization

Does any of this help?

- Depends on language, but not much with English
- Some results with 8 European languages (Hollink et al. 2004)
  - Diacritic removal helps with Finnish, French, and Swedish
  - Stemming helps with Finnish (30% improvement)
    - With English gains 0–5%, even poorer with lemmatizer
  - Compound splitting improved Swedish (25%) and German (5%)
  - Intra-word 4-grams helped Finnish (32%), Swedish (27%), and German (20%)
- In summary, morphologically rich languages benefit most

Edit distances and spelling correction

- If user types term that is not in our vocabulary, it is possibly misspelled
- We can try to recover from that by mapping the query term to the most similar term in our vocabulary
- But to do that we need to define a distance between terms
- We can consider basic types of spelling errors
  - adding extra characters (hoouse vs. house)
  - omitting some characters (huse)
  - using wrong character (hiuse)

Hamming edit distance

- All distances should admit triangle inequality
  - $d(x,y) \leq d(x,z) + d(z,y)$ for strings x, y, and z and distance d
- Hamming is the simplest distance

Hamming distance of strings x and y is the number of positions where x and y are different.

- Normally x and y must be of same length
  - We can pad the shorter one with null characters
- Corresponds to only using wrong characters
- Example:
  - Hamming distance between *car* and *bar* is 1, and between *house* and *hoosse* 3

Longest common subsequence

- Correspond to case when we have only dropped (or added) characters
- A *subsequence* of two strings *x* and *y* is a string *s* such that all characters of *s* appear in *x* and *y* in the same order as in *s* but not necessarily contiguously
  - Set of all subsequences of *x* and *y* is denoted *S(x,y)*

Longest common subsequence (LCS) distance of strings *x* and *y* (of *n* and *m* characters, respectively) is

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

- Example: LCS of *banana* and *atana* is *aana* and LCS distance is 2

Levenshtein edit distance

- All three types of errors are allowed

> **(Levenshtein) edit distance** of strings x and y is the number of additions, deletions, or substitutions of single characters of x required to make x equal to y.

- Example: distance between *houses* and *trousers* is 3:
  houses → *<u>r</u>ouses* → *<u>t</u>rouses* → *trouse<u>r</u>s*
- We can also add weights for edit operations
  - Different weights to substituting different characters
    - Based on how close the characters are on a keyboard
  - With proper weights, can be very effective

Computing the edit distance

- Dynamic-programming algorithm
  - Takes time $O(|x| \times |y|)$

```
int LevenshteinDistance(char s[1..m], char t[1..n])
{
 declare int d[0..m, 0..n]

 for i from 0 to m
  d[i, 0] := i // the distance of any first string to an empty second string
 for j from 0 to n
  d[0, j] := j // the distance of any second string to an empty first string

 for j from 1 to n
 {
  for i from 1 to m
  {
   if s[i] = t[j] then
    d[i, j] := d[i-1, j-1]      // no operation required
   else
    d[i, j] := minimum
           (d[i-1, j] + 1,  // a deletion
            d[i, j-1] + 1,  // an insertion
            d[i-1, j-1] + 1 // a substitution)
  }
 }
 return d[m,n]
}
```