Information Retrieval and Web Search

Text processing

Instructor: Rada Mihalcea

(Note: Some of the slides in this slide set were adapted from an IR course taught by Prof. Ray Mooney at UT Austin)
IR System Architecture

User Need
User Feedback
Query
Ranked Docs

User Interface
Text Operations
Query Operations
Searching
Ranking

Logical View
Indexing
Index
Retrieved Docs

Inverted file
Database Manager
Text Database
Text Processing Pipeline

Documents to be indexed
OR
User query

Token stream

Linguistic modules

Modified tokens

Indexer

Inverted index

Friends, Romans, countrymen.

friends roman countryman

friend roman countryman

2 4
1 2
13 16
From Text to Tokens to Terms

- **Tokenization** = segmenting text into tokens:
- **token** = a sequence of characters, in a particular document at a particular position.
- **type** = the class of all tokens that contain the same character sequence.
  - “… to **be** or not to **be** …”
  - “… so **be** it, he said …”
- **term** = a (normalized) type that is included in the IR dictionary.
- Example
  - text = “I slept and then I dreamed”
  - tokens = I, slept, and, then, I, dreamed
  - types = I, slept, and, then, dreamed
  - terms = sleep, dream (stopword removal).
Simple Tokenization

• Analyze text into a sequence of discrete tokens (words).
• Sometimes punctuation (e-mail), numbers (1999), and case (Republican vs. republican) can be a meaningful part of a token.
  – However, frequently they are not.
• Simplest approach is to ignore all numbers and punctuation and use only case-insensitive unbroken strings of alphabetic characters as tokens.
• More careful approach:
  – Separate ? ! ; : “ ‘ [ ] ( ) < >
  – Care with . - why? when?
  – Care with ... ??
Tokenization

• **Apostrophes** are ambiguous:
  – possessive constructions:
    • the book’s cover => the book’s cover
  – contractions:
    • he’s happy => he is happy
    • aren’t => are not
  – quotations:
    • ‘let it be’ => let it be

• **Whitespaces** in proper names or collocations:
  – San Francisco => San_Francisco
    • how do we determine it should be a single token?
Tokenization

• **Hyphenations:**
  - co-education => co-education
  - state-of-the-art => state of the art? state_of_the_art?
  - lowercase, lower-case, lower case => lower_case
  - Hewlett-Packard => Hewlett_Packard? Hewlett Packard?

• **Period**
  - Abbreviations: Mr., Dr.
  - Acronyms: U.S.A.
  - File names: a.out
Tokenization

• Numbers
  – 3/12/91
  – Mar. 12, 1991
  – 55 B.C.
  – B-52
  – 100.2.86.144

• Unusual strings that should be recognized as tokens:
  – C++, C#, B-52, C4.5, M*A*S*H.
Tokenization

• **Tokenizing HTML**
  – Should text in HTML commands not typically seen by the user be included as tokens?
    • Words appearing in URLs.
    • Words appearing in “meta text” of images.
  – Simplest approach is to exclude all HTML tag information (between “<“ and “>”) from tokenization.

*Note*: it’s important to use the same tokenization rules for the queries and the documents
Tokenization is Language Dependent

- Need to know the language of the document/query:
  - **Language Identification**, based on classifiers trained on short character subsequences as features, is highly effective.

- **French** (reduced definite article, postposed clitic, pronouns):
  - l'ensemble, un ensemble, donne-moi.

- **German** (compound nouns), need *compound splitter*:
  - Computerlinguistik
  - Lebensversicherungsgesellschaftsangestellter
  - *(life insurance company employee)*

  - **Compound Splitting for German**:
    - usually implemented by finding segments that match against dictionary entries.
Tokenization is Language Dependent

- **East Asian languages**, need *word segmenter*:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
    - Not always guaranteed a unique tokenization
  - Complicated in Japanese, with multiple alphabets intermingled.

  - **Word Segmentation for Chinese**:
    - ML sequence tagging models trained on manually segmented text:
      - *Logistic Regression, HMMs, Conditional Random Fields*.
    - Multiple segmentations are possible:
      - 和尚

      These characters can either be treated as one word “monk”, or as a sequence of two words “and” and “still”
Tokenization is Language Dependent

- **Agluttinative languages**
  - Tusaatsiarunnnangittualuujunga
  - *I can't hear very well.*

This long word is composed of a root word *tusaa*-'to hear' followed by five suffixes:

- -tsiaq- well
- -junnaq- be able to
- -nngit- not
- -tualuu- very much
- -junga 1st pers. singular present indicative non-specific
Tokenization is Language Dependent

- **Arabic and Hebrew:**
  - Written right to left, but with certain items like numbers written left to right.
  - Words are separated, but letter forms within a word form complex ligatures

أصبح الجزائر مستقلة في سنة 1962 بعد 132 عامًا من الاحتلال الفرنسي.

Algeria achieved its independence in 1962 after 132 years of French occupation.
Language Identification

- Simplest (and often most effective) approach: calculate the likelihood of each candidate language, based on training data
- Given a sequence of words (or characters) $w_1^n$
- For each candidate language, calculate:
  - $P(w_1^n) \approx \prod_{k=1}^n P(w_k | w_{k-1}); w_0 = \text{<start>}$
  - $P(w_n | w_{n-1}) = C(w_{n-1}w_n)/C(w_{n-1})$
    - where $C(w_{n-1}w_n)$ and $C(w_{n-1})$ are simple frequency counts collected from that language’s training data
- Choose the language that maximizes the probability
Language Identification – Avoiding Zero Counts

• What if a word / character (or a pair of words / characters) never occurred in the training data?

• Zero counts mislead your total probability

• Use smoothing: simple techniques to add a (very) small quantity to any count

• $P(w_n | w_{n-1}) = \frac{[C(w_{n-1}w_n)+1]}{[C(w_{n-1})+V]}$
  - where $C(w_{n-1}w_n)$ and $C(w_{n-1})$ are simple frequency counts collected from that language’s training data
  - $V$ is the vocabulary, i.e., total number of unique words (or characters) in the training data

• A tip: use log(P) to avoid very small numbers
Exercise

French:
ainsi de la même manière que l'on peut dire que le Royaume-Uni est un pays, on peut dire que l'Angleterre est un pays.

English:
the same way that we can say that United Kingdom is a country, we can also say that England is a country.

this

Assume character-level bigram model. Apply add-one smoothing. Assume V for French is 27, V for English is 23
Stopwords

• It is typical to exclude high-frequency words (e.g., function words: “a”, “the”, “in”, “to”; pronouns: “I”, “he”, “she”, “it”).

• Stopwords are language dependent

• For efficiency, store strings for stopwords in a hashtable to recognize them in constant time.
  – E.g., simple Python dictionary
Exercise

• How to determine a list of stopwords?
• For English? – may use existing lists of stopwords
  – E.g. SMART’s commonword list (~ 400)
  – WordNet stopword list
• For Spanish? Bulgarian?
Stopwords

- The trend is away from using them:
  - From large stop lists (200-300), to small stop lists (7-12), to none.
  - Good compression techniques (or cheap hardware) mean the cost for including stop words in a system is very small.
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - Relational queries: “flights to London”
Normalization

- **Token Normalization** = reducing multiple tokens to the same canonical term, such that matches occur despite superficial differences.
  1. Create equivalence classes, named after one member of the class:
    - {anti-discriminatory, antidiscriminatory}
    - {U.S.A., USA}
  2. Maintain relations between unnormalized tokens:
     - can be extended with lists of synonyms (car, automobile).
     1. Index unnormalized tokens, a query term is expanded into a disjunction of multiple postings lists.
     2. Perform expansion during index construction.
Normalization

- **Accents and diacritics** in French:
  - résumé vs. resume.

- **Umlauts** in German:
  - Tuebingen vs. Tübingen

- **Most important criterion:**
  - How do users like to write their queries for these words?
    - Even in languages that standardly have accents, users often may not type them:
      - Often best to normalize to a de-accented term
        - Tuebingen, Tübingen, Tubingen => Tubingen
Normalization

• **Case-Folding** = reduce all letters to lower case:
  – allow Automobile at beginning of sentences to match automobile.
  – allow user-typed ferrari to match Ferrari in documents.
  – but may lead to unintended matches:
    • the Fed vs. fed.
    • Bush, Black, General Motors, Associated Press, ...

• **Heuristic** = lowercase only some tokens:
  – words at beginning of sentences.
  – all words in a title where most words are capitalized.

• **Truecasing** = use a classifier to decide when to fold:
  – trained on many heuristic features.
Normalization

• British vs. American spellings:
  – colour vs. color.

• Multiple formats for dates, times:
  – 09/30/2013 vs. Sep 30, 2013.

• Asymmetric expansion:
  – Enter: window Search: window, windows
  – Enter: windows Search: Windows, windows, window
  – Enter: Windows Search: Windows
Lemmatization

- Reduce inflectional/variant forms to base form
- Direct impact on vocabulary size
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*

- How to do this?
  - Need a list of grammatical rules + a list of irregular words
  - Children → child, spoken → speak ...
  - Practical implementation: use WordNet’s morphstr function
    - Perl: WordNet::QueryData (first returned value from validForms function)
    - Python: NLTK.stem
Stemming

- Reduce tokens to “root” form of words to recognize morphological variation.
  - “computer”, “computational”, “computation” all reduced to same token “compute”
- Correct morphological analysis is language specific and can be complex.
- Stemming “blindly” strips off known affixes (prefixes and suffixes) in an iterative fashion.

for example compressed and compression are both accepted as equivalent to compress.

for example compres and compres are both accepted as equivalent to compres.
Porter Stemmer

- Simple procedure for removing known affixes in English without using a dictionary.
- Can produce unusual stems that are not English words:
  - “computer”, “computational”, “computation” all reduced to same token “comput”
- May conflate (reduce to the same token) words that are actually distinct.
- Not recognize all morphological derivations.
Typical rules in Porter

- **sses** $\rightarrow$ **ss**
- **ies** $\rightarrow$ **i**
- **ational** $\rightarrow$ **ate**
- **tional** $\rightarrow$ **tion**

- See class website for link to “official” Porter stemmer site
  - Provides Python ready to use implementations
Porter Stemmer Errors

• Errors of “comission”:
  – organization, organ → organ
  – police, policy → polic
  – arm, army → arm

• Errors of “omission”:
  – cylinder, cylindrical
  – create, creation
  – Europe, European
Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
  - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)

- Stemming is language- and often application-specific:
  - open source and commercial plug-ins.

- Does it improve IR performance?
  - mixed results for English: improves recall, but hurts precision.
    - operative (dentistry) ⇒ oper
  - definitely useful for languages with richer morphology:
    - Spanish, German, Finish (30% gains).