Text Preparation through Extended Tokenization

Data Mining and Information Engineering
11. – 13. July 2006, Prague

Marcus Hassler
Günther Fliedl

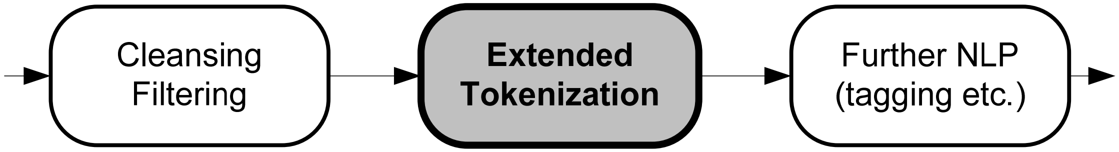
ALPEN-ADRIA
UNIVERSITÄT
KLAGENFURT

Overview

- Introduction
- Token Concepts
- Extended Tokenization
- Token Typing
- JavaTok
- Conclusion

Introduction

- Tokenization
  - first step of NL text preparation
  - stream of characters → stream of tokens (processing units)
- Supports any further NLP task
  - Tagging, Named Entity Recognition, Parsing, etc.



- Task taken for granted
  - already solved problem
  - theoretically uninteresting
  - without large impact (e.g., Information Retrieval)

Introduction

- Standard tokenization algorithm
  1) split strings separated by blanks / linefeeds
  2) split all punctuation marks (‘.’, ‘!’, ‘?’) ending those strings
- Difficulties of the standard algorithm
  - unclear token borders: doesn’t
  - sentence borders: der 15. Platz, semicolons
  - abbreviations: e.g., etc.
- Assumptions
  - language independent
  - domain independent
  - application independent

Token Concepts

- Single-tokens
  - Strings without non-printable or delimiting characters
  - Examples:
    - single words: `car`, `information`, `Sidney`
    - numbers: `12345`, `12.43`, `8,45`
    - internet addresses: `http://www.google.com`
- Multi-tokens
  - Strings through interpretation (may contain delimiters)
  - Examples:
    - composite nouns: `traffic jam`, `information retrieval`
    - special formats: `+43 463 2700-3511`, `ISDN-12 34567 / 89`
    - named entities: `United States of America`
    - formulas: `$x = x+1$`

Extended Tokenization

- Do as much as possible on STRING level, but not more!
- Extended Tokenization process
  1) identify single-tokens (standard tokenization)
  2) **type single-tokens**
  3) identify sentence end markers
  4) **reinterpret single-token types**
  5) merge and split tokens recursively (multi-tokens)
  6) **reinterpret any token type**

Extended Tokenization

- Incorporate many kinds of linguistic knowledge like
  - semantically motivated string patterns
    - e.g., phone numbers, serial codes, dates, URLs
  - dictionaries
    - e.g., abbreviations, names
  - morphosyntactic and sentence related rules
    - e.g., derivation (cold - cold*ness*), composition (scar*face*), capitalized term must start a new sentence

- Resources
  - language dependent
  - domain specific
  - application oriented

Token Typing

- Pre-linguistic classification process

- 3 step typing process
  1) type single-tokens (basic token types)
  2) reinterpret single-token types (user-defined token types)
  3) reinterpret token types (user-defined token types)

Token Typing: Basic Types

- Assigned straight-forward
- Basic token types (4)
  - alphabetics: `test`, `Test`, `TEST`, `TesT`
  - numerics: `123`, `12.3`, `1,23`, `12:34`
  - punctuation marks
    - sentence end marker
    - sentence-internal marks like comma
    - pair wise markers like brackets and quotes
  - mixtures
    - ending with sentence end marker
    - starting/ending with hyphen
    - containing slashes / hyphens
    - containing numbers
    - others

Token Typing: User-defined Types

- User-defined Token Types
  - expressed through strings
  - identified by rules and minimal dictionary knowledge

- Includes
  - domain knowledge
    - e.g., knowledge about data warehouses
  - gazetteer knowledge
    - e.g., country names, organization names
  - expert knowledge
    - e.g., medicine
  - pure linguistic knowledge
    - e.g., morphological and syntactical rules

Token Typing: User-defined Types

- Examples
  - abbreviations
  - acronyms
  - dates and times
  - phone numbers
  - email addresses
  - sequences of capitalized single-tokens (NE candidates)
  - stopwords
  - etc.

JavaTok

- Prototype
  - fully implemented in Java
  - part of the NLP toolset actually developed
  - online demo available at: http://nlp.ifit.uni-klu.ac.at/NLP/
- Features
  - free configuration and adaptation (UTF-16)
  - completely rule-based with dictionary support
  - enables user-defined token type definition
  - string replacements (abbreviation resolution, zero elimination, thesaurus, ...)
  - pre-tagging functionality (based on token types)
  - multiple output formats (TXT, HTML, XML)

JavaTok

- Rules
  - applied on token strings, token types, or both
  - support RegEx matching / substitution
  - access arbitrary long sequence of tokens
- Examples
  - suffix identification of well-known endings (e.g., `-ly`, `-ness`).
  - identification and reconcatenation of hyphenated words
  - sentence border disambiguation
  - multi-token identification
  - special character treatment, e.g., `& % $ § ° ´ \ /`

JavaTok

- Example text output for
  `The` **`Red Cross`** `is` **`aka.`** `RK.`

Output	S	M	R
The Red Cross is aka. RK .			
The Red Cross is **also known as** RK .			X
The (Red Cross)/**INST** is aka. RK .		X	
The (Red Cross)/**INST** is (**also known as**)/**ABBR** RK .		X	X
The/T_{a2} Red/T_{a2} Cross/T_{a2} is/T_{a1} aka./**ABBR** RK/T_{a3} ./T_{p1}	X		
The/T_{a2} Red/T_{a2} Cross/T_{a2} is/T_{a1} **also**/T_{a1} **known**/T_{a1} **as**/T_{a1} RK/T_{a3} ./T_{p1}	X		X
The/T_{a2} (Red/T_{a2} Cross/T_{a2})/**INST** is/T_{a1} aka./**ABBR** RK/T_{a3} ./T_{p1}	X	X	
The/T_{a2} (Red/T_{a2} Cross/T_{a2})/**INST** is/T_{a1} (**also**/T_{a1} **known**/T_{a1} **as**/T_{a1})/**ABBR** RK/T_{a3} ./T_{p1}	X	X	X

S = single-token typing, M = multi-token typing, R = replacement of strings

JavaTok

- Preliminary results
  - improvements of tagging outputs for
    - Stanford ME tagger
    - openNLP Tools ME tagger
    - QTag
- Corpus-based training (rule generation)
  - INEX (INitiative for the Evaluation of XML retrieval) collection
- Further steps
  - large scale evaluation
  - compare results to others

Conclusion

- Proper tokenization is crucial for *any* further NLP task
- Relies on the *token definition*
- Supported by rule-based *token typing*
- Online implementation *JavaTok*
  - http://nlp.ifit.uni-klu.ac.at/NLP/