Natural Language Processing

Basic Text Processing

Thanks to Dan Jurafsky and Chris Manning for reuse of (some) slides!
Basic text processing

• Before we can start processing a piece of text:
  • Segment text into tokens (words, punctuation marks, and other symbols)
  • Regularize tokens in various ways (spelling, capitalization, inflection)
  • Chunk tokens into sentences (often the largest meaningful units in NLP)

• Finite state technology
  • Transduction = mapping an input sequence to an output sequence
  • Performed efficiently using finite state automata (FSA, FST)
  • Specified using regular expressions (RE)
Regular Expressions

• A formal language for specifying text strings
• How can we search for any of these?
  • woodchuck
  • woodchucks
  • Woodchuck
  • Woodchucks
Regular Expressions: Literals

- Most characters represent themselves

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>woodchuck</td>
<td>woodchuck</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
</tbody>
</table>

- Some characters have to be escaped

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

This is a sentence. $100,000
Regular Expressions: Disjunctions

- **Letters inside square brackets []**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[wW]oodchuck</td>
<td>Woodchuck, woodchuck</td>
</tr>
<tr>
<td>[1234567890]</td>
<td>Any digit</td>
</tr>
</tbody>
</table>

- **Ranges [A–Z]**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A–Z]</td>
<td>An upper case letter Drenched Blossoms</td>
</tr>
<tr>
<td>[a–z]</td>
<td>A lower case letter my beans were impatient</td>
</tr>
<tr>
<td>[0–9]</td>
<td>A single digit Chapter 1: Down the Rabbit Hole</td>
</tr>
</tbody>
</table>
Regular Expressions: Character Classes

- Ranges work well for English, but what about é, ç, Å, ß, œ?
- Often better to use (LOCALE dependent) character classes
- In Python:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>ASCII equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>\w</td>
<td>Alphanumeric</td>
<td>[0-9A-Za-z_]</td>
</tr>
<tr>
<td>\d</td>
<td>Digit</td>
<td>[0-9]</td>
</tr>
<tr>
<td>\s</td>
<td>Whitespace</td>
<td>[ \t\n\r\f\v]</td>
</tr>
</tbody>
</table>
### Regular Expressions: Negation in Disjunction

- **Negations** \([ ^Ss ]\)
  - Caret means negation only when first in \[\]

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(^A-Z)</td>
<td>Not an upper case letter</td>
<td>Oyfn pripetchik</td>
</tr>
<tr>
<td>[^Ss]</td>
<td>Neither ‘S’ nor ‘s’</td>
<td>I have no exquisite reason”</td>
</tr>
<tr>
<td>[^e^]</td>
<td>Neither e nor ^</td>
<td>Look here</td>
</tr>
<tr>
<td>a(^b)</td>
<td>The pattern a caret b</td>
<td>Look up a(^b) now</td>
</tr>
</tbody>
</table>
Regular Expressions: More Disjunction

- Woodchucks is another name for groundhog!
- The pipe `|` for disjunction

<table>
<thead>
<tr>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>groundhog</td>
</tr>
<tr>
<td>yours</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>= [abc]</td>
</tr>
<tr>
<td>[gG]roundhog</td>
</tr>
</tbody>
</table>

Photo D. Fletcher

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>colou?r</code></td>
<td>Optional previous char</td>
</tr>
<tr>
<td><code>oo*h!</code></td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td><code>o+h!</code></td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td><code>baa+</code></td>
<td></td>
</tr>
<tr>
<td><code>beg.n</code></td>
<td></td>
</tr>
</tbody>
</table>

Stephen C Kleene
Kleene *, Kleene +
Regular Expressions: Anchors $^\text{\textdollar}$

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^[A-Z]$</td>
<td>Palo Alto</td>
</tr>
<tr>
<td>$^[^A-Za-z]$</td>
<td>1 &quot;Hello&quot;</td>
</tr>
<tr>
<td>.$</td>
<td>The end_</td>
</tr>
<tr>
<td>.$</td>
<td>The end? The end!</td>
</tr>
</tbody>
</table>
Example

• Find me all instances of the word “the” in a text.

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

[^\w][tT]he[^\w]

Good enough?
Errors

• The process we just went through was based on **fixing two kinds of errors**
  
  • Matching strings that we should not have matched *(there, then, other)*
    
    • False positives *(Type I)*
  
  • Not matching things that we should have matched *(The)*
    
    • False negatives *(Type II)*
Errors

- In NLP we are always dealing with these kinds of errors
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives)
Summary

• Regular expressions play a surprisingly large role
  • Sophisticated sequences of regular expressions are often the first model for any text processing text

• For many hard tasks, we use machine learning classifiers
  • But regular expressions are used as features in the classifiers
  • Can be very useful in capturing generalizations
Tokenization

- Segment text into tokens
  - words (*cat*, *part-of-speech*, *IBM*)
  - numerical expressions (*3.14*, *10,000*)
  - punctuation marks (*?, !, …*)
  - other symbols (*%, #, &*)

- Main sources of information in alphabetic writing
  - character classes (letters, numbers, etc.)
  - whitespace (space, newline, etc.)
Issues in Tokenization

• Finland’s capital → Finland Finlands Finland’s ?
• what’re, I’m, isn’t → What are, I am, is not
• Hewlett-Packard → Hewlett Packard ?
• state-of-the-art → state of the art ?
• Lowercase → lower-case lowercase lower case ?
• San Francisco → one token or two?
• m.p.h., PhD. → ??

NB: Different applications may require different tokenisations
Tokenization: language issues

- French
  - *L'ensemble* → one token or two?
  - *L ? L’ ? Le ?*
  - Want *l’ensemble* to match with *un ensemble*

- German noun compounds are not segmented
  - *Lebensversicherungsgesellschaftsangestellter*
  - ‘life insurance company employee’
  - German information retrieval needs *compound splitter*
Tokenization: language issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida

- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あたり$500K(約6,000万円)
Word Tokenization in Chinese

• Also called Word Segmentation
• Chinese words are composed of characters
  • Characters are generally 1 syllable and 1 morpheme.
  • Average word is 2.4 characters long.
• Standard baseline segmentation algorithm:
  • Maximum Matching (also called Greedy)
Maximum Matching

• Given a wordlist of Chinese, and a string:
  1) Start a pointer at the beginning of the string
  2) Find the longest word in dictionary that matches the string starting at pointer
  3) Move the pointer over the word in string
  4) Go to 2
Max-Match Segmentation Illustration

• Thecatinthehat the cat in the hat
• Thetabledownthere the table down there
• Doesn’t generally work in English!

• But works astonishingly well in Chinese
  • 莎拉波娃现在居住在美国东南部的佛罗里达。
  • 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
Normalization

- After tokenization, sometimes need to “normalize” terms
  - Information Retrieval: indexed text & query terms must have same form.
    - We want to match **U.S.A.** and **USA**
- We implicitly define equivalence classes of terms
  - For examples, deleting periods in a term
- Alternative: asymmetric expansion:
  - Enter: *window* Search: *window, windows*
  - Enter: *windows* Search: *Windows, windows, window*
  - Enter: *Windows* Search: *Windows*
Case folding

• Applications like IR: reduce all letters to lower case
  • Since users tend to use lower case
  • Possible exception: upper case in mid-sentence?
    • e.g., *General Motors*
    • *Fed* vs. *fed*
    • *SAIL* vs. *sail*

• For sentiment analysis, MT, information extraction
Morphology

• **Morphemes**: The building blocks of words
  - **Stems**: The core meaning-bearing units (*car, compute*)
  - **Affixes**: Bits and pieces that adhere to stems
    - Derivational – new words (*computer, computational*)
    - Inflectional – forms of a word (*cars, computed*)
Lemmatization and Stemming

- **Lemmatization**: Reduce inflected forms to base form (**lemma**)
  - *am, are, is → be*
  - *car, cars, car's, cars' → car*

- **Stemming**: Strip off (inflectional or derivational) suffixes
  - *car, cars, car's, cars' → car*
  - *computes, computed, computing, computer → comput*
Morphologically Rich Languages

• Some languages require complex morpheme segmentation
  • Turkish
    uygarlastiramadiklarimizdanmissinizcasina
    `(behaving) as if you are among those whom we could not civilize’
  • Analysis:
    uygar `civilized’ + las `become’ + tir `cause’ + ama `not able’
    + dik `past’ + lar ‘plural’ + imiz ‘p1pl’ + dan ‘abl’ + mis ‘past’
    + siniz ‘2pl’ + casina ‘as if’
Finite State Morphology

- Morphological analysis can be performed by a special kind of finite state automata called transducers
Sentence Segmentation

- !, ? are relatively unambiguous
- Period “.” is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Build a binary classifier
  - Decides EndOfSentence/NotEndOfSentence
  - Classifiers: hand-written rules, regular expressions, or machine learning
Decision Tree for Sentence Segmentation

- Lots of blank lines after me?
  - YES: E-O-S
  - NO
    - YES: Final punctuation is ?, !, or ?
    - NO
      - YES: E-O-S
      - NO: Final punctuation is period

- I am “etc” or other abbreviation
  - YES: Not E-O-S
  - NO
    - YES: Not E-O-S
    - NO: E-O-S
Implementing Decision Trees

- A decision tree is just an if-then-else statement
- The interesting research is choosing the features
- Setting up the structure is often too hard to do by hand
  - Hand-building only possible for very simple features, domains
  - Structure usually learned by machine learning from a training corpus