Basic Text Processing

Regular Expressions
Word Tokenization
Word Normalization
Sentence Segmentation

Many slides adapted from slides by Dan Jurafsky
Basic Text Processing

Regular Expressions
Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks
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>
<th>the First Match in an example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A–Z]</td>
<td>An upper case</td>
<td>Drenched Blossoms</td>
</tr>
<tr>
<td>[a–z]</td>
<td>A lower case letter</td>
<td>my beans were impatient</td>
</tr>
<tr>
<td>[0–9]</td>
<td>A single digit</td>
<td>Chapter 1: Down the Rabbit Hole</td>
</tr>
</tbody>
</table>
Regular Expressions: Negation in Disjunction

- Negations \[^Ss\]
  - Carat means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>Not an upper case</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 carat 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>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>groundhog</td>
<td>woodchuck</td>
</tr>
</tbody>
</table>
| yours|mine | yours  
machine |
| a|b|c|ab | abc |
| [gG]roundhog|[Ww]oodchuck |         |

Photo D. Fletcher
## Regular Expressions: \(? \) \(\ast\) \(+\) \(\cdot\)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>colou?r</td>
<td>0 or 1 of previous char</td>
</tr>
<tr>
<td></td>
<td>color, colour</td>
</tr>
<tr>
<td>oo*h!</td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td></td>
<td>oh!, ooh!, oooh!, ooooh!</td>
</tr>
<tr>
<td>o+h!</td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td></td>
<td>oh!, ooh!, oooh!, ooooh!</td>
</tr>
<tr>
<td>baa+</td>
<td></td>
</tr>
<tr>
<td></td>
<td>baa, baaa, baaaa, baaaaa</td>
</tr>
<tr>
<td>beg.n</td>
<td>any char</td>
</tr>
<tr>
<td></td>
<td>begin, begun, begun, begun, beg3n</td>
</tr>
</tbody>
</table>

Stephen C Kleene

Kleene *, Kleene +
## Regular Expressions: Anchors

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

  [^a-zA-Z][tT]he[^a-zA-Z]
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 cont.

- 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 task
• 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
Basic Text Processing

Regular Expressions
Basic Text Processing

Word tokenization
Text Normalization

• Every NLP task needs to do text normalization:
  1. Segmenting/tokenizing words in running text
  2. Normalizing word formats
  3. Segmenting sentences in running text
How many words?

• I do uh main- mainly business data processing
  • Fragments, filled pauses

• Seuss’s *cat* in the hat is different from other *cats*!
  • **Lemma**: same stem, part of speech, rough word sense
    • *cat* and *cats* = same lemma
  • **Wordform**: the full inflected surface form
    • *cat* and *cats* = different wordforms
How many words?

they lay back on the San Francisco grass and looked at the stars and their

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 15 tokens (or 14)
  - 13 types (or 12) (or 11?)
How many words?

\[ N = \text{number of tokens} \]

\[ V = \text{vocabulary} = \text{set of types} \]

\( |V| \) is the size of the vocabulary

Church and Gale (1990): \( |V| > O(N^{1/2}) \)

|                      | Tokens = N  | Types = |V|  |
|----------------------|-------------|---------|-----|
| Switchboard phone    | 2.4 million | 20 thousand |
| Shakespeare          | 884,000     | 31 thousand |
| Google N-grams       | 1 trillion  | 13 million |
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. → ??
Tokenization: language issues

- French
  - *L'ensemble* → one token or two?
    - 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
Basic Text Processing

Word tokenization
Basic Text Processing

Word Normalization and Stemming
Normalization

• 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
  • e.g., 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
  - Case is helpful (US versus us is important)
Lemmatization

• Reduce inflections or variant forms to base form
  • \textit{am, are, is} \rightarrow \textit{be}
  • \textit{car, cars, car's, cars'} \rightarrow \textit{car}

\textbf{Context dependent.} for instance:
- in our last meeting (noun, meeting).
- We’re meeting (verb, meet) tomorrow.

• \textit{the boy's cars are different colors} \rightarrow \textit{the boy car be different color}

• Lemmatization: have to find correct dictionary headword form
Morphology

- Morphemes:
  - The small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Bits and pieces that adhere to stems
    - Often with grammatical functions
Stemming  

context independent

- Reduce terms to their stems in information retrieval
- Stemming is crude chopping of affixes
  - language dependent
  - e.g., *automate(s), automatic, automation* all reduced to *automat*.

For example, compressed and compression are both accepted as equivalent to compress.
Porter’s algorithm
The most common English stemmer

fixed rules put in groups, applied in order.  
https://tartarus.org/martin/PorterStemmer/

Step 1a

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example词语</th>
<th>Stemmed单词</th>
</tr>
</thead>
<tbody>
<tr>
<td>sses → ss</td>
<td>caresses → caress</td>
<td></td>
</tr>
<tr>
<td>ies → i</td>
<td>ponies → poni</td>
<td></td>
</tr>
<tr>
<td>ss → ss</td>
<td>caress → caress</td>
<td></td>
</tr>
<tr>
<td>s → ∅</td>
<td>cats → cat</td>
<td></td>
</tr>
</tbody>
</table>

Step 1b

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example词语</th>
<th>Stemmed单词</th>
</tr>
</thead>
<tbody>
<tr>
<td>(<em>v</em>)ing → ∅</td>
<td>walking → walk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sing → sing</td>
<td></td>
</tr>
<tr>
<td>(<em>v</em>)ed → ∅</td>
<td>plastered → plaster</td>
<td></td>
</tr>
</tbody>
</table>

Step 2 (for long stems)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example词语</th>
<th>Stemmed单词</th>
</tr>
</thead>
<tbody>
<tr>
<td>ational → ate</td>
<td>relational → relate</td>
<td></td>
</tr>
<tr>
<td>izer → ize</td>
<td>digitizer → digitize</td>
<td></td>
</tr>
<tr>
<td>ator → ate</td>
<td>operator → operate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Step 3 (for longer stems)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example词语</th>
<th>Stemmed单词</th>
</tr>
</thead>
<tbody>
<tr>
<td>al → ∅</td>
<td>revival → reviv</td>
<td></td>
</tr>
<tr>
<td>able → ∅</td>
<td>adjustable → adjust</td>
<td></td>
</tr>
<tr>
<td>ate → ∅</td>
<td>activate → activ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Basic Text Processing

Word Normalization and Stemming
Basic Text Processing

Sentence Segmentation and Decision Trees
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
  - Looks at a “.”
  - Decides EndOfSentence/NotEndOfSentence
  - Classifiers: hand-written rules, regular expressions, or machine-learning
Determining if a word is end-of-sentence: a Decision Tree
Decision Trees and other classifiers

- We can think of the questions in a decision tree
- As features that could be exploited by any kind of classifier
  - Logistic regression
  - SVM
  - Neural Nets
  - etc.
Sentence Splitters

- Stanford coreNLP: (deterministic)
  - http://stanfordnlp.github.io/CoreNLP/

- UIUC sentence splitter: (deterministic)
  - https://cogcomp.cs.illinois.edu/page/tools_view/2
Basic Text Processing

Sentence Segmentation and Decision Trees