Text processing

NLP tasks typically require multiple steps of text processing:
- Segmenting/tokenizing words in text
- Normalizing word forms
- Segmenting sentences
Tokenization

- Tokenization: segmenting text into words, using punctuation as separate tokens
- Example: "The San Francisco-based restaurant," they said, "doesn’t charge".
  "The | San | Francisco-based | restaurant | , | " | they | said | , | " | does | n’t | charge | "
- Seems like an easy problem. But:
  – Commas typically appear at word boundaries. Except: 1,000,000
  – Apostrophes should be parsed differently than quotation marks than in situations like: "books’s cover or they’re"

Issues in tokenization

- Finland’s capital  →  Finland 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?
    • L? L’? L.?

• German noun compounds are not segmented
  – Lebensversicherungsgesellschaftsangestellter
  – ‘life insurance company employee’
  – German information retrieval needs **compound splitter**

Normalization

• Need to “normalize” terms: bring them into a common form
  – Example: we want to match US, U.S.A. and USA

• Case folding: convert everything to lowercase
  – But: need to distinguish between US and us
  – Exceptions: upper case in mid-sentence
    • e.g., *General Motors*
    • *Fed vs. fed*
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
    - Example: the word cats is composed of the stem "cat" and "s"

Stemming

- Reduce terms to their stems
- 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.

for exampl compress and compress ar both accept as equval to compress
Porter’s algorithm
The most common English stemmer

Step 1a

| sses → ss | caresses → caress |
| ies → i  | ponies → poni   |
| ss → ss  | caress → caress |
| s  → ø   | cats → cat     |

Step 1b

(*v*)ing → ø  walking → walk
sing  → sing
(*v*)ed → ø  plastered → plaster

Step 2 (for long stems)

| ational → ate | relational → relate |
| izer → ize    | digitizer → digitize |
| ator → ate    | operator → operate  |

Step 3 (for longer stems)

| al → ø    | revival → reviv |
| able → ø  | adjustable → adjust |
| ate → ø   | activate → activ |

Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

(*v*)ing → ø  walking → walk
sing  → sing

tr -sc 'A-Za-z' '
' < shakes.txt | grep 'ing$' | sort | uniq -c | sort -nr

1312 King
548 being
541 nothing
152 something
388 king
375 bring
358 thing
307 ring
152 something
145 coming
116 Being
130 morning
102 going
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, e.g. [A–Z]

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

- Negation \[^{\text{Ss}}\]  
  - Caret means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>Not an upper case letter</td>
</tr>
<tr>
<td>[^{\text{Ss}}]</td>
<td>Neither ‘S’ nor ‘s’</td>
</tr>
<tr>
<td>[^{\text{e}}]</td>
<td>Neither e nor ^</td>
</tr>
<tr>
<td>a^b</td>
<td>The pattern a caret b</td>
</tr>
</tbody>
</table>

Regular Expressions: More Disjunction

- Woodchuck 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>
<tr>
<td>yours</td>
<td>mine</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>[gG]roundhog</td>
<td>[Ww]oodchuck</td>
</tr>
</tbody>
</table>
### Regular Expressions: \(? \) \(* \) \(+ \) \(.*\)

<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>

### 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>“Hello”</td>
</tr>
<tr>
<td>.<code>\$</code></td>
<td>The end_</td>
</tr>
<tr>
<td><code>\.$</code></td>
<td>The end!</td>
</tr>
</tbody>
</table>

---

Stephen C Kleene

Kleene *, Kleene +
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
  - Not matching things that we should have matched (The)
    - False negatives
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 precision (minimizing false positives)
  – Increasing coverage or recall (minimizing false negatives).

Regular expressions

• 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
Lemmatization

- Reduce variant forms to a base form
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- the boy's cars are different colors → the boy car be different color

Stemming vs lemmatization

- Stemming: a crude heuristic process that chops off the ends of words
- Lemmatization: return the base or dictionary form of a word, which is known as the lemma.
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

```
Lots of blank lines after me?

E-O-S

Final punctuation is ?, !, or :?

E-O-S

Final punctuation is period

I am “etc” or other abbreviation

Not E-O-S

Not E-O-S

E-O-S
```

E-O-S