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>
</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>
</tbody>
</table>

_Drenched Blossoms_  
_my beans were impatient_  
_Chapter 1: Down the Rabbit Hole_
Regular Expressions: Negation in Disjunction

Negations \([ ^Ss ]\)

- Carat 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><em>Oyfn pripetchik</em></td>
</tr>
<tr>
<td>[ ^Ss ]</td>
<td>Neither ‘S’ nor ‘s’</td>
<td><em>I have no exquisite reason”</em></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

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>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>colou?r</td>
<td>Optional previous char</td>
</tr>
<tr>
<td></td>
<td><strong>color</strong> <strong>colour</strong></td>
</tr>
<tr>
<td>oo*h!</td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td></td>
<td><strong>oh! ooh! oooh! ooooh!</strong></td>
</tr>
<tr>
<td>o+h!</td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td></td>
<td><strong>oh! ooh! oooh! ooooh!</strong></td>
</tr>
<tr>
<td>baa+</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>baa baaa baaaa baaaaa</strong></td>
</tr>
<tr>
<td>beg.n</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>begin begun begun begun beg3n</strong></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><em>Palo Alto</em></td>
</tr>
<tr>
<td><code>^[^A-Za-z]</code></td>
<td><em>1 “Hello”</em></td>
</tr>
<tr>
<td>.$</td>
<td>The end_</td>
</tr>
<tr>
<td>.$</td>
<td>The end? The end!</td>
</tr>
</tbody>
</table>

The end.
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:

1. Matching strings that we should not have matched (there, then, other)
   False positives (Type I errors)

2. Not matching things that we should have matched (The)
   False negatives (Type II errors)
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 text

For hard tasks, we use machine learning classifiers
  ◦ But regular expressions are still used for pre-processing, or as features in the classifiers
  ◦ Can be very useful in capturing generalizations
Basic Text Processing

Regular Expressions
More Regular Expressions: Substitutions and ELIZA
Substitutions

Substitution in Python and UNIX commands:

\texttt{s/regexp1/pattern/}

e.g.:
\texttt{s/colour/color/}
Capture Groups

• Say we want to put angles around all numbers:
  \textbf{the 35 boxes} \rightarrow \textbf{the <35> boxes}

• Use parens () to "capture" a pattern into a numbered register (1, 2, 3…)

• Use \textbackslash 1 to refer to the contents of the register
  \texttt{s/\(( [0-9]+\)/<\textbackslash 1>/}
Capture groups: multiple registers

/the (.*)er they (.*)/, the \
1er we \2/

Matches

  the faster they ran, the faster we ran

But not

  the faster they ran, the faster we ate
But suppose we don't want to capture?

Parentheses have a double function: grouping terms, and capturing

Non-capturing groups: add a ?: after paren:

```
/(?:some|a few) (people|cats) like some \1/
```

matches
- some cats like some cats

but not
- some cats like some some
Lookahead assertions

(?= pattern) is true if pattern matches, but is zero-width; doesn't advance character pointer

(?! pattern) true if a pattern does not match

How to match, at the beginning of a line, any single word that doesn’t start with “Volcano”:

/^(?![Volcano])[A-Za-z]+/
Simple Application: ELIZA

Early NLP system that imitated a Rogerian psychotherapist

Uses pattern matching to match, e.g.,:
  ◦ “I need X”
and translates them into, e.g.
  ◦ “What would it mean to you if you got X?”
Simple Application: ELIZA

Men are all alike.
IN WHAT WAY

They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
How ELIZA works

s/.*/I’M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \

s/.*/I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \

s/.*/all .*/IN WHAT WAY?/

s/.*/always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/
More Regular Expressions: Substitutions and ELIZA