

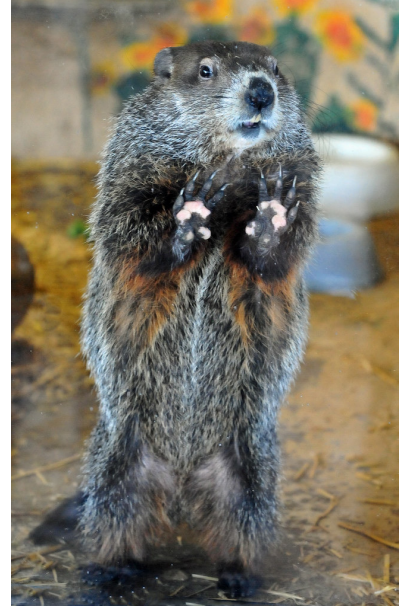
1.2 Regular Expressions

Edwin Simpson

Department of Computer Science,
University of Bristol, UK.

How Can I Find Occurrences Of...

- Different word forms referring to the same thing:
 - *Woodchuck, woodchuck, woodchucks, or Woodchucks?*
 - *Barack Obama, Obama, or 44th President of the US?*
 - *Happy, joyful, cheerful, or delighted?*



How Can I Find Occurrences Of...

- Phrases within a particular expression:
 - *The movie was very ... (e.g., enjoyable/boring)*
 - *The patient's symptoms are ...*

Regular Expressions (REs)

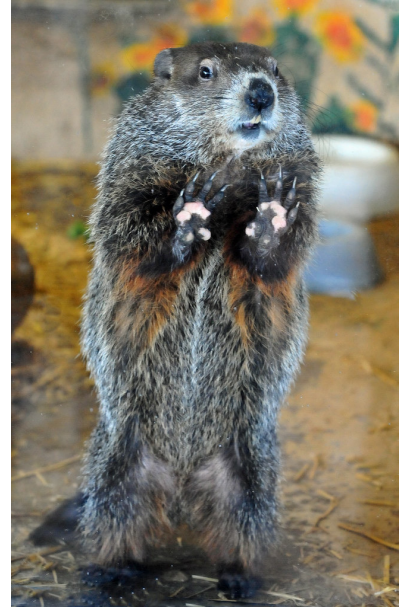
- A formal language for describing text strings
- Used in:
 - Unix tools like grep, emacs
 - Search
 - Computer languages
 - Parsing
 - Various NLP tasks and pipeline steps

Regular Expressions (REs)

- Used to **search for** phrases that match a pattern
 - Information extraction without machine learning!
 - Define text features for a classifier
- ...and to **replace** occurrences of a pattern
 - Error correction
 - Preprocessing: use a more convenient form for downstream methods
 - E.g., so that *Woodchuck*, *woodchuck*, *woodchucks*, and *Woodchucks* are all replaced by a single form so they can be treated the same by a classifier

Regular Expressions (RegExps)

- RegExp: defines a set of strings
- RegExp search function: returns all occurrences of strings within that set from a corpus of text
- <https://www.whatsmyip.org/regular-expression-tester/>
- To search for “woodchuck”: /woodchuck/



Building Blocks for REs: Disjunctions

- Use **disjunctions** match with alternative characters
- E.g., */[Ww]oodchuck/*

Pattern	What does it match?
[Ww]	W or w
[A-Z]	Any upper-case letter
[a-z]	Any lower-case letter
[0-9]	Any single digit

Building Blocks: Wildcards, Negation, Repetition

Pattern	Role	Example	What does it match?
<code>^</code> inside <code>[]</code>	Negation	<code>[^A-Z]</code>	any char not an upper-case letter
<code>.</code>	Any character	<code>password.</code>	<code>password1</code> , <code>passwordx</code> , ...
<code>?</code>	Optional	<code>woodchucks?</code>	<code>woodchuck</code> , <code>woodchucks</code>
<code>*</code>	Repeat 0 or more times	<code>Woodchucks*</code>	<code>Woodchuck</code> , <code>Woodchucks</code> , <code>Woodchucksssss</code> , ...
<code>+</code>	Repeat 1 or more times	<code>Woo+dchucks</code>	<code>Woodchucks</code> , <code>Woodchucks</code> , <code>Woooodchucks...</code>

Building Blocks: Combining Patterns

Pattern	Role	Example	What does it match?
	Disjunction (logical OR)	<i>groundhog woodchuck</i>	<i>groundhog, woodchuck</i>
()	Sub-expression	<i>wood(chuck louse)</i>	<i>woodlouse, woodchuck</i>

Building Blocks: Start and End

Pattern	Role	Example	What does it match?
\b	Word boundary	\bround	<i>round</i> but not <i>ground</i>
^ outside []	Start of sentence	^[Rr]	<i>R</i> in <i>Round</i> or <i>r</i> in <i>round</i>
\$	End of sentence	.\$	<i>!</i> in <i>ground!</i> , <i>.</i> in <i>round.</i>

Constructing an RE: an Example

The word “the”

- Test it out at <https://www.whatsmyip.org/regular-expression-tester/>
- You can also try to write an RE for:
 - All strings with two consecutive repeated words, e.g., “the the”
 - All integers

Constructing an RE: an Example

The word “the”

- /the/

Constructing an RE: an Example

The word “the”

- /the/
- /[tT]he/ -- match capitalised/uncapitalised

Constructing an RE: an Example

The word “the”

- `/the/`
- `/[tT]he/`
- `^b[tT]he\b/` -- match only whole words

Constructing an RE: an Example

The word “the”

- `/the/`
- `/[tT]he/`
- `^b[tT]he\b/`
- `/[^a-zA-Z][tT]he[^a-zA-Z]/` -- match when “the” is preceded or followed by non-letter tokens like ‘_’ or space

Constructing an RE: an Example

The word “the”

- `/the/`
- `/[tT]he/`
- `\b[tT]he\b/`
- `/[^a-zA-Z][tT]he[^a-zA-Z]/`
- `/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/` -- Also match when the word occurs at the start or end of a line

False Positives and Negatives

- Each line tries to reduce false +ves and –ves
 - False positive (FP) = a match that was incorrect
 - False negative (FN) = incorrectly did not match

	True label = +ve	True label = -ve
Prediction = +ve	True positive (TP)	False positive (FP)
Prediction = -ve	False negative (FN)	True negative (TN)

- NLP systems make a trade-off between these apparently opposing metrics

Summary

- REs encode text patterns that we want to find and replace
- Wide ranging uses in feature extraction and preprocessing
- Often used to implement a baseline model for many text mining tasks
- Referring to the introduction: REs are a tool for hard-coding knowledge
- For more complicated tasks, we will need to use machine learning

Reading

- Dan Jurafsky and James H. Martin. Speech and language processing (3rd edition draft). **Chapter 2.**
- <https://web.stanford.edu/~jurafsky/slp3/>