University of Petra

307307
Introduction to Text Mining and Natural Language Processing

Introduction to Regular Expressions

Regular expressions are used everywhere

- Part of every text processing task
  - Not a general NLP solution (for that we use large NLP systems we will see in later lectures)
  - But very useful as part of those systems (e.g., for pre-processing or text formatting)
  - Necessary for data analysis of text data
- A widely used tool in industry and academics

Regular expressions

- A formal language for specifying text strings
- How can we search for mentions of these cute animals in text?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks
  - Groundhog
  - groundhogs

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any one digit

- Ranges using the dash [A-Z]

Pattern	Matches	
[A-Z]	An upper-case letter	Drenched Blossoms
[a-z]	A lower-case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

- Carat ^ as first character in [] negates the list
  - Note: Carat means negation only when it's first in []
  - Special characters (., *, +, ?) lose their special meaning inside []

Pattern	Matches	Examples
`[^A-Z]`	Not an upper-case letter	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`I have no exquisite reason”`
`[^.]`	Not a period	`Our resident Djinn`
`[e^]`	Either e or ^	`Look up ^ now`

Regular Expressions: Convenient aliases

Pattern	Expansion	Matches	Examples
\d	[0-9]	Any digit	Fahreneit 451
\D	[^0-9]	Any non-digit	Blue Moon
\w	[a-ZA-Z0-9_]	Any alphanumeric or _	Daiyu
\W	[^\w]	Not alphanumeric or _	Look!
\s	[\r\t\n\f]	Whitespace (space, tab)	Look␣up
\S	[^\s]	Not whitespace	Look up

Regular Expressions: More Disjunction

- Groundhog is another name for woodchuck!
- The pipe symbol | for disjunction

Pattern	Matches
groundhog|woodchuck	woodchuck
yours|mine	yours
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	Woodchuck

Wildcards, optionality, repetition: . ? * +

Pattern	Matches	Examples
beg.n	Any char	begin begun beg3n beg n
woodchucks?	Optional s	woodchuck woodchucks
to*	0 or more of previous char	t to too tooo
to+	1 or more of previous char	to too tooo toooo

Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ $

Pattern	Matches
`^[A-Z]`	Palo Alto
`^[^A-Za-z]`	1 “Hello”
`\.$`	The end.
`.$`	The end? The end!

A note about Python regular expressions

- Regex and Python both use backslash "\" for special characters. You must type extra backslashes!
  - `"\\d+"` to search for 1 or more digits
  - `"\n"` in Python means the "newline" character, not a "slash" followed by an "n". Need `"\\n"` for two characters.
- Instead: use Python's **raw string notation** for regex:
  - `r"[tT]he"`

The iterative process of writing regex's

- 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`

False positives and false negatives

The process we just went through was based on fixing two kinds of errors:

1. Not matching things that we should have matched (The)

False negatives

2. Matching strings that we should not have matched (there, then, other)

False positives

Characterizing work on NLP

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- Increasing coverage (or *recall*) (minimizing false negatives).
- Increasing accuracy (or *precision*) (minimizing false positives)

Regular expressions play a surprisingly large role

Widely used in both academics and industry

1. Part of most text processing tasks, even for big neural language model pipelines
   - including text formatting and pre-processing
2. Very useful for data analysis of any text data

Basic Text Processing

Regular Expressions

Basic Text Processing

More Regular Expressions: Substitutions and ELIZA

Substitutions

- Substitution in Python and UNIX commands:

- `s/regexp1/pattern/`
- e.g.:
- `s/colour/color/`

Capture Groups

- Say we want to put angles around all numbers:

the 35 boxes → *the <35> boxes*

- Use parens () to "capture" a pattern into a numbered register (1, 2, 3…)
- Use \1 to refer to the contents of the register

```
s/([0-9]+)/<\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
  - Joseph Weizenbaum, 1966.

- 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 \1/
- s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
- s/.* all .*/IN WHAT WAY?/
- s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/