WARWICK

CS918: LECTURE 1b

Introduction to Regular Expressions

Arkaitz Zubiaga, 3rd October, 2018

REGULAR EXPRESSIONS

WARWICK

- A formal language for specifying **text patterns**: for **searching or replacing text**.

- For example:
  - Search for all occurrences of ‘the’ in a text.

    The theatre is on the hill.

    We need to match ‘The’ and ‘the’, regardless of the case.

    We don’t want to match ‘theatre’.

REGULAR EXPRESSIONS

WARWICK

- A formal language for specifying **text patterns**: for **searching or replacing text**.

- For example:
  - Find URLs in a text.

    This is a tweet http://www.webpage.com/

    But URLs can vary a lot.

APPLICATIONS OF REGULAR EXPRESSIONS

WARWICK

- ELIZA: A simple, early chatbot from the 1960s.

- Using regular expressions, e.g.:

  When the user says “You are X”, ELIZA responds with “What makes you think I am X?”, for any X.

REGULAR EXPRESSIONS: SYNTAX

WARWICK

- Today, we will see:
  - Brackets: sets and ranges.
  - Negations.
  - Disjunction.
  - Repetition.
  - Anchors.
  - Special characters.

BRACKETS: SETS AND RANGES

- With **brackets**, we can indicate a **set of characters**, e.g.:

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

- As well as a **range of characters**, e.g.:

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`

NEGATIONS

- Caret (^) means negation **only** when first in [].

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`See here`
`[^e^]`	Neither e nor ^	`e^ps`
`a^b`	The pattern a caret b	`Look up a^b now`

DISJUNCTION

- Vertical bar (|) indicates **OR**.

Pattern	Matches
groundhog|woodchuck	groundhog woodchuck
yours|mine	yours mine
a|b|c (equivalent to [abc])	baby static electric
[gG]roundhog|[Ww]oodchuck	Groundhog groundhog Woodchuck woodchuck

REPETITION

- **?:** 0/1 occurrences.
- **+:** 1 or more occurrences.
- ***:** 0 or more occurrences.
- **{n}:** exactly n times.
- **{m, n}:** between m and n times.

Pattern	Matches	
`colou?r`	Optional previous char	color colour
`oo*h!`	0 or more of previous char	oh! ooh! oooh! ooooh! ...
`o+h!`	1 or more of previous char	oh! ooh! oooh! ooooh! ...
`baa+`	1 or more	baa baaa baaaa baaaaa ...
`ba{5}`	5 times	baaaaa
`ba{2,4}`	2 to 4 times	baa baaa baaaa

SPECIAL CHARACTERS

- **. (period):** matches any character (generally except new line).
- **\ (back slash):** escape a special character.
- **\b:** word boundary.
- **():** group characters.

Pattern	Matches	
`.`	Any character	`Anything` `*look`
`\.`	A period	`Hello.`
`\*`	An asterisk	`A*`
`.*`	Any sequence	`daf734*DVA`
`\b[Tt]he\b`	The word 'the' (or 'The')	`Then the theatre`
`(th)*`	Repetitions of th	`athththa`

ANCHORS

- Beginning (^) and end ($) of line.

Pattern	Matches
`^[A-Z]`	Coventry
`^[^A-Za-z]`	1 "Hello"
`\.$`	The end.
`.$`	The end? The end!

HOW TO WRITE REGULAR EXPRESSIONS

WARWICK

- Example: find numbers in a text, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

HOW TO WRITE REGULAR EXPRESSIONS

WARWICK

- Example: look for numbers in texts, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

- First, we need to think of repetitive and optional patterns:
  - After a comma, there must be 3 digits.
  - Commas are optional and unlimited.
  - Before a comma, we can have at most 3 digits.

HOW TO WRITE REGULAR EXPRESSIONS

- Example: look for numbers in texts, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

- after comma, 3 digits; commas, optional & unlimited; before comma, 1-3 digits.

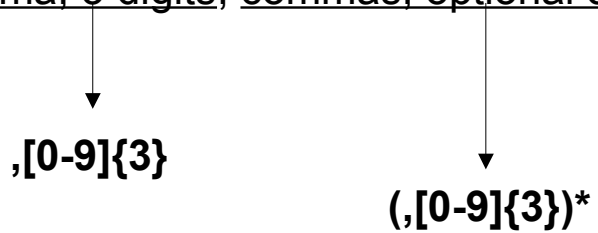
,[0-9]{3}

HOW TO WRITE REGULAR EXPRESSIONS

WARWICK

- Example: look for numbers in texts, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

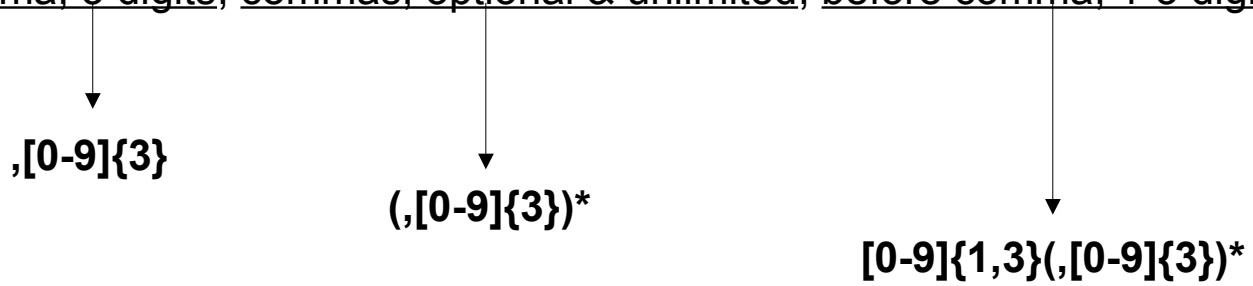
- after comma, 3 digits; commas, optional & unlimited; before comma, 1-3 digits.



HOW TO WRITE REGULAR EXPRESSIONS

- Example: look for numbers in texts, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

- after comma, 3 digits; commas, optional & unlimited; before comma, 1-3 digits.



HOW TO WRITE REGULAR EXPRESSIONS

- Example: look for numbers in texts, which can have thousand separators, e.g.:
  - **65,640,000** people live in the UK's **4** nations.

- after comma, 3 digits; commas, optional & unlimited; before comma, 1-3 digits.

,[0-9]{3}

(,[0-9]{3})*

[0-9]{1,3}(,[0-9]{3})*

Or even: **[1-9][0-9]{0,2}(,[0-9]{3})*** **(if the 1st number can't be a 0!)**

REGULAR EXPRESSIONS: ERRORS

- We can make 2 types of errors:
  - False positives (Type I errors).

    instances we have output, but shouldn't.

  - False negatives (Type II errors).

    instances we haven't output, but should.

FALSE POSITIVES: TYPE I ERRORS

- Instances that **should not be output**.
- For instance, if we search for “[Tt]he” (Type I errors highlighted in red):

  There are 10 people in the room, they all have a laptop with them.

FALSE NEGATIVES: TYPE II ERRORS

- Instances that **have been missed**.
- For instance, if we search for “the”:

  The laptop is in the kitchen.

  we’ve missed it (Type II error)

EVALUATION

- We'll see these kinds of errors throughout the module.

- We want to achieve **two antagonistic goals**:
  - Minimise false positives (i.e. increase **precision**)
  - Minimise false negatives (i.e. increase coverage or **recall**).

EVALUATION: PRECISION AND RECALL

Example: *There are 10 people in the room, they all have a laptop with them.*

- $$\text{Precision} = \frac{tp}{tp + fp}$$ (ratio of correct items among those output)

  ¼ = 0.25

- $$\text{Recall} = \frac{tp}{tp + fn}$$ (ratio of reference items that have been output)

  1/1 = 1

EVALUATION: F1 SCORE

We want to optimise for both precision and recall:

- $F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$ (harmonic mean of precision and recall)

Equation as follows, however generally ß = 1:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

REGULAR EXPRESSIONS: REFERENCES

- Regular expressions with Python:
  https://docs.python.org/3.7/howto/regex.html

- Testing regular expressions online:
  https://regex101.com/

RESOURCES

- Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 3rd edition. **Chapters 1-2.**

- Bird Steven, Ewan Klein, and Edward Loper. Natural Language Processing with Python. O'Reilly Media, Inc., 2009. **Chapters 1-3.**