CS 333: Natural Language Processing

Fall 2025

Prof. Carolyn Anderson
Wellesley College

Text Processing

Basic Text Processing

Today we're going to focus on **finding** and **extracting** text using pattern-matching.

Python (like many other languages) provides a regular language for matching strings: regular expressions.

Regular Expressions

Regular expressions are a formal language for specifying strings of text.

Assume we're interested in searching a collection of novels for all mentions of cookies.

Minimally, we'd want to match:

- cookie
- cookies
- Cookie
- Cookies

Disjunction

We can use square brackets [] to match one of a set of characters. We can also express a range this way.

\d : any digit (equivalent to [0-9])

Matches	Pattern
cookie, Cookie:	[cC]ookie
Any digit:	[0-9]
Any uppercase letter:	[A-Z]

Negation

We can negate individual characters with ^. But it must be first within square brackets.

Matches	Pattern
Non-uppercase letter	[^A-Z]
Anything other than *c* or C	[^Cc]
Neither *c* nor ^	[^c^]

Disjunction continued

For longer terms, we can use | for disjunction. Some old recipes use the spelling *cooky*.

Matches	Pattern
cooky, cookie	cooky | cookie
a or *b* or *c*	a|b|c [abc]
letter or digit	[Aa-Zz]|[0-9]

Quantifiers

Sometimes we want to repeat terms multiple times.

cookies*   matches: cookie   cookies   cookiesss

Matches	Pattern
cookies, cookie	cookies?
cookies with 0 or more !	cookies!*
cookies with 1 or more !	cookies!+ cookies!!*

Wildcard

We can also match on any character using .

to escape:

\.

.* : any number of any characters

Matches	Pattern
cookie?, cookie!, cookies …	cookie.
rose, nose, …	.ose

Anchors

We can match the beginning or end of a string using ^ and $.

cookie ^cookie$

Matches	Pattern	Doesn't Match
cookie?, cookie jar	^cookie.*	*I want a cookie*
I want a cookie	.*cookie$	*cookie jar, cookies?*

Exercise

Work with the person next to you to come up with a regular expression to match all instances of the word *the* in a text.

Evaluation

What's wrong with this solution?

Error Types

False positives: we incorrectly matched words other than *the*

(For example, *The)*

False negatives: we missed some true instances of *the*

(For example, *theta* or *theorem)*

Error Types

	TRUTH: the	TRUTH: not the
Guess: the	True positive	False positive
Guess: not the	False negative	True negative

Error Types

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

Making progress often involves trying to optimized for two conflicting objectives:

- minimizing false positives Precision
- minimizing false negatives Recall

the

Capture Groups

Say we want to put dollar signs around all numeric terms: *5.25 —> 5.25*.

We can use parentheses to "capture" terms and refer back to them with back-references: \1 , \2, etc:

```
myString = "35th place"
                   \d
x = re.sub(r"([0-9]+)", r"$\1$","35th place")
print(x)
$35$th place
```

Capture Groups

We can use parentheses to "capture" terms and refer back to them with back-references: \1 , \2, etc.

Matches	Pattern	Doesn't Match
the faster they ran, the faster we ran	the (.*)er they (.*), the \1er we \2	*the faster they ran, the faster we ate*

Non-Capture Groups

It is also handy to group expressions without capturing them. For instance, a more elegant disjunction for *cookie* | *cooky:* (?:

Matches	Pattern
cooky, cookie	(?:cook) y | ie
some cats like some cats	(?: some | a few) (people | cats) like some \1

a few cats like some cats   some people like some people

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

Matches	Pattern
at the line start, any single word that doesn't start with *Volcano*	start of line ^(?!Volcano)[A-Za-z]+

Regular Expression Applications: ELIZA

ELIZA (Weizenbaum 1966) was a simple early NLP system that imitated a Rogerian psychotherapist

Uses pattern matching to match:

- *I need X*

and translates into, e.g.

- *What would it mean to you if you got X?*

Regular Expression Applications: 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

Regular Expression Applications: ELIZA

Under the hood:

```
sub: .* I'M (depressed|sad)
with: .*/I AM SORRY TO HEAR YOU ARE \1

sub: .* I AM (depressed|sad)
with: .*/WHY DO YOU THINK YOU ARE \1

sub: .* all .*
with IN WHAT WAY?

sub: .* always .*
with CAN YOU THINK OF A SPECIFIC EXAMPLE?
```

Other Key Text-Processing Skills

- list comprehensions
- sorting with lambda
- strip(), replace(), split()
- sys.argv for reading in command-line arguments
- json and CSV for reading/writing data

Homework 1

- Part 1: regex practice
  - feel free to use online platforms for regex testing (but not regex generation)
- Part 2: extending our poetry chatbot
  - utilize different regex features
  - implement basic sentiment analysis
  - add length preferences