NLP For Economists

Lecture 2: Python and Text- An Overview

MGSE - LMU Munich
Guest Course, October 2022

6th October 2022

Session Outline

- Python - an overview (Notebook)
- Python and Text - an overview (Notebook)
- Regular expressions in Python (slides)

Regular Expressions

1. Regular expressions can be described as a system of creating rules to specify patterns that can be extracted from data.
2. Where are regular expressions useful?

Regular Expressions

1. Regular expressions can be described as a system of creating rules to specify patterns that can be extracted from data.
2. Where are regular expressions useful?
   - Searching inside texts, information extraction from texts
   - Also used extensively in theoretical computer science (we are not concerned with this aspect in this class!)
   - Substituting one pattern with another.

Regular Expressions

1. Regular expressions can be described as a system of creating rules to specify patterns that can be extracted from data.
2. Where are regular expressions useful?
   - Searching inside texts, information extraction from texts
   - Also used extensively in theoretical computer science (we are not concerned with this aspect in this class!)
   - Substituting one pattern with another.
3. Every commonly used programming language supports regular expressions.
4. Unix based operating systems have pre-installed terminal based tools such as grep, egrep etc. that allow you to use regular expressions for text processing.

RegEx for text processing

What do we need?

- A corpus of texts (or a single text)
- A description of what we want to search for or extract
- A pattern that meets this description.

Simple Patterns

- Plain sequence of characters: a pattern "Python" matches all occurrences of Python in text.
- Regular expressions are case-sensitive. To match "python" and "Python", you should have a pattern: [pP]ython.
- the pattern [abc] matches a or b or c. [pP]ython matches python or Python.
- patterns [a-z], [A-Z] match all lower and upper case characters respectively. [0-9] matches digits.

Use of special characters in RegEx

- caret: [ˆX] matches any single character that is not X. If the caret occurs anywhere else in the sequence, it is treated as a caret symbol.
- asterisk: zero or more occurrences of something. "ba*" matches b, ba, baa, baaa etc.
- plus: one or more occurrences of something. "ba+" matches ba, baa, baaa etc. "(ba)+" matches ba, baba, bababa ..
- question mark: the pattern "questions?" catches question and questions.
- period: just a "." matches everything. To match a period, you have to use "\.". "p.n" matches any character between p and n in a text.
- .*: matches all characters. "p.*n" matches all characters between p and n.

"Anchor" characters

- caret, without [], when put in front of a pattern matches the beginning of a line. "^The" matches lines starting with "The".
- $ matches the end of the line. "Dog\.$" matches lines that end in "Dog."
- \b matches word boundary, \B matches non-boundary. E.g., "\bthe\b" matches "the" but not "other".
- |(pipe symbol): is used to represent "or" operation. "cat|dog" matches "cat" or "dog".
- pupp(y|ies) matches puppy or puppies.

Advanced operators

- \d matches any digit. \D matches any non-digit.
- \w matches any alpha numeric character or underscore. \W matches anything other than alphanumeric characters or underscores.
- \s matches whitespace. \S matches any non-white space character.
- \n matches newline.
- \t matches tab.

The use of {}

- {n} indicates n occcurences of a previous character/expression. "a{2}" matches aa.
- {n,} indicates n or more occurrences of a previous character or expression. "a{2,}" matches aa, aaa etc.
- {m,n} indicates m to n occurrrences. a{2,5} matches 2 to 5 occurrences of a's together (aa, aaa, aaaa, aaaaa)

Substitution and number operator

- We can substitute one pattern with another. E.g., s/colour/color substitutes colour with color (syntax is for illustration. Works with some languages, may not work with python)
- An operator \1 is used in regular expression syntax, to refer to a previous part of the full expression.
- For example, consider this pattern: s/([0-9]+)/<\1> replaces 99 with $< 99 >$.
- Such numbered patterns are "memorized" and are called registers. You can have \1, \2 etc in complex patterns. Anything within () counts as one register.

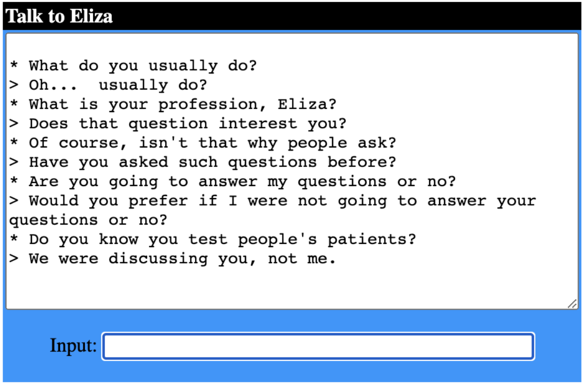
Substitution and number operator

- These operators are very useful in creating canned responses for standard question forms.
- Sometimes, they create an impression of real natural language understanding happening behind screen.

Regular Expressions: Eliza

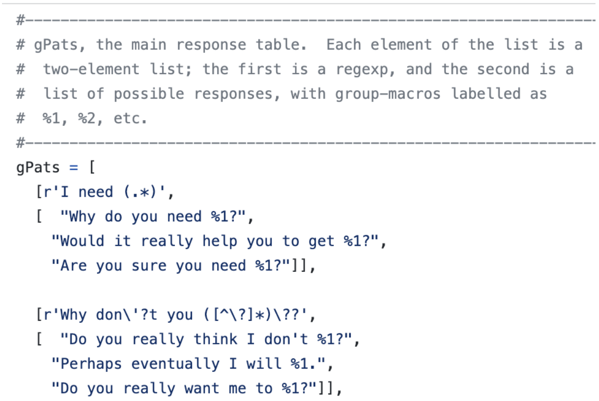
- Eliza was an early NLP program from 1960s.
- It simulated conversation between a human and a machine, using pattern machine techniques.
- One version imitates a Rogerian Psychotherapist (demo in next slide)
- To think that all that can be done just with what we discussed in the class so far is amazing :-)

Eliza - Demo



```
* What do you usually do?
> Oh...  usually do?
* What is your profession, Eliza?
> Does that question interest you?
* Of course, isn't that why people ask?
> Have you asked such questions before?
* Are you going to answer my questions or no?
> Would you prefer if I were not going to answer your
questions or no?
* Do you know you test people's patients?
> We were discussing you, not me.
```

Eliza - Code snippet



```
#-----------------------------------------------------------------
# gPats, the main response table.  Each element of the list is a
#  two-element list; the first is a regexp, and the second is a
#  list of possible responses, with group-macros labelled as
#  %1, %2, etc.
#-----------------------------------------------------------------
gPats = [
  [r'I need (.*)',
  [  "Why do you need %1?",
    "Would it really help you to get %1?",
    "Are you sure you need %1?"]],

  [r'Why don\'?t you ([^\?]*)\??',
  [  "Do you really think I don't %1?",
    "Perhaps eventually I will %1.",
    "Do you really want me to %1?"]],
```

source: https://github.com/jezhiggins/eliza.py/blob/main/eliza.py

Practice writing RegEx

source: Exercise 2.1 in J&M

Let us say these are our requirements. How do we write a regex for each case?

1. all lowercase alphabetic strings ending in b.
2. All lines that start at the beginning of the line with a number, and that end with a word.
3. All lines that have both the words "the" and "of" in them (but not "then", they", "often" etc)
4. all strings with two consecutive repeated words ("big big" but not "big bug")

Solutions

- all lowercase alphabetic strings ending in b:

  `[a-z]*b\b`

- all lines that start at the beginning of the line with a number, and that end with a word:

  `^\d.*\b[a-zA-Z]+\.$`

- all lines that have both the words "the" and "of" in them (but not "then", they", "often" etc):

  `\bthe\b.*\bof\b`

- all strings with two consecutive repeated words ("big big" but not "big bug"):

  `\b(\w+)\s\1 (not \b(\w+)\b\1. Why?)`

Python and Regular Expressions-1

- re is the python library that supports processing with regular expressions (import re)
- re.compile(*some pattern*) is used to compile a pattern into a "pattern" object, and use the pattern again in the program.
- re.search(pattern,string,*flags) is used to search for the first location of a pattern in a given string.
- re.match(same params) is similar to search(), but only matches the pattern at the start of the string.
- re.fullmatch(same params): shows a match only if the full string matches with the pattern.
- Important flags: re.MULTILINE (matches regular expressions looks for matches at each line), re.DOTALL (includes newlines in matching).

Refer: https://docs.python.org/3/library/re.html

Python and Regular Expressions-2

- re.findall(pattern,string,*flags): finds all matches for a pattern, and returns a list.
- re.sub(pattern,replacement,string,*flags): Replace one pattern with another. Returns the new string with replacements.
- re.subn(same params): Same as sub() but returns a tuple (new_string, num. of replacements made).
- Tip: Use of ? after .* in Python regular expressions lets you match shortest matches. Otherwise, python matches longest possible match by default.
- re.split() - similar to split() of strings, but accepts patterns along with plain strings.

RegEx - some examples

```
import re

fh = open("example.txt")

#Prints lines having "th" after a space
for line in fh:
    if re.search(" th",line):
        print(line)
        print()

#Prints a list of matches spanning multiple lines
content = open("example.txt").read()
temp = re.findall("sometimes.*difficult",
            content,re.DOTALL)
#remove re.DOTALL and try.
print(temp)
```

RegEx - some examples

Continuation from previous slide

```
#re.MULTILINE overview
temp2 = re.findall("^This",content,re.MULTILINE)
#Remove re.MULTILINE or replace with re.DOTALL and try.
print(temp2)

#re.sub() overview
temp3 = re.sub("Th", "HA", content)
#replace sub with subn and try
print(temp3)

#For more details on other re functions, visit:

#https://docs.python.org/3/library/re.html
```

RegEx - Programming practice

All wikipedia pages have links in their side panel, that links to the versions of an article in other languages. Write a Python program that uses regular expressions and string functions, and prints these links.

A solution

```
import re
from urllib.request import urlopen

def get_input_url():
    link = input("Enter the wikipedia url: ")
    p = re.compile("<li class=\"interlanguage-link.*?>.*?</a></li>")
    #why the question mark after .*?
    #How to get this pattern above?
    #Look at the source of HTML from browser.
    try:
        html = urlopen(link).read().decode(encoding="utf8").strip()
        temp_list = re.findall(p, html)
        for item in temp_list:
           #print(item)
           #printing this told me what I should split.
           my_link = item.split("a href=\"")[1].split(" ")[0]
           #What is this???
           print(my_link)
    except:
        print("something is wrong. Quitting the program")
        exit()
```

Summary

- There is far more than what I just introduced, and regex is a very useful tool while doing NLP
- Even though it would seem like they have no role in modern NLP dominated by deep learning, they do.
- Look inside the code of popular NLP libraries. You will see a regex somewhere!
- Sometimes, knowing the right regex will save a lot of time and effort.

Tomorrow's Session

- Overview of various NLP methods
- Hands on + lecture
- ToDo: Try to talk to your friends and choose a team of 2-3 people for group discussion. You can also present individually, if you want. Let me know whatever you choose, by Monday (10th October 2022).
- The above item counts for your grade.