LING 520: Computational Analysis of English

Semester: FALL '16

Instructor: Sowmya Vajjala

Iowa State University, USA

6 September 2016

Class outline

- Review of last week
- Some instructions about Assignment 1
- Regular expressions - review
- Regular expressions practice exercises

Last Week

- NLTK installation: was everyone successful? Did everyone manage to go through examples in Chapter 1 of NLTK book?
- Problem Set 1: Questions?
- Assignment 1 progress?
- Any other questions?

Assignment 1: Some instructions

- Your program has to run without throwing any errors, and show some output when I give input. This is non-negotiable. Any program that throws syntax errors and the likes will get a 0 score.
- If a program is not perfect, works for some cases, it is tolerable. You may or may not get full credit, but will never get 0.
- Submit in the format I asked, not in the format of your choice.
- When I said input sample.txt, don't imagine I will name my file sample.txt and keep it in the same directory where I run your program. Your programs should accept file path from the user as input.

Regular Expressions

1. How would you define regular expressions?

1. How would you define regular expressions?
    - It is a language to describe patterns in textual data.
2. Where are regular expressions useful?

1. How would you define regular expressions?
   - It is a language to describe patterns in textual 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.

1. How would you define regular expressions?
   - It is a language to describe patterns in textual 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 - a brief history

- First described by Stephen Kleene in 1950s. Original purpose: to describe finite state machines, formal languages etc.
- Ken Thompson, in 1968, used this concept to match patterns in text files, in an early text editor.
- By 70s, they became commonly used methods for text processing on Unix systems (most of these tools are still available as terminal based apps. in MacOS and other Unix systems)
- Small differences exist, but largely, RegEx syntax remains the same across languages.

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: [^X] 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.
- Best example: Eliza program. http://goo.gl/lBDD2n
- Another, slightly more recent one: Alice bot http://goo.gl/0tulbW
- Implementing Eliza in Python: http://goo.gl/nREmwN

Practice writing RegEx

source: Exercise 2.1 in J&M

Go to Pythex.org or pyregex.com or any such regular expression tester online. Choose any text you want, and write regular expressions for the following:

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.

An example Python program

RegExOverview.py on Blackboard.

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.

Next class

- Topics: Regular Expressions continued. Tokenizing, Sentence Splitting
- Videos: Week 2, video 7 in Radev's coursera course (12 min); Two videos from another NLP course by Jurafsky and Manning (20 min total) - All uploaded on Blackboard.
- Assignment 1 - submission deadline towards the end of next week!