Handout 9: Regular Expressions, Pre-Processing

Regular expressions

1. Regular expressions are **patterns** that match some strings and not others

a. Does pattern "a" match "cat"?

```
>>> import re
>>> m = re.search(r'a', 'cat')
>>> m
<_sre.SRE_Match object; span=(1, 2), match='a'>
>>> m.group()
'a'
>>> m.span()
(1, 2)
```

b. Contrast

```
>>> re.search(r'b', 'cat')
>>>
```

c. Strings of form `r'...'`

```
>>> len(r'\n')
2
>>> r'\n'[0]
'\\'
>>> len('\n')
1
```

d. Collecting hits: `[w for w in words if re.search(pat, w)]`

e. Returns only the first match

2. Motivating examples

a. Searching in treebank

```
>>> from nltk.corpus import treebank
>>> wsj = sorted(set(treebank.words()))
>>> def grep (pat):
...     return [w for w in wsj if re.search(pat, w)]
...
```

b. Some patterns

```
>>> grep(r'zz')
['Lazzaroni', 'Muzzling', 'buzz', 'fizzled', 'muzzling', 'puzzled']
>>> grep(r'ou?r')
['14-hour', '36-store', '87-store', '90-cent-an-hour', ...]
>>> grep(r't[aeiou]c')
```

```
['20-stock', '500-Stock', '500-stock', 'Article', ...]
>>> grep(r't[aeiou]{2,}c')
['schoolteacher', 'teach', 'teacher', 'teacher-cadet', ...]
>>> grep(r'\W\w+\W')
['*EXP*-1', '*EXP*-2', '*EXP*-3', '*ICH*-1', '*ICH*-2', ...]
>>> grep(r'\d[A-Za-z]')
["'30s", "'40s", "'50s", "'80s", '10th', '11th', ...]
```

3. Give examples of strings matched by:

- **a.** `ab`
- **b.** `a|b`
- **c.** `ab|c`
- **d.** `a(b|c)`
- **e.** `a+b`
- **f.** `a*b`
- **g.** `ab*`
- **h.** `(ab)*`
- **i.** `a?b`
- **j.** `ab?`
- **k.** `((ab)+)?`

4. Counting

- **a.** `a{2}`
- **b.** `a{2,4}`
- **c.** `a{2,}`
- **d.** `a{,4}`

5. Character classes

- **a.** `[ab]c`
- **b.** `c[aeiou]t`
- **c.** `[a-z]c`
- **d.** `[a-z0-9]c`
- **e.** `[-z]c`
- **f.** `a-zc`
- **g.** `(a-z)c`
- **h.** `a.b`
- **i.** `a.*b`
- **j.** `a\.*b`

6. Common classes

a. `\w` – letters, digits, underscore

b. `\d` – digits only

c. `\s` – whitespace (space, tab, newline, carriage return, vertical tab, formfeed)

d. Complements: `\W`, `\D`, `\S`

7. Backslash escapes

a. `r'\n'` is a two-character sequence, not a newline character. But as an RE, it matches the newline character.

b. Ditto for `r'\t'`, `r'\r'`

8. Anchors

a. `^ab`

b. `ab$`

c. `^ab$`

d. Beginning of string versus beginning of line:

```
>>> re.search(r'^a', 'this\naardvark')
>>> re.search(r'^a', 'this\naardvark', re.M)
<_sre.SRE_Match object; span=(5, 6), match='a'>
```

9. Findall, finditer

```
>>> re.findall(r'a', 'baa')
['a', 'a']
>>> for m in re.finditer(r'a', 'baa'):
...     print m.span()
...
(1, 2)
(2, 3)
```

10. Word boundaries

a. Examples

```
>>> re.findall(r'\b.', 'eat kids')
['e', ' ', 'k']
>>> re.findall(r'\b.', 'eat, kids!')
['e', ',', 'k', '!']
```

b. `\b` matches transition from word to nonword or vice versa. “Word” = alpha, digit, underscore.

```
>>> re.findall(r'\b.', 'eat_20, kids!')
['e', ',', 'k', '!']
```

11. Greediness

a. `re.findall(r'a+', 'baaa')` → `['aaa']`

b. `re.findall(r'a+?', 'baaa')` → `['a', 'a', 'a']`

c. `re.findall(r'a{1,2}', 'baaa')` → `['aa', 'a']`

d. `re.findall(r'a{1,2}?', 'baaa')` → `['a', 'a', 'a']`

12. Grouping

```
>>> m = re.search(r'(.)([aeiou])', 'tacoma')
>>> m.group(0)
'ta'
>>> m.group(1)
't'
>>> m.group(2)
'a'
```

13. Findall returns group 0 only if there are no explicit parens

a. Example

```
>>> re.findall(r'(a|o)c', 'tock')
['o']
```

b. If there are parentheses, `findall` returns tuple with value of each

```
>>> re.findall(r'.[aeiou]', 'tacoma')
['ta', 'co', 'ma']
>>> re.findall(r'(.)[aeiou]', 'tacoma')
['t', 'c', 'm']
>>> re.findall(r'(.)([aeiou])', 'tacoma')
[('t', 'a'), ('c', 'o'), ('m', 'a')]
>>> re.findall(r'((.)([aeiou]))', 'tacoma')
[('ta', 't', 'a'), ('co', 'c', 'o'), ('ma', 'm', 'a')]
```

c. Sometimes we need parentheses but not sub-expressions

```
>>> re.findall(r'(?:a|o)c', 'tock')
['oc']
>>> re.findall(r'.(aw|ee)', 'pawnee')
['aw', 'ee']
>>> re.findall(r'.(?:aw|ee)', 'pawnee')
['paw', 'nee']
```

14. Split

```
>>> re.split(r'[^aeiou]', 'alberta')
['a', '', 'e', '', 'a']
>>> re.split(r'[^aeiou]+', 'alberta')
['a', 'e', 'a']
```

Processing pipeline

15. The typical pipeline

- **a.** Fetch text from file or web
- **b.** Eliminate HTML mark-up, if necessary
- **c.** **Sentence segmentation**
- **d.** **Tokenization**
- **e.** Part-of-speech tagging
- **f.** Named-entity tagging
- **g.** Parsing
- **h.** To extract terms for information retrieval, or features for machine learning: **stemming**

16. Sentence segmentation

- **a.** A first step in processing raw text
- **b.** NLTK segmenter is `sent_tokenize`:

```
>>> from nltk import sent_tokenize
>>> raw = '''Mr. J.R. Reynolds purchased A.B.C.  On Sunday,
... he went golfing.  Did it rain?  No!'''
>>> sent_tokenize(raw)
['Mr. J.R. Reynolds purchased A.B.C.', 'On Sunday,\nhe went golfing.',
'Did it rain?', 'No!']
```

17. Tokenization

```
>>> s = 'Hi, Hugh!'
>>> s.split()
['Hi,', 'Hugh!']
>>> re.split(r'\s', s)
['Hi,', 'Hugh!']
>>> re.split(r'\W+', s)
['Hi', 'Hugh', '']
>>> from nltk import word_tokenize
>>> word_tokenize(s)
['Hi', ',', 'Hugh', '!']
```

18. Using `regexp_tokenize`

- **a.** Allowing spaces and comments in REs: `(?x)`

```
>>> re.findall(r'  [aeiou]  # test', 'hi')
[]
>>> re.findall(r'(?x)  [aeiou]  # test', 'hi')
['i']
```

b. Example

```
>>> pattern = r'''(?x)
... ([A-Z]\.)+ |
... \w+(-\w+)* |
... \w+n't |
... \$?\d+(\.\d+)?%? |
... \.\.\.+ |
... \S'''
>>> from nltk import regexp_tokenize
>>> regexp_tokenize('A.B.C. up $2.50 on 10-10-15.')
['A.B.C.', 'up', '$2.50', 'on', '10-10-15', '.']
```

19. Stemming = stripping suffixes

a. `re.findall(r'^(.*?)(ing|ly|ed|es|s)$', word)`

b. What does the output look like?

c. Why the anchors?

d. Why the question mark after the star?

e. Screws up on some examples

20. Pre-packaged stemmers

a. Porter stemmer:

```
>>> from nltk import PorterStemmer
>>> s = PorterStemmer()
>>> s.stem('kites')
'kite'
>>> s.stem('churches')
'church'
>>> s.stem('volitional')
'volit'
```

b. Similar: `LancasterStemmer`

c. Similar, but method is `lemmatize`: `WordNetLemmatizer`