Lab Session IV (b)

Dr. Jasmeet Singh,
TIET

Applications of Regular Expressions

- Apart from checking whether a regular expression matches a word, we can use regular expressions to extract material from words, or to modify words in specific ways.
- Regular Expressions can be used for:
  - **Extracting Word Pieces**
  - **Finding Word Stems**
  - **Searching Tokenized Text**

Extracting Word Pieces

- The re.findall() (“find all”) method finds all (non-overlapping) matches of the given regular expression.
- The following examples find all the vowels in a word, then count them:
  - >>> word = 'supercalifragilisticexpialidocious'
  - >>> re.findall(r'[aeiou]', word)
  - >>> len(re.findall(r'[aeiou]', word))

Extracting Word Pieces Contd…..

The following example look for all sequences of two or more vowels in some text, and determine their relative frequency:

```
>>>wsj=sorted(set(nltk.corpus.treebank.words()))
>>> fd=nltk.FreqDist(w for w in wsj
...      for w in re.findall(r'[aeiou]{2,}', w))
>>> fd.items()
```

Extracting Word Pieces Contd…..

- Regular expressions can be combined with conditional frequency distributions.
- The next example extract all consonant-vowel sequences from the words of austen emma.txt, such as ka and si and tabulate the frequency of each pair
- >>> rotokas_words = nltk.corpus.toolbox.words('rotokas.dic')
- >>> cvs = [cv for w in rotokas_words for cv in re.findall(r'[ptksvr][aeiou]', w)]
- >>> cfd = nltk.ConditionalFreqDist(cvs)
- >>> cfd.tabulate()

Finding Stems

- We can use regular expressions to find all the specified suffixes and remove those suffixes from the input string to return the stem.
- The following examples shows the function for stemming:

```
>>> def stem(word):
... regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
... stem, suffix = re.findall(regexp, word)[0]
...return stem
...
 words=nltk.corpus.gutenberg.words('austen-emma.txt')
[stem(w) for w in words]
```

- The regular expression removed the s from ponds but also from is and basis. It produced some non-words, such as distribut and deriv, but these are acceptable stems in some applications.

Searching Tokenized Text

- You can use a special kind of regular expression for searching across multiple words in a text (where a text is a list of tokens).
- The angle brackets are used to mark token boundaries, and any whitespace between the angle brackets is ignored.
- In the following example, <.*> , which will match any single token coming in between <a> and <man>
- >>> words = nltk.Text(nltk.corpus. gutenberg.words('melville-moby_dick.txt'))
- >>> words.findall(r"<a> (<.*>) <man>")
- The next example finds three-word phrases ending with the word bro:
- >>> chat = nltk.Text(nltk.corpus.nps_chat.words())
- >>> chat.findall(r"<.*> <.*> <bro>")

Searching Tokenized Text Contd…

- The next example finds sequences of three or more words starting with the letter l:
- >>> chat.findall(r"<l.*>{3,}")
- The next example find for expressions of the form x and other ys allows us to discover hypernyms:
- >>> hobbies_learned = nltk.Text(nltk.corpus.brown.words(categories=['hobbies', 'learned']))
- >>> hobbies_learned.findall(r"<\w*> <and> <other> <\w*s>")

Normalizing Text-Stemmers

- NLTK includes several off-the-shelf stemmers
- We should use one of these in preference to crafting our own using regular expressions, since NLTK's stemmers handle a wide range of irregular cases.
- >>> words=nltk.corpus.gutenberg.words('austen-emma.txt')
- >>> porter = nltk.PorterStemmer()
- >>> lancaster = nltk.LancasterStemmer()
- >>> [porter.stem(t) for t in words]
- >>> [lancaster.stem(t) for t in words]

Normalizing Text-Lemmatizers

- The WordNet lemmatizer removes affixes only if the resulting word is in its dictionary.
- This additional checking process makes the lemmatizer slower than the stemmers.
- >>> wnl = nltk.WordNetLemmatizer()
- >>> [wnl.lemmatize(t) for t in tokens]

Regular Expressions for Tokenizing Text

Tokenization is the task of cutting a string into identifiable linguistic units that constitute a piece of language data.

- The very simplest method for tokenizing text is to split on whitespace.
- Consider the following text from Alice's Adventures in Wonderland:
- raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
- ... though), 'I won't have any pepper in my kitchen AT ALL. Soup does very
- ... well without--Maybe it's always pepper that makes people hot-tempered,'…"""
- We could split this raw text on whitespace using raw.split() .
- To do the same using a regular expression, it is not enough to match any space characters in the string (as in 1) , since this results in tokens that contain a \n newline character; instead, we need to match any number of spaces, tabs, or newlines (as in 2):
- re.split(r' ', raw) ……. (1)
- re.split(r'[\t\n]+', raw) ……. (2)
- The second statement in the preceding example can be rewritten as re.split(r'\s+', raw)

Regular Expressions for Tokenizing Text Contd….

- An alternative is to use the spiting on white space that Python provides us with a character class \w for word characters,equivalent to [a-zA-Z0-9_] .
- It also defines the complement of this class, \W , i.e., all characters other than letters, digits, or underscore. We can use \W in a simple regular expression to split the input on anything other than a word character:
- >>> re.split(r'\W+', raw)
- This gives us empty strings at the start and the end.
- With re.findall(r'\w+', raw) , we get the same tokens, but without the empty strings, using a pattern that matches the words instead of the spaces.

Regular Expressions for Tokenizing Text Contd….

- The \w+ in the preceding expression can be generalized to permit word-internal hyphens and apostrophes: « \w+([-']\w+)* »
- This expression means \w+ followed by zero or more instances of [-']\w+ ; it would match hot-tempered and it's.
- We need to include ?: in this expression for greedy searching. We have to also add a pattern to match quote characters so these are kept separate from the text they enclose.
- print re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw)
- The expression in this example also included « [-.(]+ », which causes the double hyphen, ellipsis, and open parenthesis to be tokenized separately

Regular Expressions for Tokenizing Text Contd….

- The following table lists the regular expression character class symbols:

Symbol	Function
\d	Any decimal digit (equivalent to [0-9])
\D	Any non-digit character (equivalent to [^0-9])
\s	Any whitespace character (equivalent to [\t\n\r\f\v]
\S	Any non-whitespace character (equivalent to [^ \t\n\r\f\v])
\w	Any alphanumeric character (equivalent to [a-zA-Z0-9_])
\W	Any non-alphanumeric character (equivalent to [^a-zA-Z0-9_])
\t	The tab character
\n	The newline character

NLTK's Regular Expression Tokenizer

- The function nltk.regexp_tokenize() is similar to re.findall()
- However, nltk.regexp_tokenize() is more efficient for this task, and avoids the need for special treatment of parentheses.
- The special (?x) "verbose flag" tells Python to strip out the embedded whitespace and comments.

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)                  # set flag to allow verbose regexps
...    ([A-Z]\.)+                    # abbreviations, e.g. U.S.A.
...    | \w+(-\w+)*                  # words with optional internal hyphens
...    | \$?\d+(\.\d+)?%?            # currency and percentages, e.g. $12.40, 82%
...    | \.\.\.                      # ellipsis
...    | [][.,;"'?():-_`]            # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Sentence Segmentation

- Tokenization is an instance of a more general problem of segmentation.
- Before tokenizing the text into words, we need to segment it into sentences. NLTK facilitates this by including the Punkt sentence segmenter (Kiss & Strunk, 2006).

```
>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = sent_tokenizer.tokenize(text)
```

Writing Results to File

- The following code opens a file output.txt for writing, and saves the program output to the file.
- >>> output_file = open('output.txt', 'w')
- >>> words = set(nltk.corpus.genesis.words('english-kjv.txt'))
- >>> for word in sorted(words):
- … output_file.write(word + "\n")
- When we write non-text data to a file, we must convert it to a string first.
- For example we can write the total number of words to our file as follows:
- >>> str(len(words))
- >>> output_file.write(str(len(words)) + "\n")
- >>> output_file.close()