Chapter 3 - Processing Raw Text

Jianzhang Zhang

Alibaba Business School
Hangzhou Normal University

April 7, 2022

Table of Contents

Table of Contents

Electronic Books

You can browse the catalog of 25,000 free online books at Project Gutenberg , and obtain a URL to an ASCII text file.

```
# 本章代码都先导入以下模块
import nltk, re, pprint
from nltk import word_tokenize

from urllib import request
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
respon = request.urlopen(url)
if respon.code == 200:
    raw = respon.read().decode('utf-8-sig')
else:
    print(" 未能成功获取网页内容，请检查网络")
```

Electronic Books (contd.)

Tokenization breaks up the string into **a list of words and punctuation**.

Create an **NLTK text** from this list and carry out all of the other linguistic processing we saw before.

```
tokens = word_tokenize(raw)
type(tokens) # list
len(tokens) # 257058
tokens[:10] # ['The','Project','Gutenberg','eBook','of',
↪  'Crime','and','Punishment',',','by']

text = nltk.Text(tokens)
text.collocations() # 寻找词语搭配
# Katerina Ivanovna; Pyotr Petrovitch; Pulcheria
↪  Alexandrovna; Avdotya Romanovna; Rodion Romanovitch;
```

Dealing with HTML

To access the HTML and get text out of HTML, we will use the Python library **urllib** (or **requests**) and **BeautifulSoup** seperately.

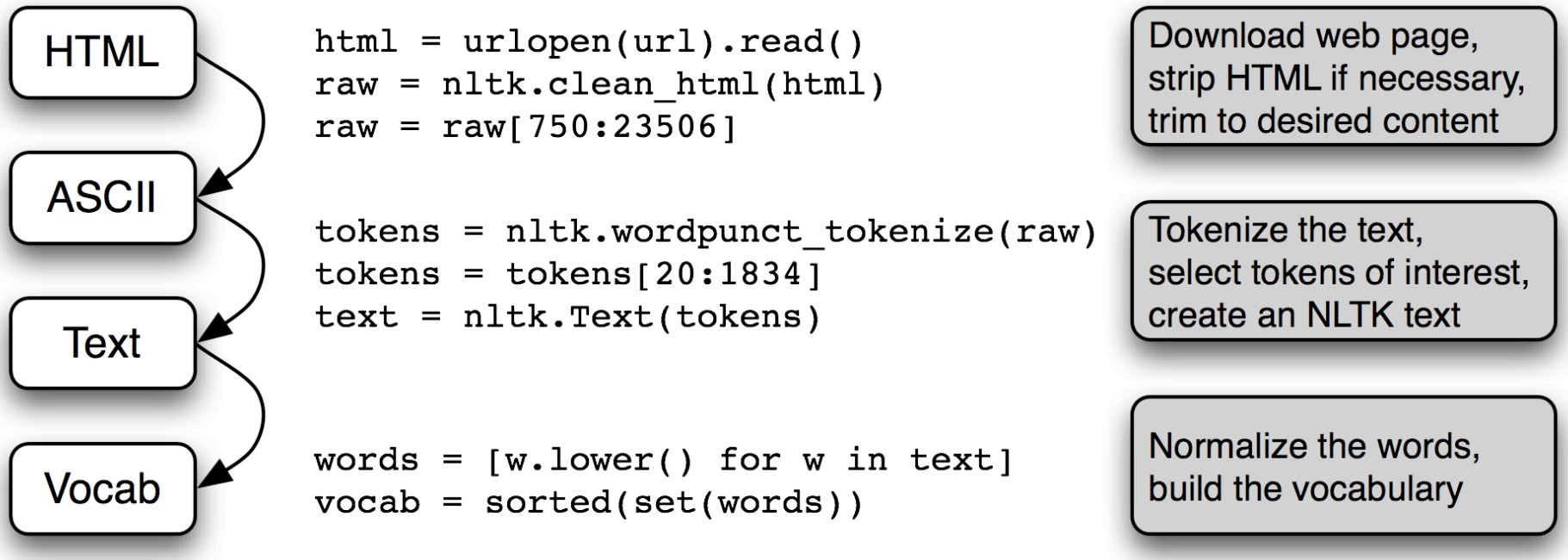
```
# China's leading liquor producer reports gains in first two
↪  months
# 中国领先的白酒生产商报告前两个月收益
# From China Daily
url = 'https://www.chinadaily.com.cn/a/202203/12/WS622c503fa310c
↪  dd39bc8c31b.html'
html = request.urlopen(url).read().decode('utf8')
print(html)
from bs4 import BeautifulSoup
raw = BeautifulSoup(html, 'html.parser').get_text().strip()
tokens = word_tokenize(raw)
print(tokens[:100])
```

For reading Local Files, please refer to chapter 9 of my programming basics course

Dealing with Other Inputs

- **Reading Local Files**
  Please refer to chapter 9 of my programming basics course.
- **Extracting Text from PDF, MSWord and other Binary Formats**
  Find excellent third-party packages for extracting text from PDF, MS Word, and other binary formats by yourself.
- **Capturing User Input**
  Use the input function.

The NLP Pipeline



We open a URL and read its HTML content, remove the markup and select a slice of characters; this is then tokenized and optionally converted into an nltk.Text object; we can also lowercase all the words and extract the vocabulary.

Table of Contents

The contents of a word, and of a file, are represented by programming languages as a fundamental data type known as a **string**. Please refer to chapter 3 of my programming basics course for the following topics:

- Basic Operations with Strings
- Printing Strings
- Accessing Individual Characters
- Accessing Substrings
- The Difference between Lists and Strings

Table of Contents

Table of Contents

Using Basic Meta-Characters

Many linguistic processing tasks involve pattern matching. Regular expressions give us a more powerful and flexible method for describing the character patterns we are interested in.

```
import re # 使用正则表达式需要导入re包

# 过滤掉专有名词
wordlist = [w for w in nltk.corpus.words.words('en') if
↪  w.islower()]
# 找出以ed为结尾的单词
[w for w in wordlist if re.search('ed$', w)]
# 找出满足如下条件的单词：长度为8，第3个字符为j，第6个字符为t
[w for w in wordlist if re.search('^..j..t..$', w)]
# 数一下email/e-mail在text中出现的次数
sum(1 for w in text if re.search('^e-?mail$', w))
```

Ranges and Closures

```
# 九宫格键盘上的数字按键序列4653对应的合法单词
[w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
# 匹配只包含字母a和h的单词
[w for w in chat_words if re.search('^[ha]+$', w)]
# 匹配只包含字母m、i、n、e的单词或空字符串
[w for w in chat_words if re.search('^m*i*n*e*$', w)]
```

Figure 1: T9: Text on 9 Keys

[]表示范围，+ 和 * 称为闭包。^[ghi]匹配单词的开头，后跟 g 或 h 或 i 其中之一；+表示匹配前一项的一个或多个实例，* 表示表示匹配前一项的 0 个或多个实例。

```
# find words like "black-and-white"
[w for w in wsj if re.search('^[a-z]{5,}-[a-z]{2,3}-[a-z]{,6}$',
↪  w)]
# find words ending with "ed" or "ing"
[w for w in wsj if re.search('(ed|ing)$', w)]
```

Operator	Behavior
.	Wildcard, matches any character
^abc	Matches some pattern *abc* at the start of a string
abc$	Matches some pattern *abc* at the end of a string
[abc]	Matches one of a set of characters
[A-Z0-9]	Matches one of a range of characters
ed|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item, e.g. a*, [a-z]* (also known as *Kleene Closure*)
+	One or more of previous item, e.g. a+, [a-z]+
?	Zero or one of the previous item (i.e. optional), e.g. a?, [a-z]?
{n}	Exactly *n* repeats where n is a non-negative integer
{n,}	At least *n* repeats
{,n}	No more than *n* repeats
{m,n}	At least *m* and no more than *n* repeats
a(b|c)+	Parentheses that indicate the scope of the operators

Figure 2: Basic Regular Expression Meta-Characters

Table of Contents

Extracting Word Pieces

```
# 在某些文本中查找两个或多个元音的所有序列，并确定它们的相对频率
wsj = sorted(set(nltk.corpus.treebank.words()))
fd = nltk.FreqDist(vs for word in wsj for vs in
↪  re.findall(r'[aeiou]{2,}', word))
fd.most_common(12)
```

For English words, it is still easy to read when word-internal vowels (元音) are left out.

```
# 匹配初始元音序列、最终元音序列和所有辅音，忽略其他
regexp = r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]'

def compress(word):
  pieces = re.findall(regexp, word)
  return ''.join(pieces)

compress('problem') # prblm
```

Finding Word Stems

For some language processing tasks we want to ignore word endings (e.g. ed, ing, es etc.), and just deal with word stems.

```
# 使用正则表达式找出单词的stem和后缀
def stem(word):
  regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
  stem, suffix = re.findall(regexp, word)[0]
  return stem

stem('processes') # process
stem('language') # language
stem('process') # proces
```

`*` 默认是贪婪模式，会匹配尽可能多的字符，`*?` 为表示非贪婪模式。

Searching Tokenized Text

```
from nltk.corpus import gutenberg, nps_chat

# 找出描述男士的形容词
moby = nltk.Text(gutenberg.words('melville-moby_dick.txt'))
# monied; nervous; dangerous; white
moby.findall(r"<a> (<.*>) <man>")

from nltk.corpus import brown

# X and other Ys, Y是x的上位词
hobbies_learned = nltk.Text(brown.words(categories=['hobbies',
↪  'learned']))
# speed and other activities
hobbies_learned.findall(r"<\w*> <and> <other> <\w*s>")
```

The above usage of `findall` method is unique to `nltk.Text`.

Table of Contents

- Lowercase
- Stemming: strip off any affixes
- Lemmatization: return a known word in a dictionary

```
'PYTHON'.lower() # python

porter = nltk.PorterStemmer()
porter.stem('lying') # lie

wnl = nltk.WordNetLemmatizer()
wnl.lemmatize('listening') # listen
```

Table of Contents

```
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,'..."""

# split on whitespace
raw.split()
re.split(r'\s+', raw)

# 以非单词字符分割输入文本
re.split(r'\W+', raw)

# 多次试错后可以得到更加完善的分词正则表达式
print(re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw))

# 调用NLTK的正则表达式分词器，并指定分词所需的正则表达式
nltk.regexp_tokenize(text, pattern = r'\w+|\$[\d\.]+|\S+')
```

Table of Contents

Tokenization is an instance of a more general problem of segmentation.

- Sentence Segmentation: 有些句点同时表示缩写 (如首字母缩略词) 和句子结束。
- Word Segmentation: 爱 / 国人，爱国 / 人。
- 处理口语语音时也会遇到类似分词的问题，即，听者必须将连续的语音流分割成单个单词以便理解 (类似婴幼儿学习语言的过程)。但是，人在说话时，每句话之间会有自然的停顿，听者便知道句子是如何分割的，因此，句子分割可以作为一段待分词文本的初始分割。

```
# Punkt sentence segmenter
text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = nltk.sent_tokenize(text)
pprint.pprint(sents[79:89])
```

Non-Deterministic Search Using Simulated Annealing

Suppose there are four sentences as follow:

a. *doyouseethekitty -> do you see the kitty*

b. *seethedoggy -> see the doggy*

c. *doyoulikethekitty -> do you like the kitty*

d. *likethedoggy -> like the doggy*

Firstly, we formalize the word segmentation problem to a search problem.

```
# 原始文本
text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
# 句子分割，1表示在该字符之后进行分割
seg1 = "000000000000001000000000100000000000000100000000000"
# 分词
seg2 = "010010010010000100100100001010010001001000010010010000"
```

Basic idea: we can define an objective function, a scoring function whose value we will try to optimize, based on the size of the lexicon (number of characters in the words plus an extra delimiter character to mark the end of each word) and the amount of information needed to reconstruct the source text from the lexicon.

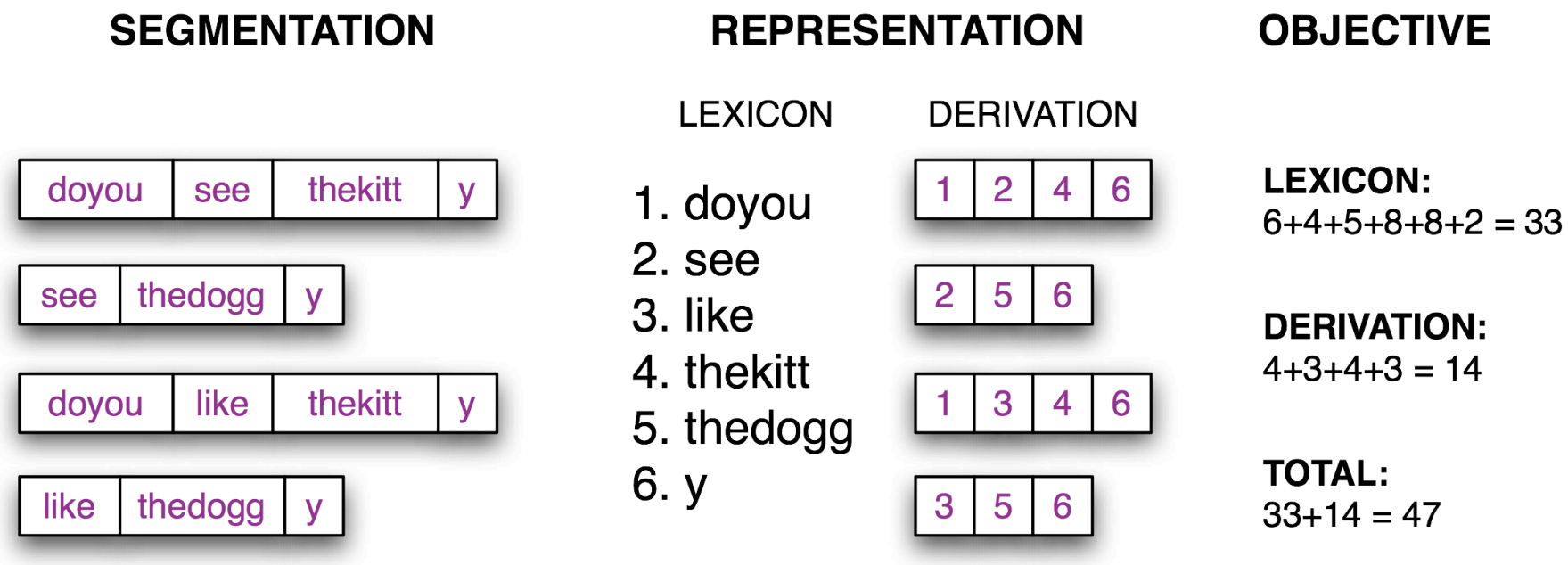


Figure 3: Calculation of Objective Function

Calculation of Objective Function:

1. Given a hypothetical segmentation of the source text (on the left); (给定一个假设分割)
2. Derive a lexicon and a derivation table (导出一个词典和导出表) that permit the source text to be reconstructed;
3. Total up the number of characters used by each lexical item (including a boundary marker) and the number of lexical items used by each derivation, to serve as a score of the quality of the segmentation;

   Smaller values of the score indicate a better segmentation.

Secondly, we define the objective function:

```
# 定义目标函数
def evaluate(text, segs):
  words = segment(text, segs)
  text_size = len(words)
  lexicon_size = sum(len(word) + 1 for word in set(words))
  return text_size + lexicon_size

text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
seg1 = "000000000000001000000000100000000000000100000000000"
seg2 = "010010010010000100100100001010010001001000010001001000"
seg3 = "000010010000001100100000011000100010000001100010000001"
segment(text, seg3)

evaluate(text, seg3) # 47
evaluate(text, seg2) # 48
evaluate(text, seg1) # 64
```

Finally, we search for the pattern of zeros and ones that minimizes this objective function:

```python
from random import randint

# 扰动一次0或1
def flip(segs, pos):
    return segs[:pos] + str(1-int(segs[pos])) + segs[pos+1:]

# 扰动n次0或1
def flip_n(segs, n):
    for i in range(n):
        segs = flip(segs, randint(0, len(segs)-1))
    return segs

def anneal(text, segs, iterations, cooling_rate):
    temperature = float(len(segs))
    print('Initial temperature: {}'.format(temperature))
    # 终止条件为温度不大于0.5
    while temperature > 0.5:
        print('Current Temperature: {}'.format(temperature))
        best_segs, best = segs, evaluate(text, segs)
        # 对每个temperature，迭代iterations次
        for i in range(iterations):
            # 根据温度确定每次迭代的扰动次数
            # 温度越低扰动次数越少
            guess = flip_n(segs, round(temperature))
            score = evaluate(text, guess)
            if score < best:
                best, best_segs = score, guess
        score, segs = best, best_segs
        # 降温（退火）
        temperature = temperature / cooling_rate
        print(evaluate(text, segs), segment(text, segs))
    print()
    return segs

text = "doyouseethekittyseethedoggydoyoulikethekittylikethedoggy"
seg1 = "000000000000001000000000010000000000000001000000000"
anneal(text, seg1, 5000, 1.2)
```

Table of Contents

Please refer to Chapter 3 and Chapter 9 of my programming basics course:

- join method;
- string formatting;
- print to file;
- write results to file.

THE END