Chapter 3

Karl Ledermüller & Norbert Walchhofer

Research Seminar - Statistical Natural Language Processing

5th November 2010

Inhalt

1. Accessing Text
   - Accessing Text from the Web and from Disk
   - tokenization
   - collocations
   - find text in document
   - read html text
   - concordances
   - Google Hits
   - Readers
2. string manipulation
3. Regular Expressions
4. Normalizing Text

Electronic Books - Gutenberg

Python

```
>>> from urllib import urlopen
>>> url = "http://www.gutenberg.org/files/2554/2554.txt"
>>> raw = urlopen(url).read()
>>> type(raw)
<type 'str'>
>>> len(raw)
1176831
>>> raw[:75]
'The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky\r\n'
```

R

```
> DosSource <- URISource("http://www.gutenberg.org/files/2554/2554.txt")
> Dostoevsky <- Corpus(DosSource)
> Dostoevsky[[1]][1]
[1] "The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky"
```

differences: In Phython the book seems to be saved as a string, in R/tm within a corpus object - no automatic handling of meta information found for Project Gutenberg in R (would be interesting)

tokenization

Python

```
>>> tokens = nltk.word_tokenize(raw)
>>> type(tokens)
<type 'list'>
>>> len(tokens)
255809
>>> tokens[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of', 'Crime', 'and', 'Punishment', ',', 'by']
```

R

```
> tf <- termFreq(Dostoevsky[[1]])
> head(tf)
      --"that          --and           --he          --she --translator's     --zossimov
            1              2              1              1              1              1
```

differences: In Phython the tokenization works on a string object. Tokenization rules can be modified (p.111). R seems to use white space tokenization. It seems to be possible to change tokenization rules.

collocations

Within the area of corpus linguistics, collocation is defined as a sequence of words or terms which co-occur more often than would be expected by chance. (en.wikipedia.org/wiki/Collocation)

Python

```
>>> text = nltk.Text(tokens)
>>> type(text)
<type 'nltk.text.Text'>
>>> text.collocations()
Katerina Ivanovna; Pulcheria Alexandrovna; Avdotya Romanovna; Pyotr
Petrovitch; Project Gutenberg; Marfa Petrovna; Rodion Romanovitch;
Sofya Semyonovna; Nikodim Fomitch; did not; Hay Market; Andrey
Semyonovitch; old woman; Literary Archive; Dmitri Prokofitch; great
deal; United States; Praskovya Pavlovna; Porfiry Petrovitch; ear rings
```

R Seems to be missing?

find text

Python

```
>>> raw.find("PART I")
5303
>>> raw.rfind("End of Project Gutenberg's Crime")
1157681
```

R

```
> grep("PART I", as.character(Dostoevsky[[1]]))
[1]   132  3529  7979 11407
> Dostoevsky[[1]][132]
[1] "PART I"
> Dostoevsky[[1]][3529]
[1] "PART II"
> Dostoevsky[[1]][7979]
[1] "PART III"
> Dostoevsky[[1]][11407]
[1] "PART IV"
```

grep gives the line info

html text

Python

```
>>> url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = urlopen(url).read()
>>> html[:60]
'<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN'
```

R

```
> art <- URISource("http://news.bbc.co.uk/2/hi/health/2284783.stm")
> article <- Corpus(art)
> article[[1]][1]
[1] "<!doctype html public \"-//W3C//DTD HTML 4.0 Transitional//EN\" \"http://www.w3.org/TR/REC-html40/loose.dt
> article[[1]][10]
[1] "<meta name=\"Headline\" content=\"Blondes 'to die out in 200 years'\">"
```

grep gives the line info

concordances

A concordance is an index of all main words in a book along with their immediate contexts. (wordnetweb.princeton.edu/perl/webwn)

Python

```
>>> tokens = tokens[96:399]
>>> text = nltk.Text(tokens)
>>> text.concordance('gene')
 they say too few people now carry the gene for blondes to last beyond the next tw
t blonde hair is caused by a recessive gene . In order for a child to have blonde
to have blonde hair , it must have the gene on both sides of the family in the gra
there is a disadvantage of having that gene or by chance . They don ' t disappear
ondes would disappear is if having the gene was a disadvantage and I do not think
```

R

```
> center <- grep("gene", as.character(article[[1]]))
> concordances <- sapply (center, function(x) substr (article[[1]][x],
(gregexpr("gene",article[[1]][x])[[1]][1])-40,(gregexpr("gene",article[[1]][x])[[1]][1])+40 ))
> concordances
[1] "t they say too few people now carry the gene for blondes to last beyond the next "
[2] "at blonde hair is caused by a recessive gene. "
[3] "d to have blonde hair, it must have the gene on both sides of the family in the g"
[4] " there is a disadvantage of having that gene or by chance. They don't disappear,\""
[5] "londes would disappear is if having the gene was a disadvantage and I do not thin"
```

Search Engine Results - Google Hits

Python A project counts hits of collocations (two dimensions)

R The Rgoogle Search of Mazanec does exactly the same
http://www.wu.ac.at/itf/downloads/software/gsearch

Load RSS Feeds - or local data

Python The library http://feedparse.org/ loads RSS Feeds The library pypdf and pywin32 access .pdf and .doc files

R

```
> getReaders()
 [1] "readDOC" "readGmane" "readPDF" "readReut21578XML"
 [5] "readReut21578XMLasPlain" "readPlain" "readRCV1" "readRCV1asPlain"
 [9] "readTabular" "readXML"
>
```

Inhalt

1 Accessing Text

2 string manipulation
- methods
- Encoding
- Pattern Matching

3 Regular Expressions

4 Normalizing Text

5 Segmentation & Formatting

6 Conclusions

string manipulation

Python Phyton works on strings. String operations can be found on page 91f.

```
s.find(t)  index of first instance of string t inside s (-1 if not found)
s.rfind(t)  index of last instance of string t inside s (-1 if not found)
s.index(t)  like s.find(t) except it raises ValueError if not found
s.rindex(t)  like s.rfind(t) except it raises ValueError if not found
s.join(text)  combine the words of the text into a string using s as the glue
s.split(t)  split s into a list wherever a t is found (whitespace by default)
s.splitlines()  split s into a list of strings, one per line
s.lower()  a lowercased version of the string s
s.upper()  an uppercased version of the string s
s.title()  a titlecased version of the string s
s.strip()  a copy of s without leading or trailing whitespace
s.replace(t, u)  replace instances of t with u inside s
```

R In R there exist methods of string manipulation paste(), substr(), nchar(), strsplit(), grep(), regexpr(), sub(), gsub() the tm_map() method does pre-processing within corpus objects and allows to include functions which use the methods stated above.

```
> getTransformations()
[1] "as.PlainTextDocument" "removeNumbers" "removePunctuation" "removeWords" "stemDocument"
[6] "stripWhitespace"

corp <- tm_map(corp, removeNumbers)
corp <- tm_map(corp, function(x) gsub("expressionA","expressionB",x))
```

string manipulation - examples

```
\textbf{Python}
>>> grail = 'Holy Grail'
>>> print monty + grail
Monty PythonHoly Grail
>>> print monty, grail
Monty Python Holy Grail
>>> print monty, "and the", grail
Monty Python and the Holy Grail
```

R

```
> monty <- "Monty Phyton"
> grail <- "Holy Grail"
> print(monty)
[1] "Monty Phyton"
> print(grail)
[1] "Holy Grail"
> print(c(monty, grail))
[1] "Monty Phyton" "Holy Grail"
> substring(monty,1,1)
[1] "M"
> substring(monty,7,15)
[1] "Phyton"
> strsplit("Phyton",monty)
[[1]]
[1] "Phyton"
```

Unicode - Encoding

Python In Phyton, they also work on decoded text, and encode or decode it with different encodings.

R Also R delivers decoding and encoding possibilities

```
> x <- "fa\xE7ile"
> Encoding(x)
[1] "latin1"
> Encoding(x) <- "latin1"
> x
[1] "façile"
> xx <- iconv(x, "latin1", "UTF-8")
> Encoding(c(x, xx))
[1] "latin1" "UTF-8"
> c(x, xx)
[1] "façile" "façile"
```

Meta Characters

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]*
+	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

Pattern Matching - mobile phones example

Regular expressions (and their metacharacters) in Python and R are quite similar, just the methods differ. Two explaining example are stated below.

Python

```
>>> [w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
['gold', 'golf', 'hold', 'hole']
```

R

```
> mobile <- c("gold", "golf", "hole", "hold", "anyword", "anotherword")
> regexpr("^[ghi][mno][jkl][def]$",mobile)
[1]  1  1  1  1 -1 -1
attr(,"match.length")
[1]  4  4  4  4 -1 -1
> mobile[regexpr("^[ghi][mno][jkl][def]$",mobile)!=-1]
[1] "gold" "golf" "hole" "hold"
> grep("^[ghi][mno][jkl][def]$",mobile)
[1] 1 2 3 4
> mobile[grep("^[ghi][mno][jkl][def]$",mobile)]
[1] "gold" "golf" "hole" "hold"
```

Pattern Matching - chat example

Python

```
>>> chat_words = sorted(set(w for w in nltk.corpus.nps_chat.words()))
>>> [w for w in chat_words if re.search('^m+i+n+e+$', w)]
['miiiiiiiiiiiinnnnnnnnnnneeeeeeeeee', 'miiiiiinnnnnnnnneeeeeeee', 'mine',
'mmmmmmmmiiiiiiiiinnnnnnnnneeeeeeee']
```

R

```
> mobile <- c("gold", "golf", "hole", "hold", "anyword", "anotherword")
> chatwords <- c("mine","mmmmiiiiinnnneeee","mmmmmmiiiiiiiiiiiiinnnnnnnnnnnnnneeeeeeeeeeeee",
"xmmiinnneee","yours","maxi")
> regexpr("^m+i+n+e+$",chatwords)
[1]  1  1  1 -1 -1 -1
attr(,"match.length")
[1]  4 17 50 -1 -1 -1
> grep("^m+i+n+e+$",chatwords)
[1] 1 2 3
> regexpr("m+i+n+e+$",chatwords)
[1]  1  1  1  2 -1 -1
attr(,"match.length")
[1]  4 17 50 10 -1 -1
> grep("m+i+n+e+$",chatwords)
[1] 1 2 3 4
> regexpr("m+i+n+e",chatwords)
[1]  1  1  1  2 -1 -1
attr(,"match.length")
[1]  4 14 37  8 -1 -1
> grep("m+i+n+e",chatwords)
[1] 1 2 3 4
```

Inhalt

1 Accessing Text

2 string manipulation

3 Regular Expressions
- Application of RegExps

4 Normalizing Text

5 Segmentation & Formatting

6 Conclusions

7 Project

Find Vowels

Python

```
>>> word = 'supercalifragilisticexpialidocious'
>>> re.findall(r'[aeiou]', word)
['u', 'e', 'a', 'i', 'a', 'i', 'i', 'i', 'e', 'i', 'a', 'i', 'o', 'i', 'o', 'u']
>>> len(re.findall(r'[aeiou]', word))
16
```

R

```
> word <- "supercalifragilisticexpialidocious"
> word
[1] "supercalifragilisticexpialidocious"
> strsplit(word,split="[aeiou]")
[[1]]
 [1] "s"  "p"  "rc" "l"  "fr" "g"  "l"  "st" "c"  "xp" ""   "l"  "d"  "c"  ""
[16] ""   "s"

> as.vector(strsplit(word, split="")[[1]])[as.vector(gregexpr("[aeiou]", word)[[1]])]
 [1] "u" "e" "a" "i" "a" "i" "i" "i" "e" "i" "a" "i" "o" "i" "o" "u"
>
> gregexpr("[aeiou]", word)
[[1]]
 [1]  2  4  7  9 12 14 16 19 21 24 25 27 29 31 32 33
attr(,"match.length")
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

> length(as.vector(gregexpr("[aeiou]", word)[[1]]))
[1] 16
```

Find Combinations of Vowels

Python

```
>>> 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.items()
[('io', 549), ('ea', 476), ('ie', 331), ('ou', 329), ('ai', 261), ('ia', 253),
('ee', 217), ('oo', 174), ('ua', 109), ('au', 106), ('ue', 105), ('ui', 95),
('ei', 86), ('oi', 65), ('oa', 59), ('eo', 39), ('iou', 27), ('eu', 18), ...]
```

R No direct functionality available...

```
> # frequences of two or more vowels
> vowelcomb2 <- apply(combn(c("a", "e", "i", "o", "u"), 2), 2, paste, collapse="")
> vowelcomb3 <- apply(combn(c("a", "e", "i", "o", "u"), 3), 2, paste, collapse="")
> rbind(lapply(sapply(c(vowelcomb2, vowelcomb3), grep,tokens), length))
     ae ai  ao au ei eo eu io  iu ou  aei aeo aeu aio aiu aou eio eiu eou iou
[1,] 11 293 3  86 90 40 14 532 12 598 0   0   0   0   0   0   0   0   15  85
```

Exclude Vowels

Python

```
>>> regexp = r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]'
>>> def compress(word):
...
pieces = re.findall(regexp, word)
...
return ''.join(pieces)
...
>>> english_udhr = nltk.corpus.udhr.words('English-Latin1')
>>> print nltk.tokenwrap(compress(w) for w in english_udhr[:75])
Unvrsl Dclrtn of Hmn Rghts Prmble Whrs rcgntn of the inhrnt dgnty and
of the eql and inlnble rghts of all mmbrs of the hmn fmly is the fndtn $
```

R

```
> text <- paste(Dostoevsky[[1]][start:(start+10)], collapse=" ")
> text
[1] "PART I    CHAPTER I  On an exceptionally hot evening early in July a young man
came out of the garret in which he lodged in S. Place and walked slowly, as though
in hesitation, towards K. bridge.  He had successfully avoided meeting his landlady
on the staircase. His"
> gsub("[aeiou]", "", text)
[1] "PART I    CHAPTER I  On n xcptnlly ht vnng rly n Jly  yng mn cm t f th grrt n
whch h ldgd n S. Plc nd wlkd slwly, s thgh n hsttn, twrds K. brdg.  H hd sccssflly
vdd mtng hs lndldy n th strcs. Hs"
```

Conditional Frequency Tables

Python

```
>>>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()
    a   e   i   o   u
k 418 148  94 420 173
p  83  31 105  34  51
r 187  63  84  89  79
s   0   0 100   2   1
t  47   8   0 148  37
v  93  27 105  48  49
```

R

```
> seta <- c("a", "e", "i", "o", "u")
> setb <- c("k", "p", "r", "s", "t", "v")
> settable <- expand.grid(seta, setb)    # create all possible combinations
> setcomb <- apply(settable, 1, paste, collapse="")
> cfreqs <- rbind(lapply(sapply(setcomb, grep,tokens), length))
> cfd <- matrix(as.integer(cfreqs), byrow=T, nrow=length(setb), dimnames=list(setb, seta))
> cfd
    a    e   i   o   u
k  87   24  39  77   2
p 173  174  82 138 101
r 716 1302 208 560 409
s 419 1143 607 182 405
t 825  316 540 197 223
v 108  165 222 215   6
```

Consonant-Vowel Index

Python

```
>>> cv_word_pairs = [(cv, w) for w in rotokas_words
...                          for cv in re.findall(r'[ptksvr][aeiou]', w)]
>>> cv_index = nltk.Index(cv_word_pairs)
>>> cv_index['su']
['kasuari']
>>> cv_index['po']
['kaapo', 'kaapopato', 'kaipori', 'kaiporipie', 'kaiporivira', 'kapo', 'kapoa',
'kapokao', 'kapokapo', 'kapokapo', 'kapokapoa', 'kapokapoa', 'kapokapora', ...]
```

R instead of a just-in-time grep we can build a list of words for each cv-pair.

```
> cv_pairs <- sapply(setcomb, grep, tokens, value=T)
> cv_pairs$uv
[1] "heruvimov"    "heruvimovand" "louvain"      "manoeuvre"    "mauvais"
[6] "youve"                                                              $
```

Word Stems

Python

```
>>> re.findall(r'^(.*)(ing|ly|ed|ious|ies|ive|es|s|ment)$', 'processing')
[('process', 'ing')] $
```

R

```
> gsub("*(ing|ly|ed|ious|ies|ive|es|s|ment)$", "", "processing")
[1] "process"
> strsplit("processing", split="*(ing|ly|ed|ious|ies|ive|es|s|ment)$")
[[1]]
[1] "process"
```

Inhalt

1 Accessing Text

2 string manipulation

3 Regular Expressions

4 Normalizing Text
- Stemming
- Lemmatization
- Simple Tokenization

5 Segmentation & Formatting

6 Conclusions

Stemmers

Python Nltk uses Porter & Lancaster stemmers

```
>>> porter = nltk.PorterStemmer()
>>> lancaster = nltk.LancasterStemmer()
>>> [porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern',
'.', 'Suprem', 'execut', 'power', 'deriv', 'from', 'a', 'mandat', 'from',
'the', 'mass', ',', 'not', 'from', 'some', 'farcic', 'aquat', 'ceremoni', '.']
>>> [lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond', 'distribut',
'sword', 'is', 'no', 'bas', 'for', 'a', 'system', 'of', 'govern', '.', 'suprem',
'execut', 'pow', 'der', 'from', 'a', 'mand', 'from', 'the', 'mass', ',', 'not',
'from', 'som', 'farc', 'aqu', 'ceremony', '.']
```

R Porter Sterm applied by tm.

```
> sd <- stemDocument(Dostoevsky[[1]])
> head(sd)
[1] "The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevski"
[2] ""
[3] "This eBook is for the use of anyon anywher at no cost and with"
[4] "almost no restrict whatsoever.  You may copi it, give it away or"
[5] "re-us it under the term of the Project Gutenberg Licens includ"
[6] "with this eBook or onlin at www.gutenberg.org"
```

Lemmatization

- complex form of stemming
- deploys dictonary to transform inflicted words to lemma (lexicon headword)
- context awarness (concordances)
- higher computational effort

Examples

- women → woman
- better → good
- „set up a meeting“→ meeting (noun)
- „were meeting him“→ meet (verb)
- 1983.54 → 0.0 (mapping for decimals)
- WU → AAA (mapping for acronyms)

tm

Lemmatization not available, but Dictionaries can be build from term document matrices or character vectors: *Dictionary*()

Simple Approaches to Tokenization

Python

```
>>> re.split(r'[ \t\n]+', 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,'..."]
```

R use regexp set for space characters

```
> unlist(strsplit(Dostoevsky[[1]][(start+6):(start+8)], split="[[:space:]]", perl=T))
 [1] "On"            "an"            "exceptionally" "hot"
 [5] "evening"       "early"         "in"            "July"
 [9] "a"             "young"         "man"           "came"
[13] "out"           "of"            "the"           "garret"
[17] "in"            "which"         "he"            "lodged"
[21] "in"            "S."            "Place"         "and"
[25] "walked"        "slowly,"       "as"            "though"
[29] "in"            "hesitation,"   "towards"       "K."
```

RegExps for keeping punctuation etc...

```
\b Word boundary (zero width)
\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
```

Inhalt

1 Accessing Text

2 string manipulation

3 Regular Expressions

4 Normalizing Text

5 Segmentation & Formatting
- Sentence Segmentation
- Formatting: From Lists to Strings
- Writing & Wrapping

6 Conclusions

Sentence Segmentation

Python

```
>>> sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = sent_tokenizer.tokenize(text)
>>> pprint.pprint(sents[171:181])
['"Nonsense!',
'" said Gregory, who was very rational when anyone else\nattempted paradox.',
'"Why do all the clerks and navvies in the\nrailway trains look so sad and tired,...',
'I will\ntell you.',
'It is because they know that the train is going right.',
'It\nis because they know that whatever place they have taken a ticket\nfor that ...',
```

R

```
> raw <- gsub('\"', "", paste(Dostoevsky[[1]][(start+10):(start+66)], collapse=" "), perl=T)
> sents <- head(unlist(strsplit(raw, split="['.'?''!']( *)", perl=T)))
> head(sents)
[1] "He had successfully avoided meeting his landlady on the staircase"
[2] "His garret was under the roof of a high, five-storied house and was more like a cupboard than a room"
[3] "The landlady who provided him with garret, dinners, and attendance, lived on the floor below, and every ti
[4] "And each time he passed, the young man had a sick, frightened feeling, which made him scowl and feel asham
[5] "He was hopelessly in debt to his landlady, and was afraid of meeting her"
[6] "This was not because he was cowardly and abject, quite the contrary; but for some time past he had been in
```

Use Sentence Segmentation

Calculate measures like avg number of characters of words per sentence:

Python

```
>>> len(nltk.corpus.brown.words()) / len(nltk.corpus.brown.sents())
20.250994070456922
```

R

```
> mean(sapply(sents, nchar))    # count avg characters per sentence
[1] 118
> mean(sapply(sapply(sents, strsplit, split=" "), length)) # count avg words per sentence
[1] 21.16667
```

Formating Text

- Basic String Formating (in R: cat(), paste(), print())
- Use of conversion specifiers
- place holder for decimals, strings, etc..

Conversion Specifiers

```
>>> 'I want a %s right now' % 'coffee'
'I want a coffee right now'
>>> count, total = 3205, 9375
>>> "accuracy for %d words: %2.4f%%" % (total, 100 * count / total)
'accuracy for 9375 words: 34.1867%'
```

Writing Results & Text Wrapping

- Basic Writing funcionality (in R: write(), write.table(), print())
- Text Wrapping to print text nicely, by adding some information

Python

```
>>> from textwrap import fill
>>> format = '%s (%d),'
>>> pieces = [format % (word, len(word)) for word in saying]
>>> output = ' '.join(pieces)
>>> wrapped = fill(output)
>>> print wrapped
After (5), all (3), is (2), said (4), and (3), done (4), , (1), more
(4), is (2), said (4), than (4), done (4), . (1),
```

R

```
> paste(words[1:20], " (", nchar(words[1:20]), "), ", sep="", collapse=" ")
[1] "PART (4),  I (1),  CHAPTER (7),  I (1),  On (2),  an (2),  exceptionally (13),  hot (3)
,  evening (7),  early (5),  in (2),  July (4),  a (1),  young (5),  man (3),  came (4)
,  out (3),  of (2),  the (3),  garret (6), "
```

Inhalt

1 Accessing Text

2 string manipulation

3 Regular Expressions

4 Normalizing Text

5 Segmentation & Formatting

6 Conclusions

7 Project

Conclusions

Some taks are easy complete with R & tm, however, to account for special cases some effort is required. In most cases small generic functions would help: words(), sents(), context(), ...

Missing

- Complicated string handling (concordances)
- No index for word and sentences, or object types (required?)
- Tokenization
- Collocation
- Lecical stemming lists (like remove stopwords)

Inhalt

1 Accessing Text

2 string manipulation

3 Regular Expressions

4 Normalizing Text

5 Segmentation & Formatting

6 Conclusions

7 Project

Project Sketch

Reader for Web blogs

- Apply Google Blog API (if possible)
- Read blogs created by different CMS Systems (Word Press, Joomla, Drupal, etc...)
- Encapsulate meta-information
- Extract comments as well

Challenges:

- Find general HMTL parsing rules
- Identification of sub-comments
- Transform nested blog structure to classic tm object