Exercise Hearst Patterns

Natural Language Processing

Ulrich Heid, Christian Wartena, Johannes Schäfer

Summer Term 2022

1 Exercises

Hearst patterns can be used to find concept classes and instances of concept classes in (large) text corpora. The classic Hearst patterns are give in Table 1.

Table 1: Hearst Patterns

Name	Pattern	Example
Hearst 1	`CONCEPT such as (INSTANCE)+ ((and | or) INSTANCE)?`	Cities such as Barcelona or Madrid
Hearst 2	`CONCEPT (,?) especially (INSTANCE)+ ((and | or) INSTANCE)?`	Countries especially Spain and France
Hearst 3	`CONCEPT (,?) including (INSTANCE)+ ((and | or) INSTANCE)?`	Capitals including London and Paris
Hearst 4	`INSTANCE (,?)+ and other CONCEPT`	Eiffel Tower and other monuments
Hearst 5	`INSTANCE (,?)+ or other CONCEPT`	Colloseum or other historical places

1. Write a regular expression for the Hearst patterns displayed above and extract a list of words and their concept class from the infection Corpus from the Notebook. Count how many supporting examples for each case have been found.

2 Working with Regular Expressions in Python

In Python you can use the library `re` to work with regular expressions. Regular expressions are always written in PERL syntax, i.e. the most common syntax for regular expressions in scripting and programming languages. The library allows to use regular expression directly or to compile them first for faster execution.

2.1 Functions

Some useful functions provided are the following ones:

search First argument: a regular expression. Second argument a string. Returns a match object if the expression is found and None otherwise. The match object corresponds to the first occurrence of the pattern in the string. Optionally start end end position of the region to be searched can be given as further arguments.

match First argument: a regular expression. Second argument a string. Returns a match object if the expression is found at the beginning of the string and None otherwise. Note that it is not required that the regular expression covers the whole string; it has only to match beginning of the string. Optionally start end end position of the region to be searched can be given as further arguments.

findall First argument: a regular expression. Second argument a string. Returns a list of matching strings if no groups are used; returns a list of list of strings if groups are used.

split First argument: a regular expression. Second argument a string. Returns a list of strings that correspond to all parts of the input string that do not match the pattern.

2.2 Groups and Match Objects

The search function returns a match object. A match object has the following functions:

group has an integer as argument. If the argument is 0 the function returns the entire matching string. Otherwise, if the argument is n, the string corresponding to the n^{th} group is returned.

start If the there is no argument or the argument is 0 the starting position of the match is returned. If the argument is n, the starting position of the substring matching the n^{th} group is returned.

end If the there is no argument or the argument is 0 the end position of the match is returned. If the argument is n, the end position of the substring matching the n^{th} group is returned.

Furthermore, you can use `match[n]` as shorthand for `match.group(n)` if `match` is a match object. For more details we refer to `https://docs.python.org/3/library/re.html`

Groups are parts of the regular expression within brackets. The groups are numbered starting with 1 (since 0 is reserved for the entire match). Groups also can be nested and in that case simply are numbered according to the position of their left brackets.

2.3 Examples

```
import re

zeile = ' Prof. Günther zeigt, wie mit tödlichen Viren geforscht wird'
fund = re.search('Vir(us|en)',zeile)
if fund:
    print(fund.group(0),fund.start(),'-',fund.end())
```

The program now prints *Viren 39 - 43*.

```
import re

zeile = ' und verschiedenen Arboviren.'
fund = re.search('(\w+)-?[Vv]ir(us|en)',zeile)
if fund:
     print(fund.group(0),fund.group(1),fund.group(2), sep = '|')
```

The program now prints *Arboviren|Arbo|en*.

Note that we need an additional pair of brackets if we want to output the entire string using the function findall:

```
import re

zeile = ' und  verschiedenen Arboviren.'
fundliste = re.findall('((\w+)-?[Vv]ir(us|en))',zeile)
if len(fundliste) > 0:
     for fund in fundliste:
          print(fund)
```

2.4 Compiling Regular Expressions

If execution time is important you can compile an expression. You now can use the same functions as before but now as methods of the regular expression object. The first argument is now implicit:

```
import re

pattern = re.compile('\w+')
string = "Isaac Newton, physicist"
position = 0
more = True

while more:
    m = pattern.search(string,position)
    if m:
         print(m.group(0))
         position = m.end(0)
    else:
         more = False
```

This program will output:

```
Isaac
Newton
physicist
```

2.5 Syntax Overview

Table 2: Alternatives and Repetition in Perl-Syntax for regular expressions

Symbol	Meaning
$\alpha \mid \beta$	α or β
$\alpha*$	α arbitrarily often (Kleene-closure)
$\alpha+$	α arbitrarily often, but at least once
$\alpha?$	α or nothing (α is optional)
$\alpha\{n\}$	α n-times repeated
$\alpha\{n,m\}$	α at least n times and at most m times
$\alpha\{n,\}$	α at least n times

Table 3: Special characters and character classes

Symbol	meaning
.	Any character except newline
\.	.
\t	Tab
\n	Line break (line feed)
\r	carriage return
\d	Digit
\D	Any character except a digit
\w	digit or letter ('word-character')
\W	Any character except a word character
\s	space or tab
\S	Any character except space and tab
$[\alpha]$	Any character from α, e.g. [aoueiAOUEI]
$[x-y]$	Any character from the range x to y
$[\hat{}\alpha]$	Any characte, except a character from α

Table 4: Symbols for boundaries in Perl-Syntax for regular expressions

Symbol	Meaning
^	Start of line
$	End of line
\b	Word boundary (start or end of a word)