READERS

Once we have our datasets, we have to be able to do something with them. For many tasks, we want to extract the words or the words with their tags. For this, we use the notion of readers from Chapter 4. We specify where we want to read from, what we want to use when we are reading a leaf file, and optionally a pattern that leaf files should match (we may need to do this if there are files other than actual data files stored in the directory where we are looking).

We start by thinking about the BNC. Leaf files in the BNC contain tagged sentences like the following:

<s n="1"><w c5="NN1" hw="factsheet" pos="SUBST">FACTSHEET </w><w c5="DTQ" hw="what" pos="PRON">WHAT </w><w c5="VBZ" hw="be" pos="VERB">IS </w><w c5="NN1" hw="aids" pos="SUBST">AIDS</w><c c5="PUN">?</c></s></head><p>

In order to extract words, with or without tags, from this we use the following regular expression:

<(?P<tagtype>w|c) .\*?c5="(?P<c5>[A-Z0-9]\*?)(-[A-Z0-9]\*)?".\*?>(?P<form>\S\*?)\s\*</(?P=tagtype)>

This will match an opening angle bracket followed by either w (for words) or c (for punctuation marks) followed by some stuff and then an optional tag (c5) and a closing bracket, then some non-whitespace, which is the word itself, followed by a closing </w> or </c> depending on whether the opening was w or c. There is a slight complication here because the BNC sometimes contains ambiguous tags, e.g. at one point the word care is given as <w c5="NN1-VVB" hw="care" pos="SUBST">care</w>, which says that it is not known whether this word is a noun or a verb. In such cases we assume that the first tag is correct.

We can look at all the occurrences of this pattern in the sentence above:

>>> s = """<s n="1"><w c5="NN1" hw="factsheet" pos="SUBST">FACTSHEET </w><w c5="DTQ" hw="what" pos="PRON">WHAT </w><w c5="VBZ" hw="be" pos="VERB">IS </w><w c5="NN1" hw="aids" pos="SUBST">AIDS</w><c c5="PUN">?</c></s></head><p>"""

>>> for i in readers.BNCWordPattern.finditer(s): print(i.groupdict())

{'tagtype': 'w', 'c5': 'NN1', 'form': 'FACTSHEET'}

{'tagtype': 'w', 'c5': 'DTQ', 'form': 'WHAT'}

{'tagtype': 'w', 'c5': 'VBZ', 'form': 'IS'}

{'tagtype': 'w', 'c5': 'NN1', 'form': 'AIDS'}

{'tagtype': 'c', 'c5': 'PUN', 'form': '?'}

We can clearly use this to find just the forms (by looking only for the group form) or by finding form: tag pairs (by looking for the values of form and c5).

The function reader will make a reader to scan a directory, looking for words or tagged words in leaf files whose name matches a given pattern (e.g. when scanning through the BNC we just want to look at files whose name ends with .xml, because there are other leaf files in this directory which we do not want to look at: note the pattern is just a standard regular expression, not something which will match Unix filename wildcards). Calling reader produces a generator object:

>>> bncwordreader = readers.reader(BNC.PATH, readers.BNCWordReader, pattern=".\*xml")

>>> bncwordreader

<generator object reader at 0x7fe2a63d4ac0>

We can turn this into a list, but given the size of the BNC this might not be the best thing to do, because loading the entire BNC will produce quite a long list and is likely to take a while:

>>> bncwords = list(bncwordreader)

>>> len(bncwords)

111651731

It may therefore make sense just read a smaller initial fragment: for sure, using 1M words for training some algorithm will give us a rough idea of whether it's any good, so we might as well do that first:

>>> bncwordreader = readers.reader(BNC.PATH, BNCWordReader, pattern=".\*xml")

>>> bncwords1000000 = [next(bncwordreader) for i in range(1000000)]

>>> len(bncwords1000000)

1000000

We can also specify that we only want words from one (or more) of the subdirectories:

This will just look at leaf files for which the path contains /B or /C, i.e. files from the B and C subsections of the BNC. This can be useful when splitting the corpus into training and testing portions – you might train an algorithm on everything except these two sections and then test it on these two.

>>> bncwordsBC = list(bncwordreaderBC)

>>> len(bncwordsBC)

27046896

We can also get a reader that will produce tagged items (just get the first 1M this time to save time!):

>>> bnctaggedwordreader = readers.reader(readers.BNC.PATH, readers.BNCTaggedWordReader, pattern=".\*xml")

>>> tagged = [next(bnctaggedwordreader) for i in range(1000000)]

>>> readers.printall(tagged[:20])

('FACTSHEET', 'NN')

('WHAT', 'DT')

('IS', 'VB')

('AIDS', 'NN')

('?', 'PU')

('AIDS', 'NN')

('(', 'PU')

('Acquired', 'VV')

('Immune', 'AJ')

('Deficiency', 'NN')

('Syndrome', 'NN')

(')', 'PU')

('is', 'VB')

('a', 'AT')

('condition', 'NN')

('caused', 'VV')

('by', 'PR')

('a', 'AT')

('virus', 'NN')

('called', 'VV')

There are a number of further parameters that can be tweaked: we will look at those as we need them later on.

UDT Readers

We can use the same machinery for making readers for the UDT. This time a typical sentence looks like

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | what | what | PRON | \_ | PronType=Int,Rel | 0 | root | \_ | \_ |
| 2 | is | be | AUX | \_ | Mood=Ind | 1 | cop | \_ | \_ |
| 3 | the | the | DET | \_ | PronType=Art | 4 | det | \_ | \_ |
| 4 | cost | cost | NOUN | \_ | Number=Sing | 1 | nsubj | \_ | \_ |
| 5 | of | of | ADP | \_ | \_ | 7 | case | \_ | \_ |

i.e. each line describes a word, with the second element on a line giving the surface form, the third giving the root form (so the second line above has is as the surface form and be as the root), and the third giving the major tag. We can therefore use the following regeular expression to read lines that contain words and dig out the form and tag:

UDTPattern = re.compile("(?P<COUNTER>\d\*)\t(?P<form>\S\*)\t(?P<FORM2>\S\*)\t(?P<TAG1>\S\*)\t(?P<TAG2>\S\*)\t(?P<TAG3>\S\*)\t(?P<hd>\d\*)\t(?P<ROLE>\S\*)")

This pattern will match lines that contain ten tab-separated items, where the first and seventh are numbers. The only line that match this in the UDT are indeed descriptions of words, so extracting the values for the group form will get us the words in the chosen files, and extracting form and TAG1 will get us the tagged versions.

We can use the pattern to specify for instance that we want the words in all the English datasets:

>>> udtwordreader = readers.reader(readers.UDT.PATH, readers.UDTWordReader, pattern=".\*En.\*.wholething")

>>> udtwords = list(udtwordreader)

>>> len(udtwords)

426888

>>> readers.printall(udtwords[:10])

what

is

the

cost

of

a

round

trip

flight

from

We see that there are 426888 words in all the English datasets in the UDT, and we can see the first ten of them (or whatever we want to see).

We can specify a particular dataset, e.g. the English ATIS dataset (this time we will get the tagged versions):

>>> udttaggedwordreader = readers.reader(readers.UDT.PATH, readers.UDTTaggedWordReader, pattern=".\*UD\_English-Atis.\*.wholething")

>>> udttagged = list(udttaggedwordreader)

>>> len(udttagged)

61879

>>> readers.printall(udttagged[:10])

('what', 'PRON')

('is', 'AUX')

('the', 'DET')

('cost', 'NOUN')

('of', 'ADP')

('a', 'DET')

('round', 'NOUN')

('trip', 'NOUN')

('flight', 'NOUN')

('from', 'ADP')

Other corpora

The same machinery will work for other corpora. You have to write a regular expression that picks out the components that you want, but after that you just use that to make a datafile reader and embed that in a general purpose reader. The PATB (Penn Arabic Treebank), for instance, provides a tagged Arabic corpus. Individual words look like

INPUT STRING: تعزيز

LOOK-UP WORD: tEzyz

Comment:

INDEX: P1W1

SOLUTION 1: (taEoziyz) [taEoziyz\_1] taEoziyz/NOUN

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride

\* SOLUTION 2: (taEoziyzu) [taEoziyz\_1] taEoziyz/NOUN+u/CASE\_DEF\_NOM

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride + [def.nom.]

SOLUTION 3: (taEoziyza) [taEoziyz\_1] taEoziyz/NOUN+a/CASE\_DEF\_ACC

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride + [def.acc.]

where a number of interpretations are given, with the preferred one in the current case marked with a \*.

Given that words in the PATB look like this, the following pattern will find a word and its tag:

>>> print(readers.PATBWordPattern.pattern)

INPUT STRING: (?P<form>\S\*).\*?\\* .\*?/(?P<tag>[^/]\*)

This says that if you look for the words INPUT STRING, then the non-white space that immediately follows is the form. Then look for a \* (written as \\* in the pattern, since \* is a significant term in regexes), which will indicate which is the preferred interpretation, and then look for something appearing between two slashes. This pattern will find us words and their tags in the PATB.

>>> s = """INPUT STRING: تعزيز

LOOK-UP WORD: tEzyz

Comment:

INDEX: P1W1

SOLUTION 1: (taEoziyz) [taEoziyz\_1] taEoziyz/NOUN

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride

\* SOLUTION 2: (taEoziyzu) [taEoziyz\_1] taEoziyz/NOUN+u/CASE\_DEF\_NOM

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride + [def.nom.]

SOLUTION 3: (taEoziyza) [taEoziyz\_1] taEoziyz/NOUN+a/CASE\_DEF\_ACC

(GLOSS): support/backing:strengthening/reinforcing:praise/encouragement/pride + [def.acc.]"""

>>> p = readers.PATBWordPattern.search(s)

('تعزيز', 'NOUN+u')

Unfortunately the PATB is not freely available, so we cannot redistribute it here. If you do happen to have it, then you can use our PATBWordReader and PATBTaggedWordReader functions to extract data from it, but the point here is more to note that it is fairly straightforward to write readers for arbitrary datasets so long as you can write a regular expression to match words in the format used by the dataset. Once you have done that you can follow the pattern in readers.py for defining a reader: define the regex, embed it in a function for extracting items from leaf files, and use that to make a reader:

BNCWordPattern = re.compile("""<(?P<tagtype>w|c) .\*?c5="(?P<c5>[A-Z0-9]\*?)(-[A-Z0-9]\*)?".\*?>(?P<form>\S\*?)\s\*</(?P=tagtype)>""")

def BNCWordReader(data):

for i in BNCWordPattern.finditer(open(data).read()):

form = i.group("form")

yield form

def BNCTaggedWordReader(data, N=2):

for i in BNCWordPattern.finditer(open(data).read()):

form = i.group("form")

yield (form, i.group("c5")[:N])

PATBWordPattern = re.compile("INPUT STRING: (?P<form>\S\*).\*?\\* .\*?/(?P<tag>[^/\n]\*)", re.DOTALL)

def PATBWordReader(path):

for i in PATBWordPattern.finditer(open(path).read()):

yield i.group("form")

def PATBTaggedWordReader(path):

for i in PATBWordPattern.finditer(open(path).read()):

yield i.group("form"), i.group("tag")

UDTPattern = re.compile("(?P<COUNTER>\d\*)\t(?P<form>\S\*)\t(?P<FORM2>\S\*)\t(?P<TAG1>\S\*)\t(?P<TAG2>\S\*)\t(?P<TAG3>\S\*)\t(?P<hd>\d\*)\t(?P<ROLE>\S\*)")

def UTDWordReader(data):

for i in UDTPattern.finditer(open(data).read()):

yield i.group("form")

def UDTTaggedWordReader(data):

for i in UDTPattern.finditer(open(data).read()):

yield (i.group("form"), i.group("TAG1"))

**YourWordPattern = re.compile("...(?P<form>...)...(?P<tag>...)")**

**def YourWordReader(data):**

**for i in UDTPattern.finditer(open(data).read()):**

**yield i.group("form")**

**def YourTaggedWordReader(data):**

**for i in UDTPattern.finditer(open(data).read()):**

**yield (i.group("form"), i.group("tag"))**