Cheatography

Natural Language Processing with Python & nltk Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15485/

Handling Text

`text='Some words'`	assign string
`list(text)`	Split text into character tokens
`set(text)`	Unique tokens
`len(text)`	Number of characters

Accessing corpora and lexical resources

`from nltk.c orpus import brow n`	import CorpusReader object
`brown.w or ds( tex t_id)`	Returns pretokenised document as list of words
`brown.f il eids()`	Lists docs in Brown corpus
`brown.c at ego ries()`	Lists categories in Brown corpus

Tokenization

`text.s pli t(" ")`	Split by space
`nltk.w ord _to ken ize r( text)`	nltk in-built word tokenizer
`nltk.s ent _to ken ize (d oc)`	nltk in-built sentence tokenizer

Lemmatization & Stemming

`input= "List listed lists listing listin g s"`	Different suffixes
`words= inp ut.l ow er( ).s plit(' ')`	Normalize (lower-case) words
`porter =nl tk.P or ter Stemmer`	Initialise Stemmer
`[porte r.s tem(t) for t in words]`	Create list of stems
`WNL=nl tk.W or dNe tLe mma tizer()`	Initialise WordNet lemmatizer
`[WNL.l emm ati ze(t) for t in words]`	Use the lemmatizer

Part of Speech (POS) Tagging

`nltk.h elp.up enn _ta gse t( 'MD')`	Lookup definition for a POS tag
`nltk.p os_ tag (words)`	nltk in-built POS tagger
	<use an alternative tagger to illustrate ambiguity>

Sentence Parsing

`g=nltk.da ta.l oa d(' gra mma r.cfg')`	Load a grammar from a file
`g=nltk.CF G.f rom str ing ("""...""")`	Manually define grammar
`parser =nl tk.C ha rtP ars er(g)`	Create a parser out of the grammar
`trees= par ser.pa rse _al l(text)`	
`for tree in trees: ... print tree`	
`from nltk.c orpus import treebank`	
`treeba nk.p ar sed _se nts ('w sj_ 00 0 1.mrg')`	Treebank parsed sentences

Text Classification

`from sklear n.f eat ure _ex tra cti on.text import Co ect orizer`	
`vect=C oun tVe cto riz er( ).f it( X_t rain)`	Fit bag
`vect.g et_ fea tur e_n ames()`	Get fe
`vect.t ran sfo rm( X_t rain)`	Conve

Entity Recognition (Chunking/Chinking)

`g="NP: {<D T>? <JJ >*< NN> - }"`	Regex chunk grammar
`cp=nlt k.R ege xpP ars er(g )`	Parse grammar
`ch=cp.p ar se( pos _sent)`	Parse tagged sent. using grammar
`print(ch)`	Show chunks
`ch.draw()`	Show chunks in IOB tree
`cp.eva lua te( tes t_s ents )`	Evaluate against test doc
`sents= nlt k.c orp us.t re eba nk.t ag ged _se nts( )`	
`print( nlt k.n e_c hun k(s - ent))`	Print chunk tree

RegEx with Pandas & Named Groups

`df=pd.D at aFr ame (ti me_ sents, column s=[ 'te xt'])`
`df['te xt' ].s tr.s pl it( ).s tr.l en()`
`df['te xt' ].s tr.c on tai ns( 'word')`
`df['te xt' ].s tr.c ou nt( r'\d')`
`df['te xt' ].s tr.f in dal l(r '\d')`
`df['te xt' ].s tr.r ep lac e(r '\w +da y\b', '???')`
`df['te xt' ].s tr.r ep lac e(r '(\w)', lambda x: x.grou ps( - )[0 ][:3])`
`df['te xt' ].s tr.e xt rac t(r '( \d? \d): (\d \d)')`
`df['te xt' ].s tr.e xt rac tal l(r '(( \d? \d) :(\d\d) ?([ap ] m))')`
`df['te xt' ].s tr.e xt rac tal l(r '(? P<d igi ts> \d)')`