Features

June 14, 2023

1 Importing Packages

[1]:
```
import nltk
import numpy as np
text = nltk.corpus.treebank.tagged_sents()
tagset = set([k[1] for k in nltk.corpus.treebank.tagged_words()])
print(tagset)

class color:
   BLACK = '\033[1;90m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[1;92m'
   YELLOW = '\033[93m'
   RED = '\033[1;91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'
   BBCKGRND = '\033[0;100m'
   RBCKGRND = '\033[0;101m'

print(color.BLUE + 'Hello World !' + color.END )
```

```
{'LS', 'VBZ', 'VB', 'VBD', 'PDT', ',', '-RRB-', 'NNP', 'WP$', 'RBS', 'POS',
'VBN', 'SYM', 'DT', '.', 'CD', ':', 'JJR', 'WRB', 'PRP', 'NNS', 'WDT', 'VBP',
'$', '``', 'FW', 'RP', 'PRP$', '-NONE-', '-LRB-', 'UH', 'MD', 'EX', 'TO', 'RB',
'#', "''", 'JJS', 'NN', 'RBR', 'NNPS', 'JJ', 'CC', 'WP', 'VBG', 'IN'}
Hello World !
```

[4]:
```
len(tagset)
```

[4]: 46

[2]:
```
nltk.help.upenn_tagset()
```

```
$: dollar
    $ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
'': closing quotation mark
```

    ' ''
(: opening parenthesis
    ([{
): closing parenthesis
)] }
,: comma
    ,
--: dash
    --
.: sentence terminator
    . ! ?
:: colon or ellipsis
    : ; …
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
    seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025
    fifteen 271,124 dozen quintillion DM2,000 …
DT: determiner
    all an another any both del each either every half la many much nary
    neither no some such that the them these this those
EX: existential there
    there
FW: foreign word
    gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous
    lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte
    terram fiche oui corporis …
IN: preposition or conjunction, subordinating
    astride among uppon whether out inside pro despite on by throughout
    below within for towards near behind atop around if like until below
    next into if beside …
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary …
JJR: adjective, comparative
    bleaker braver breezier briefer brighter brisker broader bumper busier
    calmer cheaper choosier cleaner clearer closer colder commoner costlier
    cozier creamier crunchier cuter …
JJS: adjective, superlative
    calmest cheapest choicest classiest cleanest clearest closest commonest
    corniest costliest crassest creepiest crudest cutest darkest deadliest
    dearest deepest densest dinkiest …
LS: list item marker
    A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005
    SP-44007 Second Third Three Two * a b c d first five four one six three

two
MD: modal auxiliary
can cannot could couldn't dare may might must need ought shall should shouldn't will would
NN: noun, common, singular or mass
common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist …
NNP: noun, proper, singular
Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Meltex Liverpool …
NNPS: noun, proper, plural
Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques Apache Apaches Apocrypha …
NNS: noun, common, plural
undergraduates scotches bric-a-brac products bodyguards facets coasts divestitures storehouses designs clubs fragrances averages subjectivists apprehensions muses factory-jobs …
PDT: pre-determiner
all both half many quite such sure this
POS: genitive marker
' 's
PRP: pronoun, personal
hers herself him himself hisself it itself me myself one oneself ours ourselves ownself self she thee theirs them themselves they thou thy us
PRP$: pronoun, possessive
her his mine my our ours their thy your
RB: adverb
occasionally unabatingly maddeningly adventurously professedly stirringly prominently technologically magisterially predominately swiftly fiscally pitilessly …
RBR: adverb, comparative
further gloomier grander graver greater grimmer harder harsher healthier heavier higher however larger later leaner lengthier less-perfectly lesser lonelier longer louder lower more …
RBS: adverb, superlative
best biggest bluntest earliest farthest first furthest hardest heartiest highest largest least less most nearest second tightest worst
RP: particle
aboard about across along apart around aside at away back before behind by crop down ever fast for forth from go high i.e. in into just later low more off on open out over per pie raising start teeth that through under unto up up-pp upon whole with you
SYM: symbol
% & ' '' ''.)). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker

```
    to
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly
    man baby diddle hush sonuvabitch …
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build …
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated …
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing …
VBN: verb, past participle
    multihulled dilapidated aerosolized chaired languished panelized used
    experimented flourished imitated reunifed factored condensed sheared
    unsettled primed dubbed desired …
VBP: verb, present tense, not 3rd person singular
    predominate wrap resort sue twist spill cure lengthen brush terminate
    appear tend stray glisten obtain comprise detest tease attract
    emphasize mold postpone sever return wag …
VBZ: verb, present tense, 3rd person singular
    bases reconstructs marks mixes displeases seals carps weaves snatches
    slumps stretches authorizes smolders pictures emerges stockpiles
    seduces fizzes uses bolsters slaps speaks pleads …
WDT: WH-determiner
    that what whatever which whichever
WP: WH-pronoun
    that what whatever whatsoever which who whom whosoever
WP$: WH-pronoun, possessive
    whose
WRB: Wh-adverb
    how however whence whenever where whereby whereever wherein whereof why
``: opening quotation mark
    ` ``
```

2 Constructing Context Function

2.1 Constructiong consent function

Consent Function get one sentence as an input, and it will "contextualize" it by transforming each pair (of **word** and its **tag**) in that sentence to pairs of **contexts** and their correponding **tag**.

Context function Finally Context function will get a text of *2 sentences* or more, and do the

same thing as *Consent* function for each sentence.

[2]:
```
# """ Context """#####################################
text = nltk.corpus.treebank.tagged_sents()
def consent(s):
    """coordinates represents, in the same order:
    preceding word (by 2 positions),
    preceding word (by 1 position)
    central word
    1st following word
    2nd following word
    is it a number
    starts with capital
    is it a starting word
    is an ending word
    etc."""
    if len(s)>=5 :
        contxt = {}
        # First word :
        contxt[(s[0][0],s[0][1],1)] = (0, 0, s[0][0], s[1][0],s[2][0],s[0][0].
↪isdigit(), s[0][0][0].isupper(), 1, 0)
        # Second word :
        contxt[(s[1][0],s[1][1],2)] = (0, s[0][0], s[1][0], s[2][0], s[3][0],␣
↪s[1][0].isdigit(), s[1][0][0].isupper(), 0, 0)
        # For middle words
        for i, w in enumerate(s[2:-2]) :
            contxt[(w[0],w[1],i+3)] = (s[i][0], s[i+1][0], s[i+2][0],␣
↪s[i+3][0], s[i+4][0], w[0].isdigit(), w[0][0].isupper(), 0, 0)
        # The word before last - the penultimate one - the second word to last
        contxt[(s[-2][0],s[-2][1],len(s)-1)] = (s[-4][0], s[-3][0], s[-2][0],␣
↪s[-1][0], 0, s[-2][0].isdigit(), s[-2][0][0].isupper(),0, 0)
        # The last word
        contxt[(s[-1][0],s[-1][1],len(s))] = (s[-3][0], s[-2][0], s[-1][0], 0,␣
↪0, s[-1][0].isdigit(), s[-1][0][0].isupper(), 0, 1)

    # Bellow are just copies of the 11 rows above, for exceptions : small␣
↪sentences that have less than 5 words
    elif len(s)==4:
        contxt = {}
        # First word :
        contxt[(s[0][0],s[0][1],1)] = (0, 0, s[0][0], s[1][0],s[2][0],s[0][0].
↪isdigit(), s[0][0][0].isupper(), 1, 0)
        # Second word :
        contxt[(s[1][0],s[1][1],2)] = (0, s[0][0], s[1][0], s[2][0], s[3][0],␣
↪s[1][0].isdigit(), s[1][0][0].isupper(), 0, 0)
        # The word before last - the penultimate one - the second word to last
```

```
        contxt[(s[-2][0],s[-2][1],len(s)-1)] = (s[-4][0], s[-3][0], s[-2][0],␣
↪s[-1][0], 0, s[-2][0].isdigit(), s[-2][0][0].isupper(),0, 0)
        # The last word
        contxt[(s[-1][0],s[-1][1],len(s))] = (s[-3][0], s[-2][0], s[-1][0], 0,␣
↪0, s[-1][0].isdigit(), s[-1][0][0].isupper(), 0, 1)

    elif len(s)==3:
        contxt = {}
        # First word :
        contxt[(s[0][0],s[0][1],1)] = (0, 0, s[0][0], s[1][0],s[2][0],s[0][0].
↪isdigit(), s[0][0][0].isupper(), 1, 0)
        # Second word :
        contxt[(s[1][0],s[1][1],2)] = (0, s[0][0], s[1][0], s[2][0], 0, s[1][0].
↪isdigit(), s[1][0][0].isupper(), 0, 0)
        # The last word
        contxt[(s[-1][0],s[-1][1],len(s))] = (s[-3][0], s[-2][0], s[-1][0], 0,␣
↪0, s[-1][0].isdigit(), s[-1][0][0].isupper(), 0, 1)

    elif len(s)==2:
        contxt = {}
        # First word :
        contxt[(s[0][0],s[0][1],1)] = (0, 0, s[0][0], s[1][0],0,s[0][0].
↪isdigit(), s[0][0][0].isupper(), 1, 0)
        # The last word
        contxt[(s[-1][0],s[-1][1],len(s))] = (0, s[0][0], s[1][0], 0, 0,␣
↪s[-1][0].isdigit(), s[-1][0][0].isupper(), 0, 1)

    else: contxt={}; contxt[(s[0][0],s[0][1],1)] = (0, 0, s[0][0], 0,0,s[0][0].
↪isdigit(), s[0][0][0].isupper(), 1, 1)
    return ([(k[:-1],v) for k,v in contxt.items()])

def context(text):
    txt = []
    if len(text)==1 :
        txt = consent(text)
    elif len(text)>1:
        for s in text:
            txt.extend(consent(s))
    else:
        txt = "Error in text format. Check if it contains sentences by running␣
↪'print(text).' "
    return txt
```

2.2 How contexts looks like

2.2.1 For Consent function

[3]:
```
consent(text[1])[:10]
```

[3]:
```
[(('Mr.', 'NNP'), (0, 0, 'Mr.', 'Vinken', 'is', False, True, 1, 0)),
 (('Vinken', 'NNP'),
  (0, 'Mr.', 'Vinken', 'is', 'chairman', False, True, 0, 0)),
 (('is', 'VBZ'),
  ('Mr.', 'Vinken', 'is', 'chairman', 'of', False, False, 0, 0)),
 (('chairman', 'NN'),
  ('Vinken', 'is', 'chairman', 'of', 'Elsevier', False, False, 0, 0)),
 (('of', 'IN'),
  ('is', 'chairman', 'of', 'Elsevier', 'N.V.', False, False, 0, 0)),
 (('Elsevier', 'NNP'),
  ('chairman', 'of', 'Elsevier', 'N.V.', ',', False, True, 0, 0)),
 (('N.V.', 'NNP'), ('of', 'Elsevier', 'N.V.', ',', 'the', False, True, 0, 0)),
 ((',', ','), ('Elsevier', 'N.V.', ',', 'the', 'Dutch', False, False, 0, 0)),
 (('the', 'DT'),
  ('N.V.', ',', 'the', 'Dutch', 'publishing', False, False, 0, 0)),
 (('Dutch', 'NNP'),
  (',', 'the', 'Dutch', 'publishing', 'group', False, True, 0, 0))]
```

For Context Function

[4]:
```
context(text)[:10]
```

[4]:
```
[(('Pierre', 'NNP'), (0, 0, 'Pierre', 'Vinken', ',', False, True, 1, 0)),
 (('Vinken', 'NNP'), (0, 'Pierre', 'Vinken', ',', '61', False, True, 0, 0)),
 ((',', ','), ('Pierre', 'Vinken', ',', '61', 'years', False, False, 0, 0)),
 (('61', 'CD'), ('Vinken', ',', '61', 'years', 'old', True, False, 0, 0)),
 (('years', 'NNS'), (',', '61', 'years', 'old', ',', False, False, 0, 0)),
 (('old', 'JJ'), ('61', 'years', 'old', ',', 'will', False, False, 0, 0)),
 ((',', ','), ('years', 'old', ',', 'will', 'join', False, False, 0, 0)),
 (('will', 'MD'), ('old', ',', 'will', 'join', 'the', False, False, 0, 0)),
 (('join', 'VB'), (',', 'will', 'join', 'the', 'board', False, False, 0, 0)),
 (('the', 'DT'), ('will', 'join', 'the', 'board', 'as', False, False, 0, 0))]
```

3 Features Preselection

3.1 Lexical aspect (staff/ step)

We start by : 1. *preceding word* by two positions 2. *preceding word* by one position 3. *current word* 4. *following word* by one position 5. *following word* by two positions

[5]:
```
# Frequency threshold - For Features Pre-selction
freq = 10
```

```
# Preceding by two positions
def preced2(tagged_sents):
    prec2 = nltk.defaultdict(int)
    precedent2 = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            prec2[i[0]] += 1
        for k,v in prec2.items() :
            if v > freq : precedent2[t].append(k)
    return [(k,v) for k,v in precedent2.items()]

p2 = []
for i in [v for k,v in preced2([s for s in text if len(s)>1])]:
    p2.extend(i)
p2 = list(set(p2))

# Preceding by one position
def preced1(tagged_sents):
    prec1 = nltk.defaultdict(int)
    precedent1 = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            prec1[i[1]] += 1
        for k,v in prec1.items() :
            if v > freq : precedent1[t].append(k)
    return [(k,v) for k,v in precedent1.items()]

p1 = []
for i in [v for k,v in preced1([s for s in text if len(s)>1])]:
    p1.extend(i)
p1 = list(set(p1))

# Current Word
def current(tagged_sents):
    cur = nltk.defaultdict(int)
    curr = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            cur[i[2]] += 1
        for k,v in cur.items() :
            if v > freq : curr[t].append(k)
    return [(k,v) for k,v in curr.items()]

c = []
for i in [v for k,v in current([s for s in text if len(s)>1])]:
    c.extend(i)
```

```
c = list(set(c))

# Following Word (1st)
def follow1(tagged_sents):
    fol = nltk.defaultdict(int)
    foll = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            fol[i[3]] += 1
        for k,v in fol.items() :
            if v > freq : foll[t].append(k)
    return [(k,v) for k,v in foll.items()]

f1 = []
for i in [v for k,v in follow1([s for s in text if len(s)>1])]:
    f1.extend(i)
f1 = list(set(f1))

# Following Word (2nd)
def follow2(tagged_sents):
    fol = nltk.defaultdict(int)
    foll = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            fol[i[4]] += 1
        for k,v in fol.items() :
            if v > freq : foll[t].append(k)
    return [(k,v) for k,v in foll.items()]

f2 = []
for i in [v for k,v in follow2([s for s in text if len(s)>1])]:
    f2.extend(i)
f2 = list(set(f2))
```

3.2 Morphological Clues

3.2.1 Suffixes (of length 3, 2 and 1 letters)

[6]:
```
# Suffix : Last 3 letters
def suffix3(tagged_sents):
    suf = nltk.defaultdict(int)
    suff = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            suf[i[2][-3:]] += 1
        for k,v in suf.items() :
            if v > freq : suff[t].append(k)
```

```
    return [(k,v) for k,v in suff.items()]

s3 = []
for i in [v for k,v in suffix3([s for s in text if len(s)>1])]:
    s3.extend(i)
s3 = list(set(s3))

# Suffix : Last 2 letters
def suffix2(tagged_sents):
    suf = nltk.defaultdict(int)
    suff = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            suf[i[2][-2:]] += 1
        for k,v in suf.items() :
            if v > freq : suff[t].append(k)
    return [(k,v) for k,v in suff.items()]

s2 = []
for i in [v for k,v in suffix2([s for s in text if len(s)>1])]:
    s2.extend(i)
s2 = list(set(s2))

# Suffix : Last letter
def suffix1(tagged_sents):
    suf = nltk.defaultdict(int)
    suff = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            suf[i[2][-1]] += 1
        for k,v in suf.items() :
            if v > freq : suff[t].append(k)
    return [(k,v) for k,v in suff.items()]

s1 = []
for i in [v for k,v in suffix1([s for s in text if len(s)>1])]:
    s1.extend(i)
s1 = list(set(s1))

print(s1)
```

```
['w', 'l', 'n', '?', 'd', '9', 'C', 'i', 'I', 'A', '$', '`', 'h', 'g', ':', 'o',
'4', 'E', 'c', '3', '-', 'p', '.', 'z', '5', '1', 'u', 'V', '%', 'P', 'T', 'y',
'*', 'f', 'e', 'b', 'R', 'N', '6', '8', 'r', 'L', 'x', "'", 's', 'X', '&', 'D',
',', 'G', '0', '7', 'k', 'S', 'm', 't', 'M', '2', '#', 'a', ';']
```

3.2.2 Prefixes (of length 3 or 2 letters)

[7]:
```
# Prefix : First 3 letters
def prefix3(tagged_sents):
    pre = nltk.defaultdict(int)
    pref = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            pre[i[2][:3]] += 1
        for k,v in pre.items() :
            if v > freq : pref[t].append(k)
    return [(k,v) for k,v in pref.items()]

pr3 = []
for i in [v for k,v in prefix3([s for s in text if len(s)>1])]:
    pr3.extend(i)
pr3 = list(set(pr3))

# Prefix : First 2 letters
def prefix2(tagged_sents):
    pre = nltk.defaultdict(int)
    pref = nltk.defaultdict(list)
    for t in tagset:
        for i in [k[1] for k in context(tagged_sents) if k[0][1]==t]:
            pre[i[2][:2]] += 1
        for k,v in pre.items() :
            if v > freq : pref[t].append(k)
    return [(k,v) for k,v in pref.items()]

pr2 = []
for i in [v for k,v in prefix2([s for s in text if len(s)>1])]:
    pr2.extend(i)
pr2 = list(set(pr2))

print(pr2[:15])
```

```
['In', 'Sh', 'At', 'Go', 'Ma', 'oi', '3', 'ob', 'fi', 'Po', 'sm', 'li', '18',
'id', "'v"]
```

[8]:
```
[k[1] for k in context(text)[:8]]
```

[8]:
```
[(0, 0, 'Pierre', 'Vinken', ',', False, True, 1, 0),
 (0, 'Pierre', 'Vinken', ',', '61', False, True, 0, 0),
 ('Pierre', 'Vinken', ',', '61', 'years', False, False, 0, 0),
 ('Vinken', ',', '61', 'years', 'old', True, False, 0, 0),
 (',', '61', 'years', 'old', ',', False, False, 0, 0),
 ('61', 'years', 'old', ',', 'will', False, False, 0, 0),
```

```
('years', 'old', ',', 'will', 'join', False, False, 0, 0),
('old', ',', 'will', 'join', 'the', False, False, 0, 0)]
```

Testing an example of selected features: “an” as a prefix of two letters

[9]:
```
prnt = [(k,v) for k,v in dict(nltk.FreqDist(nltk.Text([k[0][1] for k in␣
↪context(text) if k[1][2][:2]=="an"]))).items()]
```

[19]:
```
print(color.RED + color.BOLD + "  tag" + color.END, "\tfreq\n")
for k,v in prnt:
    print("  ", color.BOLD + color.RED + k + color.END, "\t", v)
```

```
 tag    freq

  CC     1505
  DT     461
  NN     74
  JJ     54
  VBD    18
  VB     5
  VBN    8
  VBZ    3
  RB     10
  NNS    46
  IN     1
  VBG    1
```

[22]:
```
for k,v in [(k,v) for s in text for (k,v) in s if k.startswith("an") and␣
↪v=="CC" ][:15]:
    print(k, color.RED + v + color.END)

print("\n")

for k,v in [(k,v) for s in text for (k,v) in s if k.startswith("an") and␣
↪v=="DT" ][:15]:
    print(k, color.BLUE + color.BOLD + v + color.END)
```

```
and CC
and CC
and CC
and CC
and CC
and CC
and CC
and CC
and CC
and CC
and CC
```

```
and CC
and CC
and CC
and CC


an DT
any DT
any DT
any DT
an DT
an DT
an DT
any DT
any DT
an DT
another DT
another DT
an DT
an DT
an DT
```

[26]:
```
General_Context = p2.copy()
General_Context.extend(p1)
General_Context.extend(c)
General_Context.extend(f1)
General_Context.extend(f2)
General_Context.extend(s1)
General_Context.extend(s2)
General_Context.extend(s3)
General_Context.extend(pr2)
General_Context.extend(pr3)

print(np.array(General_Context)[np.random.randint(0,␣
 ↪len(General_Context)-1,42)])
```

```
['fall' '500' 'Mid' 'jum' '2' 'declined' 'cut' 'change' 'manager' 'Li'
 'both' 'nc' 'alf' 'order' 'fel' 'G' 'Exchange' 'spokesman' 'billion' '%'
 'large' 'others' 'us' 'Tue' 'three' 'growth' 'an' 'pho' 'advertisers' '1'
 ',' 'efforts' 'Of' 'fs' 'Michael' 'put' 'a' 'pt' 'reached' 'dividends'
 'economy' 'close']
```

[ ]:
```

```

[27]:
```
def g(phrase):
    phrase = context(phrase)
    sent = nltk.defaultdict(list)
```

```
    for i, w in enumerate(phrase):
        x = []
        for w_2 in p2:
            x.append(int(w[1][0]==w_2))
        for w_1 in p1:
            x.append(int(w[1][1]==w_1))
        for w_0 in c:
            x.append(int(w[1][2]==w_0))
        for w1 in f1:
            x.append(int(w[1][3]==w1))
        for w2 in f2:
            x.append(int(w[1][4]==w2))
        for sf3 in s3:
            x.append(int(w[1][2][-3:]==sf3))
        for sf2 in s2:
            x.append(int(w[1][2][-2:]==sf2))
        for sf1 in s1:
            x.append(int(w[1][2][-1]==sf1))
        for pre3 in pr3:
            x.append(int(w[1][2][:3]==pre3))
        for pre2 in pr2:
            x.append(int(w[1][2][:2]==pre2))
        sent[(i,w[0])].extend(x)
        sent[(i,w[0])].extend([int(w[1][5]),int(w[1][6]),w[1][7],w[1][8]])
    return sent
```

[28]:
```
K = g(text[:1000])
```

[29]:
```
kk = dict(K)
```

[30]:
```
d = nltk.defaultdict(list)
for k in list(kk.keys()):
    d[tuple(kk[k])].append(k)
```

[32]:
```
teeeeest = tuple(K[list(K.keys())[1]])
print(teeeeest[:10])
```

```
(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

[33]:
```
def h(f):
    return d[f]

h(teeeeest)
```

[33]:
```
[(1, ('Vinken', 'NNP'))]
```

[34]:
```
import pickle
with open('d_hfunction','wb') as file:
    pickle.dump(d,file)
```

[35]:
```
def f(phrase):
    phrase = context(phrase)
    sent = nltk.defaultdict(list)
    for i, w in enumerate(phrase):
        x = []
        for w_2 in p2:
            x.append(int(w[1][0]==w_2))
        for w_1 in p1:
            x.append(int(w[1][1]==w_1))
        for w_0 in c:
            x.append(int(w[1][2]==w_0))
        for w1 in f1:
            x.append(int(w[1][3]==w1))
        for w2 in f2:
            x.append(int(w[1][4]==w2))
        for sf3 in s3:
            x.append(int(w[1][2][-3:]==sf3))
        for sf2 in s2:
            x.append(int(w[1][2][-2:]==sf2))
        for sf1 in s1:
            x.append(int(w[1][2][-1]==sf1))
        for pre3 in pr3:
            x.append(int(w[1][2][:3]==pre3))
        for pre2 in pr2:
            x.append(int(w[1][2][:2]==pre2))
        sent[(i,w[0])].extend(x)
        sent[(i,w[0])].extend([int(w[1][5]),int(w[1][6]),w[1][7],w[1][8]])
    return [(k[1],v) for k,v in sent.items()]
```

[36]:
```
context(text[:2])[:10]
```

[36]:
```
[(('Pierre', 'NNP'), (0, 0, 'Pierre', 'Vinken', ',', False, True, 1, 0)),
 (('Vinken', 'NNP'), (0, 'Pierre', 'Vinken', ',', '61', False, True, 0, 0)),
 ((',', ','), ('Pierre', 'Vinken', ',', '61', 'years', False, False, 0, 0)),
 (('61', 'CD'), ('Vinken', ',', '61', 'years', 'old', True, False, 0, 0)),
 (('years', 'NNS'), (',', '61', 'years', 'old', ',', False, False, 0, 0)),
 (('old', 'JJ'), ('61', 'years', 'old', ',', 'will', False, False, 0, 0)),
 ((',', ','), ('years', 'old', ',', 'will', 'join', False, False, 0, 0)),
 (('will', 'MD'), ('old', ',', 'will', 'join', 'the', False, False, 0, 0)),
 (('join', 'VB'), (',', 'will', 'join', 'the', 'board', False, False, 0, 0)),
 (('the', 'DT'), ('will', 'join', 'the', 'board', 'as', False, False, 0, 0))]
```

4 Extracting Features (That was previously selected)

[87]:
```
ff = f(text[:2])
print("-------------------------------------------------------------------------------------------
print(color.BOLD + "  Word,\t   Tag "+ color.END, color.BOLD + "\t  Features" +␣
 ↪color.END)
print("-------------------------------------------------------------------------------------------
for i in ff:
    k = i[1][:13] + i[1][-13:]
    if len(str(i[0])) < 15:
        print(color.BOLD + str(i[0]) + color.END, "\t\t", k)
    else:
        print(color.BOLD + str(i[0]) + color.END, "\t", k)
```

```
--------------------------------------------------------------------------------
-------------
  Word,         Tag             Features
--------------------------------------------------------------------------------
-------------
('Pierre', 'NNP')         [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]
('Vinken', 'NNP')         [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
(',', ',')                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('61', 'CD')              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
('years', 'NNS')          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('old', 'JJ')             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
(',', ',')                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('will', 'MD')            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('join', 'VB')            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('the', 'DT')             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('board', 'NN')           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('as', 'IN')              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('a', 'DT')               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('nonexecutive', 'JJ')    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
('director', 'NN')       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('Nov.', 'NNP')          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
('29', 'CD')             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
('.', '.')               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
('Mr.', 'NNP')           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
('Vinken', 'NNP')        [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
('is', 'VBZ')            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('chairman', 'NN')       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('of', 'IN')             [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('Elsevier', 'NNP')      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
('N.V.', 'NNP')          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
(',', ',')               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('the', 'DT')            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('Dutch', 'NNP')         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
('publishing', 'VBG')    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('group', 'NN')          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
('.', '.')               [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

[88]:
```
idtag = {}
k = 0
for j in tagset:
    idtag[j] = k
    k += 1

print(idtag)
```

```
{'EX': 0, '-RRB-': 1, 'CD': 2, 'WRB': 3, '-LRB-': 4, '``': 5, 'WP$': 6, 'PRP':
7, 'NNP': 8, '$': 9, 'WDT': 10, 'NNPS': 11, ':': 12, 'TO': 13, 'FW': 14, 'VBP':
15, 'VBN': 16, '.': 17, 'VBG': 18, 'VBZ': 19, 'RBS': 20, 'UH': 21, 'LS': 22,
'JJS': 23, 'DT': 24, 'IN': 25, 'RB': 26, '-NONE-': 27, 'WP': 28, "''": 29, 'RP':
30, 'VB': 31, 'NNS': 32, ',': 33, 'NN': 34, 'CC': 35, 'JJR': 36, 'JJ': 37,
```

```
'PDT': 38, 'RBR': 39, 'POS': 40, 'SYM': 41, '#': 42, 'PRP$': 43, 'VBD': 44,
'MD': 45}
```

[ ]:
```
X = np.array([i[1] for i in f(text[:3000])])
```

[ ]:
```
X.shape
```

[ ]:
```
y = np.array([idtag[i[0][1]] for i in f(text[:3000])])
y.shape
```

[ ]:
```
import pickle
with open('Data','wb') as file:
    pickle.dump((y,X),file)
```

[ ]:
```
import pickle
with open('idtag','wb') as file:
    pickle.dump(idtag,file)
```

[ ]:
```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0, solver='lbfgs',␣
 ↪multi_class='multinomial').fit(X, y)
clf.predict(X[:2, :])
```

[ ]:
```
clf.predict_proba(X[:2, :])
clf.score(X, y)
```

[ ]:
```
X[1]
```

[ ]:
```

```