Aquesta unitat disposa dels següents arxius:

- Aquests mateixos materials en PDF: 04-Analisi_textual-cat.pdf
- Els programes i arxius necessaris: programes4-cat.zip
- L'arxiu Notebook de Jupyter; cap4-cat.ipynb
- L'enllaç al notebook en Google Colaboratory: https://colab.research.google.com/github/aoliverg/python/blob/master/notebooks/cap4-cat.ipynb

4.1. Accés als corpus de l'NLTK

L'NLTK es distribueix amb una gran quantitat de corpus. Tot i que una gran part d'aquests corpus estan en anglès, n'hi ha també en altres llengües, com el català i el castellà. La llista completa i actualitzada dels corpus disponibles a l'NLTK es pot trobar en el següent enllaç: http://www.nltk.org/nltk_data/

Per exemple, NLTK inclou una selecció de textos del Projecte Gutenberg. Aquest projecte recopila llibres en format electrònic (llibres en domini públic, és a dir, sense drets d'autor i per tant legalment descarregables). Si volem carregar aquest corpus i observar els arxius que conté, podem fer en un intèrpret interactiu:

```
>>> from nltk.corpus import gutenberg
>>> llibres=gutenberg.fileids()
>>> print(llibres)
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

Podem accedir a les paraules d'un determinat llibre de la col·lecció fent el següent:

```
>>> import nltk
>>> mobi=nltk.corpus.gutenberg.words("melville-moby_dick.txt")
>>> print(mobi)
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', ...]
>>> print(len(mobi))
260819
```

Amb el mètode words hem pogut accedir a les paraules del llibre (que hem emmagatzemat a la variable mobi). A la darrera línia hem comptat les posicions de la llista mobi, és a dir, el nombre de paraules del llibre. En corpus de text com aquest podem accedir a:

- Typewriteral text en brut, amb el mètode raw
- els paràgrafs, amb el mètode paras
- a les oracions, amb el mètode sents
- a les paraules, amb el mètode words

Al programa següent (programa-4-1.py), podem observar el funcionament d'aquests mètodes (quan l'executis, per passar d'un nivell al següent, pitja la tecla Enter):

```
import nltk
print("TEXT EN BRUT")
a=input()
text_en_brut=nltk.corpus.gutenberg.raw("melville-moby_dick.txt")
```

```
print(text_en_brut[0:100000])
print("PARAGRAFS")
a=input()
paragrafs=nltk.corpus.gutenberg.paras("melville-moby_dick.txt")
for para in paragrafs:
    print(para)
print("ORACIONS")
a=input()
oracions=nltk.corpus.gutenberg.sents("melville-moby_dick.txt")
for oracio in oracions:
    print(oracio)
print("PARAULES")
a=input()
paraules=nltk.corpus.gutenberg.words("melville-moby_dick.txt")
for paraula in paraules:
    print(paraula)
```

Amb a=input() el sistema espera que l'usuari introdueixi algun valor i l'emmagatzema en a (que no fem servir per res). El que aconseguim és que el sistema s'esperi fins que l'usuari premi la tecla Enter.

Per ara hem estat treballant amb un corpus textual pla, sense cap tipus d'anàlisi ni anotació. Més endavant veurem que NLTK també proporciona corpus amb diversos nivells d'anàlisi.

4.2. Accés a corpus propis

NLTK proporciona una bona col·lecció de corpus, però amb tota seguretat necessitarem també treballar amb corpus propis. Per fer això podem fer servir les instruccions que ja sabem per obrir i llegir arxius de text (mireu l'apartat 2.3), o bé podrem fer servir els diferents lectors de corpus que proporciona NLTK.

El lector de corpus més elemental que proporciona l'NLTK és el **PlaintextCorpusReader**, que serveix, com indica el seu nom, per llegir corpus que estiguin en format de text pla (és a dir, només el text, sense cap tipus d'anotació). En el programa-4-2.py es pot observar el funcionament bàsic. En aquest cas també farem servir la novel·la Moby Dick, descarregada del Projecte Gutenberg i que podem trobar en els fitxers d'aquest capítol.

```
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
corpus = PlaintextCorpusReader(".", 'mobi_dick.txt')
for oracio in corpus.sents():
    print(oracio)
```

i a la sortida tindrem les oracions, que de fet, si us fixeu, són llistes de paraules:

```
...
['In', 'a', 'word', ',', 'the', 'whale', 'was', 'seized', 'and', 'sold', ',', 'and', 'his', 'Grace', 'the', 'Duke', 'of', 'Wellington', 'received', 'the', 'money', '.']
```

...

El **PlaintextCorpusReader** carrega el fitxer de text i du a terme un procés de segmentació en paràgrafs (a partir de salts de paràgrafs), en oracions (fent servir un segmentador determinat) i en paraules (o tokens, fent servir un tokenitzador determinat). Si no indiquem res en el moment de cridar al **PlaintextCorpusReader** fa servir:

- Tokenitzador: word_tokenizer=WordPunctTokenizer(),
- Segmentador: sent_tokenizer=nltk.data.LazyLoader('tokenizers/punkt/english.pickle') És a dir, assumeix per defecte que la llengua del corpus és l'anglès.

Com que el tokenitzador que fa servir és molt simple i es basa en separar el text en seqüències de caràcters alfabètics i no alfabètics fent servir l'expressió regular **w+|[^\w\s]+** es poden produir errors, com el següent:

["", 'Won', "", 't', 'the', 'Duke', 'be', 'content', 'with', 'a', 'quarter', 'or', 'a', 'half', '?""]

on won't s'ha separat com

won

'

t

Fixem-nos ara que si apliquem aquest mateix programa a un corpus en català la tokenització no serà totalment correcta. Amb aquest capítol es distribueix un fragment del Corpus del Diari Oficial de la Generalitat de Catalunya (corpus DOGC), concretament el corresponent a la versió catalana de l'any 2015. Si carreguem el fitxer DOGC-2015-cat.txt, en comptes de mobi_dick.txt obtenim un resultat com el següent (mostrem únicament un fragment):

. . .

['El', 'ple', 'de', 'l', "", 'Ajuntament', 'd', "", 'Òrrius', 'a', 'la', 'sessió', 'extraordinària', 'celebrada', 'el', 'dia', '20', 'de', 'gener', 'de', '2015', 'va', 'aprovar', 'inicialment', 'el', 'projecte', 'd', "", 'obra',

...

Fixem-nos que, per exemple, els apòstrofs (') queden com a tokens aïllats, separats de l'article o preposició corresponent.

En aquest mateix capítol, en els apartats 4.4 i 4.5, tractem més a fons la segmentació en unitats lèxiques (tokenització) i la segmentació en oracions. El que sí que avancem ara és que al **PlaintextCorpusReader** se li pot indicar quin tokenitzador i segmentador ha de fer servir. Això es pot fer de la següent manera (programa-4-3.py) (necessitareiudescarregar l'arxiu catalan.pickle que teniu en la secció d'arxius d'aquest capítol):

```
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.tokenize import RegexpTokenizer

segmentador= nltk.data.load("catalan.pickle")
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')

corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)
for oracio in corpus.sents():
    print(oracio)
```

Fixeu-vos que hem definit un segmentador (que carreguem de catalan.pickle) i un tokenitzador basat en expressions regulars, i que fem servir aquests nous elements quan creem el **PlaintextCorpusReader**. Ara hem arreglat l'aspecte dels apòstrofs en la tokenització i la sortida (mostrem només un fragment) és:

['El', 'ple', 'de', "l'", 'Ajuntament', "d'", 'Òrrius', 'a', 'la', 'sessió', 'extraordinària', 'celebrada', 'el', 'dia', '20', 'de', 'gener', 'de', '2015', 'va', 'aprovar', 'inicialment', 'el', 'projecte', "d'", 'obra', '.']

En el apartat 4.4 veurem amb més detall el tema de les expressions regulars. Comparem ara les expressions regulars del tokenitzador per defecte:

\w+|[^\w\s]+

i el que hem creat nosaltres:

[ldsmLDSM]\'|\w+|[^\w\s]+

Hem afegit la part [ldsmLDSM]\' que defineix els tokens formats per: l', d', s', m', L', D', S', M'.

4.3. Ocurrències (tokens) i tipus (types)

Modificant molt lleugerament el programa-4-3.py, per tal que ens doni totes les paraules (o tokens) en comptes de totes les oracions i perquè ens proporcioni el nombre total de paraules (programa-4-4.py):

```
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.tokenize import RegexpTokenizer

segmentador= nltk.data.load("catalan.pickle")
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')

corpus = PlaintextCorpusReader(".", 'DOGC-2015-cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)
for paraula in corpus.words():
    print(paraula)
print("TOTAL PARAULES:",len(corpus.words()))
```

El programa s'està una bona estona mostrant paraules per pantalla i després ens dóna el nombre de paraules totals:

```
...
5524
,
d'
11
.
12
.
```

```
2009

),

TOTAL PARAULES: 2448870
```

Si miréssim amb deteniment la llista veuríem que moltes de les paraules ocorren més d'un cop (en l'apartat 4.6 veurem com calcular la freqüència i la distribució de freqüències de les paraules). Si ara el que volem és obtenir una llista de paraules diferents, podem fer servir la instrucció set, com podem observar en el programa-4-5.py:

```
import nltk

from nltk.corpus.reader.plaintext import PlaintextCorpusReader

from nltk.tokenize import RegexpTokenizer


segmentador= nltk.data.load("catalan.pickle")

tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')


corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)


ocurrencies=corpus.words()

tipus=set(ocurrencies)


print("OCURRENCIES:",len(ocurrencies))

print("TIPUS:",len(tipus))
```

```
Que ens dona la sortida:
```

```
OCURRENCIES: 2448870

TIPUS: 34872
```

Per resumir, podem dir que les ocurrències són el nombre total de paraules que apareixen en el corpus i els tipus el nombre de paraules diferents que apareixen en el corpus. Cal tenir en compte, però,que no podem parlar estrictament de paraules, ja que també s'inclouen els signes de puntuació, les xifres, etc. Podem calcular un índex de riquesa lèxica dividint el nombre d'ocurrències entre el nombre de tipus. Ho veiem al programa-4-6.py:

```
import nltk

from nltk.corpus.reader.plaintext import PlaintextCorpusReader

from nltk.tokenize import RegexpTokenizer


segmentador= nltk.data.load("catalan.pickle")

tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')


corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)


ocurrencies=corpus.words()

tipus=set(ocurrencies)

riquesalexica=len(ocurrencies)/len(tipus)
```

```
print("OCURRENCIES:",len(ocurrencies))
print("TIPUS:",len(tipus))
print("RIQUESA LÊXICA:",round(riquesalexica,2))
```

Que ens dona la següent sortida:

```
OCURRENCIES: 2448870
TIPUS: 34872
RIQUESA LÊXICA: 70.22
```

Cosa que indica que cada paraules de mitjana es fa servir 70 vegades. Experimenta una mica amb diferents tipus de text per veure com varia aquest índex de riquesa lèxica.

4.4. Tokenització

La segmentació en unitats lèxiques o tokenització, consisteix a dividir el text en unitats més petites (que sovint coincideixen amb paraules). Tot i que es tracta d'una tasca molt bàsica i necessària per a poder portar a terme tasques d'anàlisi més avançades, aquesta tasca presenta nombrosos problemes que no són fàcils de solucionar. Hi ha nombrosos treballs sobre aquesta àrea entre els quals es poden destacar els treballs de Grefenstette i Tapanainen (1994) i Mikheev (2002). Sigui com sigui, actualment es pot considerar que aquesta tasca es resol de manera satisfactòria i no hi ha una recerca activa per millorar-la. En aquest apartat aprendrem algunes tècniques per a segmentar el text en unitats lèxiques i anirem observant els diferents problemes que hi apareixen i com es poden solucionar. Les proves dels diferents sistemes els farem al l'oracio:

El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació.

Tot i que el resultat desitjat de la tokenització pot dependre de les tasques concretes, la sortida desitjada del nostre sistema hauria de ser una cosa del següent estil:

```
['El','Sr.','Martínez','arribarà','demà','d'','Alacant','amb','la','R.E.N.F.E','a','les','22.30','h.',i','s'','haurà','d'','allotjar','a','l'','hotel','de','l'','estació',.']
```

El primer tokenitzador que provarem fa servir la instrucció **split()**, que divideix una cadena segons el separador que s'indiqui, i si no s'indica res, per espais. El podem veure al programa-4-7.py:

```
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."

tokens=oracio.split()
print(tokens)
```

que ens dona la següent sortida, que no és exactament la que volíem:

```
['El', 'Sr.', 'Martínez', 'arribarà', 'demà', "d'Alacant", 'amb', 'la', 'R.E.N.F.E.', 'a', 'les', '22.30', 'h.', 'i', "s'haurà", "d'allotjar", 'a', "l'hotel", 'de', "l'estació."]
```

NLTK proporciona una sèrie de tokenitzadors que presentem a continuació:

WhitespaceTokenizer

Separa el text per espais en blanc, com ho hem fet en l'exemple anterior. En aquest cas els espais en blanc poden ser els caràcters: espai en blanc, tabulador i nova línia. Veiem-ho (programa-4-8.py):

```
import nltk
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."
tokens=nltk.tokenize.WhitespaceTokenizer().tokenize(oracio)
print(tokens)
```

Que proporciona com a sortida:

```
['El', 'Sr.', 'Martínez', 'arribarà', 'demà', "d'Alacant", 'amb', 'la', 'R.E.N.F.E.', 'a', 'les', '22.30', 'h.', 'i', "s'haurà", "d'allotjar", 'a', "l'hotel", 'de', "l'estació."]
```

Aprofito aquest exemple per explicar diverses maneres d'importar un tokenitzador, o qualsevol mètode d'una classe determinada. en el cas anterior hem importat tot l'NLTK i hem cridat al mètode fent:

```
tokens=nltk.tokenize.WhitespaceTokenizer().tokenize(oracio)
```

Això també es pot fer de la següent manera alternativa (programa-4-8b.py)(fixeu-vos com accedim ara al mètode tokenize)

```
from nltk.tokenize import WhitespaceTokenizer
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."
tokens=WhitespaceTokenizer().tokenize(oracio)
print(tokens)
```

i també de la següent manera, donant un nom a la classe que pot ser qualsevol) (programa-4-8c.py)

```
from nltk.tokenize import WhitespaceTokenizer as tokenitzador
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."
tokens=tokenitzador().tokenize(oracio)
print(tokens)
```

SpaceTokenizer

És igual que l'anterior, però en aquest cas l'únic caràcter que es té en compte és el d'espai en blanc (" "). Equival a split(" "). En el nostre exemple la sortida seria exactament la mateixa i no cal proporcionar ni el codi ni el programa.

TreebankWordTokenizer

Aquest tokenitzador fa servir les convencions del Penn Treebank corpus, que és un corpus anotat de l'anglès creat als any 1980 a partir d'articles del Wall Street Journal. Com que és per a l'anglès, no funcionarà del tot bé per al català i per això en l'exemple poso una oració de l'anglès (programa-4-9.py):

```
from nltk.tokenize import TreebankWordTokenizer as tokenitzador
oracio="We need to conduct an assessment to learn whether a student's dificulties are because he or she can't or won't complete assignments."
```

```
tokens=tokenitzador().tokenize(oracio)
print(tokens)
```

i la sortida serà:

```
['We', 'need', 'to', 'conduct', 'an', 'assessment', 'to', 'learn', 'whether', 'a', 'student', "'s", 'dificulties', 'are', 'because', 'he', 'or', 'she', 'ca', "n't", 'or', 'wo', "n't", 'complete', 'assignments', '.']
```

Aquest tokenitzador es fa servir força per a l'anglès, i per aquest motiu s'ha creat una funció específica que fa de wrapper (*wrapper function*) per simplificar el seu ús (programa-4-9b.py):

```
from nltk.tokenize import word_tokenize
oracio="We need to conduct an assessment to learn whether a student's dificulties are because he or she can't or won't complete assignments."
tokens=word_tokenize(oracio)
print(tokens)
```

que proporciona exactament la mateixa sortida.

RegexpTokenizer

A l'apartat anterior (4.3. Ocurrències (tokens) i tipus (types)) ja vam veure un exemple del tokenitzador per expressions regulars. Aquest és un tipus de tokenitzador que ens permet un control total sobre el procés de tokenització. Clar que per treure'n el màxim profit cal dominar les expressions regulars de Python. Podeu trobar una explicació detallada a Regular Expression HOWTO i també un bon resum aquí. Un altre bon recurs per treballar amb expressions regulars és la web http://regexr.com/ . Aquí podeu provar les expressions regulars sobre text i també, si aneu al menú de l'esquerra podeu trobar un resum (*Cheatsheer*).

Veiem ara alguns exemples, començant pel programa-4-10.py

```
from nltk.tokenize import RegexpTokenizer
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')
tokens=tokenitzador.tokenize(oracio)
print(tokens)
```

Que ens ofereix la següent sortida:

```
['El', 'Sr', '.', 'Martínez', 'arribarà', 'demà', "d'", 'Alacant', 'amb', 'la', 'R', '.', 'E', '.', 'N', '.', 'F', '.', 'E', '.', 'a', 'les', '22', '.', '30', 'h', '.', 'i', "s'", 'haurà', "d'", 'allotjar', 'a', "l'", 'hotel', 'de', "l'", 'estació', '.']
```

Com podem observar, ens separa Sr del punt (.) i realment voldríem tenir Sr. com a token, i el mateix passa amb h.. També tokenitza incorrectamente R.E.N.F.E. Podríem solucionar això modificant l'expressió regular (programa-4-10b.py) de manera que afegim aquestes dues unitats com a tokens.

```
tokenitzador=RegexpTokenizer('Sr\.|h\.|R\.E\.N\.F\.E\.|[ldsmLDSM]\'|\w+|[^\w\s]+')
```

que ens retornaria la tokenització correcta d'aquestes dues unitats:

['El', 'Sr', '.', 'Martínez', 'arribarà', 'demà', "d'", 'Alacant', 'amb', 'la', 'R.E.N.F.E.', 'a', 'les', '22', '.', '30', 'h', '.', 'i', "s'", 'haurà', "d'", 'allotjar', 'a', "l'", 'hotel', 'de', "l'", 'estació', '.']

El fet d'afegir una llista d'abreviatures corrents és força habitual, però no podem esperar tenir una llista prou completa d'acrònims. Per aquest motiu, hem d'intentar expressar els acrònims d'una manera més general, com per exemple la que presentem al programa-4-10c.py

```
tokenitzador=RegexpTokenizer('Sr\.|h\.|[A-Z\.]{2,}|\w+|[^\w\s]+')
```

que ens ofereix la mateixa sortida però que ara contempla qualsevol acrònim tipus R.E.N.F.E.

Per defecte, el tokenitzador per expressions regulars considera que el que expressem són els tokens, però podem definir el que són els separadors de tokens amb el modificador gaps=True. En el programa-4-10.py podem veure un exemple simple.

```
import re
from nltk.tokenize import RegexpTokenizer
oracio="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació."
tokenitzador=RegexpTokenizer('\s+',gaps=True)
tokens=tokenitzador.tokenize(oracio)
print(tokens)
```

Que es dona la següent sortida (que coincideix amb la del tokenitzador per espais en blanc):

['El', 'Sr.', 'Martínez', 'arribarà', 'demà', "d'Alacant", 'amb', 'la', 'R.E.N.F.E.', 'a', 'les', '22.30', 'h.', 'i', "s'haurà", "d'allotjar", 'a', "l'hotel", 'de', "l'estació."]

4.5. Segmentació

En la secció anterior hem vist com separar el text en unitat lèxiques, procés que també rep el nom de tokenització. En aquesta secció veurem com podem separar un paràgraf en segments, que són unitats semblants a oracions. Aquest procés rep el nom de segmentació.

En aquesta secció farem servir dos segments de prova, un que no presentarà grans problemes:

Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa.

i un altre que sí que en presenta.

El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352.

PunktSentenceTokenizer

Aquest és un segmentador senzill que bàsicament segmenta per punts. Per veure com funciona provem el programa-4-11.py.

```
from nltk.tokenize import PunktSentenceTokenizer
```

```
paragraf1="Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa."
```

paragraf2="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352."

segmentador=PunktSentenceTokenizer()

segments1=segmentador.tokenize(paragraf1)

print(segments1)

segments2=segmentador.tokenize(paragraf2)

print(segments2)

Que proporciona la següent sortida:

['Avui fa un dia molt bonic.', "Demà l'Albert anirà a dinar a casa."]

['El Sr.', "Martínez arribarà demà d'Alacant amb la R.E.N.F.E.", "a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació.", "L'endemà visitarà al Dr.", "Rovira a l'Av.", 'Tibidabo.', 'La tornada la farà en avió en el vol AF.352.']

sent_tokenize()

El procés de segmentació també es pot fer amb sent_tokenizer(), que crida a una instància especial del PunktSenteceTokenizer que ha estat entrenada i que funciona força bé per a diverses llengües europees. Es tracta d'una implementació de l'algorisme de Kiss and Strunk (2006). Veiem-ho al programa-4-11b.py.

from nltk.tokenize import sent_tokenize

paragraf1="Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa."

paragraf2="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352."

segments1=sent_tokenize(paragraf1)

print(segments1)

segments2=sent_tokenize(paragraf2)

print(segments2)

Que ens ofereix una sortida molt millor, tot i que no perfecta:

['Avui fa un dia molt bonic.', "Demà l'Albert anirà a dinar a casa."]

["El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E.", "a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació.", "L'endemà visitarà al Dr. Rovira a l'Av.", 'Tibidabo.', 'La tornada la farà en avió en el vol AF.352.']

Carregar un segmentador concret per a una llengua determinada

Amb les dades de l'NLTK es distribueixen una sèrie de segmentadors per a llengües determinades. Concretament es distribueixen els següents: txec, finlandès, noruec, espanyol, danès, francès, polonès, suec, holandès, alemany, portuguès, turc, anglès, grec, estonià, italià, eslovè. No es distribueix un per al català, però en aquesta mateixa secció aprendrem a crear un d'específic per al català. En el programa-4-12.py carreguem un d'específic per a l'anglès:

import nltk.data

paragraf="Today Mr. Smith and Ms. Johanson will meet at St. Patrick church."

segmentador=nltk.data.load("tokenizers/punkt/PY3/english.pickle")

```
segments=segmentador.tokenize(paragraf)
print(segments)
```

Que proporciona la sortida:

['Today Mr. Smith and Ms. Johanson will meet at St. Patrick church.']

Entrenament d'un segmentador

L'algorisme de Kiss and Strunk (2006) permet entrenar un segmentador a partir de text sense cap tipus d'anotació. NLTK implementa aquest algorisme. Anem a aprendre un segmentador per al català (de fet crear un catalan.pickle) a partir del corpus de DOGC corresponent a l'any 2015. El programa-4-13.py implementa aquest aprenentatge:

```
import nltk.tokenize.punkt
import pickle
import codecs
segmentador = nltk.tokenize.punkt.PunktSentenceTokenizer()
text = codecs.open("DOGC-2015-cat.txt","r","utf8").read()
segmentador.train(text)
out = open("catalan.pickle","wb")
pickle.dump(segmentador, out)
out.close()
```

Crearem el catalan.pickle que podrem fer servir com a segmentador en el programa-4-12b.py):

```
import nltk.dataimport nltk.data
paragraf="Today Mr. Smith and Ms. Johanson will meet at St. Patrick church."

segmentador=nltk.data.load("catalan.pickle")
paragraf1="Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa."

paragraf2="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352."

segments1=segmentador.tokenize(paragraf1)
print(segments1)
segments2=segmentador.tokenize(paragraf2)
print(segments2)
paragraf="Today Mr. Smith and Ms. Johanson will meet at St. Patrick church."
segmentador=nltk.data.load("catalan.pickle")
paragraf1="Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa."
paragraf2="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352."
```

```
segments1=segmentador.tokenize(paragraf1)
print(segments1)
segments2=segmentador.tokenize(paragraf2)
print(segments2)
```

Que ofereix la següent sortida:

['Avui fa un dia molt bonic.', "Demà l'Albert anirà a dinar a casa."]

["El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E.", "a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació.", "L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo.", 'La tornada la farà en avió en el vol AF.352.']

Funciona del tot correctament el segmentador que hem entrenat?

Personalitzar el segmentador

Podem personalitzar el segmentador entrenat per afegir noves abreviatures o acrònims que no han estat detectats en el procés d'entrenament. Ho podem fer específicament per a un programa determinat, com en el programa-4-12.c.py, on carreguem el catalan.pickle que hem entrenat i afegim R.E.N.F.E. (fixeu-vos que ho fem en minúscules i sense el punt final):

```
import nltk.data
segmentador=nltk.data.load("catalan.pickle")
abreviacions_extra = ['r.e.n.f.e']
segmentador._params.abbrev_types.update(abreviacions_extra)
paragraf1="Avui fa un dia molt bonic. Demà l'Albert anirà a dinar a casa."

paragraf2="El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació. L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo. La tornada la farà en avió en el vol AF.352."

segments1=segmentador.tokenize(paragraf1)
print(segments1)
segments2=segmentador.tokenize(paragraf2)
print(segments2)
```

i ara la sortida ja està millor segmentada:

["El Sr. Martínez arribarà demà d'Alacant amb la R.E.N.F.E. a les 22.30 h. i s'haurà d'allotjar a l'hotel de l'estació.", "L'endemà visitarà al Dr. Rovira a l'Av. Tibidabo.", 'La tornada la farà en avió en el vol AF.352.']

En els fitxers adjunts podeu trobar una llista d'abreviacions. Ara el que farem serà carregar el catalan.pickle que hem entrenat i modificar-lo afegint la llista d'abreviatures i acrònims del fitxer. Gravarem aquest nou segmentador com a catalan-mod.pickle. (programa-4-14.py)

```
import nltk.data
import codecs
```

```
import pickle
segmentador=nltk.data.load("catalan.pickle")
fitxer_abreviacions=codecs.open("abreviatures.txt","r",encoding="utf-8")
abreviacions_extra =[]
for abreviacio in fitxer_abreviacions.readlines():
    abreviacio=abreviacio.rstrip()
    abreviacions_extra.append(abreviacio)
segmentador._params.abbrev_types.update(abreviacions_extra)
out = open("catalan-mod.pickle","wb")
pickle.dump(segmentador, out)
out.close()
```

Modifiqueu el programa-4-12b.py per a que carregui aquest nou segmentador i verifiqueu si funciona correctament.

4.6. Freqüències i distribució de freqüències

En aquest apartat aprendrem a fer alguns càlculs senzills sobre corpus: freqüències absolutes i freqüències relatives, distribucions de freqüència i a trobar les col.locacions més freqüents d'un corpus.

4.6.1. Freqüència absoluta

Entenem per freqüència absoluta el nombre total de vegades que apareix una determinada unitat lèxica en el nostre corpus.

El càlcul de la freqüència absoluta d'una paraula és senzill: podem utilitzar un diccionari per a posar com a clau les paraules i anar incrementant el valor del diccionari cada cop que apareix la paraula. En el programa següent (programa-4-15.py) podem veure una implementació senzilla d'aquesta idea:

```
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.tokenize import RegexpTokenizer
segmentador= nltk.data.load("catalan.pickle")
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')
corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)
frequencia={}
for paraula in corpus.words():
    frequencia[paraula]=frequencia.get(paraula,0)+1
for clau in frequencia.keys():
    print(frequencia[clau],clau)
```

i per pantalla ens mostrarà les paraules i les freqüències, però d'una manera desordenada, ja que els diccionaris de Python són estructures de dades sense un ordre determinat. NLTK proporciona una funció FreqDist que facilita molt el càlcul de freqüències.Veiem el seu ús en el programa-4-16.py:

```
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.tokenize import RegexpTokenizer
from nltk import FreqDist
segmentador= nltk.data.load("catalan.pickle")
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')

corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)

frequencia=FreqDist(corpus.words())

for mc in frequencia.most_common():
    print(mc)
```

Ara sí que ens proporciona les paraules i la freqüència de cada paraula endreçada de més freqüent a menys freqüent. Com que n'hi ha moltes, podem modificar fàcilment el programa perquè ens proporcioni les 25 més freqüents, canviant la línia:

```
for mc in frequencia.most_common():
```

per

```
for mc in frequencia.most_common(25):
```

4.6.2. Freqüència relativa

La freqüència absoluta d'una paraula en un determinat corpus no ens dóna informació real sobre si la paraula és molt freqüent o no, perquè això dependrà de la mida del corpus. Que una paraula aparegui, diguem-ne, 22 vegades en el nostre corpus, no ens diu res, ja que si el corpus és molt gran potser aquest valor de freqüència és petit.

La freqüència relativa d'una paraula en un corpus és el nombre de vegades que hi apareix dividida pel nombre total de paraules en el corpus.

FreqDist ens facilita molt el càlcul de la freqüència relativa ja que la podem consultar amb el mètode freq(). Ho podem veure en el programa-4-17.py, que és una modificació de l'anterior. Ara guardem al fitxer frequencies.txt les paraules endreçades per freqüència i mostrem la freqüència absoluta i la relativa de cada paraula:

```
import nltk
from nltk.corpus.reader.plaintext import PlaintextCorpusReader
from nltk.tokenize import RegexpTokenizer
from nltk import FreqDist
import codecs
```

```
segmentador= nltk.data.load("catalan.pickle")
tokenitzador=RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')

corpus = PlaintextCorpusReader(".", 'DOGC-2015-
cat.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)

frequencia=FreqDist(corpus.words())

sortida=codecs.open("frequencies.txt","w",encoding="utf-8")

for mc in frequencia.most_common():
    paraula=mc[0]
    frequencia_absoluta=mc[1]
    frequencia_relativa=frequencia.freq(paraula)
    cadena=str(frequencia_absoluta)+"\t"+str(frequencia_relativa)+"\t"+paraula
    sortida.write(cadena+"\n")
```

I la sortida (mostrem només les 25 primeres):

```
228896 0.09347004945137961 de
139175 0.05683233491365405 ,
85450 0.0348936448239392 .
78342 0.03199108160090164 la
63452 0.02591072617166285 d'
54381 0.022206568743951292 i
51190 0.0209035187657981 l'
47788 0.01951430659855362 a
44098 0.018007489168473622 del
37175 0.015180470992743592 /
28550 0.011658438381784252 :
27148 0.011085929428675267 que
26976 0.011015692952259614 el
26544 0.010839285058006345 per
25021 0.010217365560442164 les
22050 0.00900415293584388 en
18132 0.007404231339352436 2015
15739 0.006427045943639311 -
15648 0.006389885947396146 al
```

14972 0.006113840261018347)

14063 0.005742648650193763 es

13936 0.005690787996096159 (

12529 0.005116237284951835 amb

11781 0.004810790282865158 s'

10635 0.004342819341165517 Catalunya

4.6.3. La llei de Zipf

La Llei de Zipf afirma que donat un corpus, la freqüència d'una paraula és inversament proporcional a la seva posició a la taula de freqüències (*rank*). Amb la seva llei, Zipf (1949) afirma que hi ha una constant k que es pot calcular multiplicant la freqüència de qualsevol paraula per la seva posició a la taula (rank) ($k = f \cdot r$).

En el programa següent (programa-4-18.py) avaluem la llei de Zipf amb les 50 paraules més freqüents del corpus Cess_cat:

```
import nltk
from nltk.corpus import cess_cat
import re
import codecs


paraules=cess_cat.words()
freqdist=nltk.FreqDist(paraules)
sortida=codecs.open("sortida.txt","w", encoding="utf-8")


posicio=0
p = re.compile('\w+')
for mc in freqdist.most_common(50):
    if p.match(mc[0]):
        posicio+=1
        freq=mc[1]
        fxr=posicio*freq
        sortida.write(mc[0]+"\t"+str(freq)+"\t"+str(posicio)+"\t"+str(fxr)+"\n")
```

En el fitxer sortida.txt podem observar les paraules amb la seva freqüència, posició i resultat del producte de la freqüència i la posició (que tendeix a ser constant):

de 23684 1 23684

la 15695 2 31390

que 12703 3 38109

i 11819 4 47276

el 9749 5 48745

a 9568 6 57408

l' 8892 7 62244

d' 6970 8 55760

del 5826 9 52434

en 5783 10 57830

per 5378 11 59158

un 5247 12 62964

les 5219 13 67847

va 5217 14 73038

ha 5114 15 76710

els 4207 16 67312

una 3895 17 66215

amb 3755 18 67590

es 3265 19 62035

al 2843 20 56860

no 2646 21 55566

El 2551 22 56122

dels 2445 23 56235

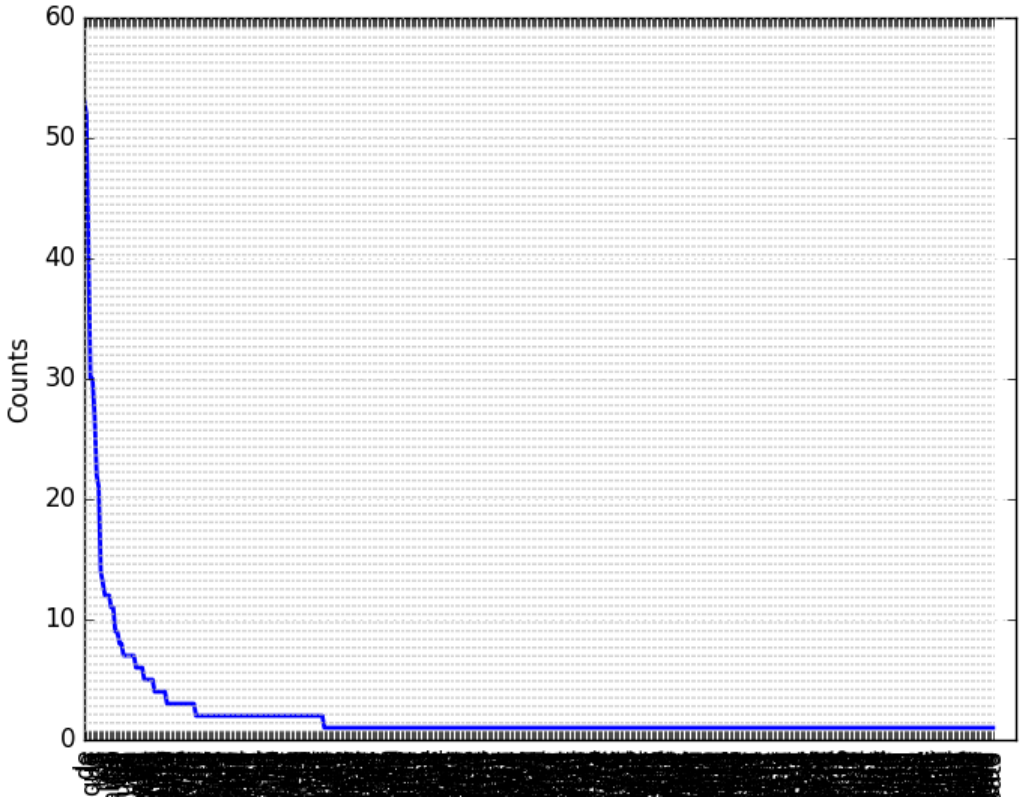
s' 2327 24 55848

és 2062 25 51550

Tot i que la llei només mostra una tendència i no és exacta, recalca el fet que en un corpus hi ha molt poques paraules molt freqüents i moltes paraules poc freqüents. En el programa següent (programa-4-19.py) grafiquem aquest fenomen amb un subconjunt del corpus Cess_cat de 1.000 paraules. Com podem observar al gràfic el principi esmentat s'acompleix, és a dir, que molt poques paraules apareixen moltes vegades i que moltes apareixen poques vegades.

Per poder executar aquest programa ens hem d'assegurar de tenir instal·lades les llibreries scipy i matplotlib. Recorda que es poden instalar amb pip (o pip3).

```
import nltk
from nltk.corpus import cess_cat
import pylab
paraules=cess_cat.words()
paraules1=paraules[:1000]
freqdist=nltk.FreqDist(paraules1)
freqdist.plot()
```


60
50
40
30
20
10
0
Counts