Introducció

En aquest mòdul estudiarem la tasca anomenada etiquetatge morfosintàctic (en anglès part-of-speech tagging o POS-tagging). Aquesta tasca consisteix a assignar a cada paraula d'un text una categoria gramatical i altra informació addicional (com poden ser diverses subcategories, el lema associat, etc.) Aquesta és una tasca fonamental en el processament del llenguatge natural, encara que no està exempta de problemes que no estan encara totalment resolts. El llenguatge natural és ambigu des de molts punts de vista, i també ho és en el morfosintàctic. Una determinada forma (com ara casa) pot tenir diverses interpretacions morfosintàctiques, pot ser un substantiu comú femení singular (amb lema casa) i també una forma de present o d'imperatiu del verb casar. Els etiquetadors morfosintàctics hauran d'intentar donar la interpretació adequada segons el context d'utilització; per tant hauran de desambiguar les diferents possibilitats.

Abans d'abordar l'etiquetatge morfosintàctic veurem també l'anàlisi morfològica, és a dir, l'anàlisi que ens permet determinar el lema i la categoria gramatical (i altres subcategorizaciones) d'una determinada forma. Un cop siguem capaços de fer l'anàlisi morfològica, passarem a estudiar diverses tècniques que ens permetran etiquetar textos des del punt de vista morfosintàctic i que intentaran desambiguar (amb més o menys èxit) les diferents possibilitats.

- D'aquesta unitat disposes dels següents arxius:
- Aquests mateixos apunts en PDF: 05-Etiquetatge_morfosintàctic-cat.pdf
- Els programes i arxius necessaris: programas-cat.zip
- L'arxiu Notebook de Jupyter; cap5-cat.ipynb
- L'enllaç al notebook en Google Colaboratory: https://colab.research.google.com/github/aoliverg/python/blob/master/notebooks/cap5-cat.ipynb

5.1. Morfologia computacional

5.1.1. El formalisme de descomposició morfològica

Hi ha tot un seguit de formalismes per descriure la morfologia d'una llengua. En aquest capítol només presentarem un, que és senzill d'implementar i ens servirà per comprendre els mecanismes fonamentals: el formalisme de descomposició morfològica (Alshawi, 1992). La idea bàsica d'aquest formalisme és senzilla i es basa en dos tipus de coneixement:

- Un diccionari que conté informació morfosintàctica sobre la base o la paraula que es considera forma de referència.
- Regles que contenen informació sobre la morfologia de la llengua.

En el programa-5-1.py podem veure una primera implementació molt senzilla d'aquesta idea:

```
t1="o"

t2="es"

t3="a"

t4="em"

t5="eu"

t6="en"


print(arrel+t1)

print(arrel+t2)

print(arrel+t3)

print(arrel+t4)

print(arrel+t5)

print(arrel+t6)
```

Si executem aquest programa, obtenim la següent sortida:

canto

cantes

canta

cantem

canteu

canten

En aquest programa definim una sèrie de variables: una que conté el lema d'un verb i altres que contenen les terminacions de present d'indicatiu. Mitjançant l'operador + que concatena cadenes obtenim les formes corresponents al present d'indicatiu.

Seguint aquesta idea bàsica idearem un formalisme per expressar les regles (que a més de les formes volem que ens donin una etiqueta de categoria gramatical i subcategoritzacions). També idearem un formalisme per expressar el diccionari de lemes, que a més del propi lema ens indicarà el tipus de flexió que segueix (és a dir, el paradigma flexiu).

Com a formalisme per a les regles proposem una sèrie de valors separats per dos punts (:):

```
terminació_forma:terminació_lema:etiqueta:paradigma
```

Per exemple:

o:ar:VMIP1S:V1

es:ar:VMIP2S:V1

a:ar:VMIP3S:V1

em:ar:VMIP1P:V1

eu:ar:VMIP2P:V1

en:ar:VMIP3P:V1

Per les etiquetes fem servir les etiquetes EAGLES per al català. Al fitxer regles.txt podem observar un conjunt més extens de regles morfològiques.

Com a formalisme per al diccionari de lemes seguirem una idea similar:

lema:paradigma

Que en l'exemple seria:

cantar:V1

Al fitxer diccionari.txt podem observar un diccionari més complet.

Ara necessitem un programa que ens permeti generar totes les formes a partir de les regles i el diccionari (programa-5-2.py)

```
fregles=open("regles.txt","r")
```

```
regles=[]

while True:
  linia=fregles.readline().rstrip()
 if not linia:break
 regles.append(linia)
fregles.close()

fdiccionari=open("diccionari.txt","r")

while True:
 linia=fdiccionari.readline().rstrip()
 if not linia:break
 (lema,tipus)=linia.split(":")
 for regla in regles:
  (tf,tl,etiqueta,tipus2)=regla.split(":")
  if ((tipus2 == tipus)&(lema.endswith(tl))):
   print(lema[0 : (len(lema)-len(tl))]+tf,lema,etiqueta)

fdiccionari.close()
```

que ens proporciona com a sortida:

```
canto cantar VMIP1S
cantes cantar VMIP2S
canta cantar VMIP3S
cantem cantar VMIP1P
canteu cantar VMIP2P
canten cantar VMIP3P
llegeixo llegir VMIP1S
llegeixes llegir VMIP2S
.....
```

Estem parlant d'anàlisi morfològica, però en realitat el que hem fet és un programa que genera formes amb el seu lema i una etiqueta morfosintàctica, el que anomenem diccionari morfològic. Els programes d'anàlisi morfològica sovint funcionen amb un diccionari morfològic.

Al següent programa (programa-5-3.py) carreguem aquest diccionari i duem a terme l'anàlisi de les paraules que indica l'usuari (el programa finalitza quan l'usuari introdueix un espai en blanc). El diccionari que hem creat en el programa anterior el podeu trobar en els arxius del capítol (diccionari-curt-cat.txt):

```
import codecs
diccionari={}
```

```
arxiu_diccionari=codecs.open("diccionari-curt-cat.txt","r",encoding="utf-8")

for entrada in arxiu_diccionari:
    entrada=entrada.rstrip()
    camps=entrada.split(" ")
    forma=camps[0]
    lema=camps[1]
    etiqueta=camps[2]
    diccionari[forma]=diccionari.get(forma,"")+":"+lema+"\t"+etiqueta

while 1:
    paraula=input()
    if paraula==" ":
        break
    if paraula in diccionari:
        print("ANALISI:",paraula,diccionari[paraula])
    else:
        print("PARAULA DESCONEGUDA")
```

que ofereix, per exemple, la següent sortida:

```
canto
ANALISI: canto :cantar VMIP1S
casa
PARAULA DESCONEGUDA
```

Si en comptes d'un diccionari tant petit fem servir un de més gran, el programa serà capaç d'analitzar moltes més paraules. Descarregueu l'arxiu diccionari-cat.txt, que és un gran diccionari morfològic extret de l'analitzador Freeling.

5.2. Anàlisi morfològica

En aquest apartat construirem un analitzador morfològic, és a dir, un programa que és capaç de donar-nos les anàlisis morfològiques de totes les paraules d'un text. Per a les paraules ambigües des del punt de vista morfosintàctic, ens retornarà totes les possibles interpretacions. Amb el que hem fet fins ara tenim tots els components:

- Un programa que carregui el diccionari morfològic del català (el programa-5-3.py)
- Un programa capaç de llegir un document i segmenar-lo en oracions i tokenitzar-lo (per exemple, el programa-4-3.py)

Al programa-5-4.py podem observar una primera versió del programa (que analitzarà l'arxiu noticia.txt, que conté un fragment de notícia publicada al diari Ara). Utilitza també el catalan-mod.pickle que hem creat al capítol anterior).

```
import codecs
import nltk
```

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

diccionari={}
arxiu_diccionari=codecs.open("diccionari-cat.txt","r",encoding="utf-8")

for entrada in arxiu_diccionari:
    entrada=entrada.rstrip()
    camps=entrada.split(" ")
    forma=camps[0]
    lema=camps[1]
    etiqueta=camps[2]
    if forma in diccionari:
        diccionari[forma]=diccionari.get(forma,"")+" "+lema+" "+etiqueta
    else:
        diccionari[forma]=lema+" "+etiqueta

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

corpus = PlaintextCorpusReader(".", 'noticia.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)

for forma in corpus.words():
    if forma in diccionari:
        info=diccionari[forma]
    else:
        info="DESCONEGUDA"
    print(forma+" "+info)
```

que ens proporciona la sortida (mostrem únicament un fragment):

```
Després DESCONEGUDA
d' de SP
un un DI0MS0 un PI0MS00
debat debat NCMS000 debatre VMIP3S0 debatre VMM02S0
llampec llampec NCMS000
lluny lluny AQ0MS00 lluny RG
de de NCFS000 de SP
l' el DA0CS0 el PP3CSA0
esperpèntic esperpèntic AQ0MS00
```

comitè comitè NCMS000

federal federal AQ0CS00

que que CS que PR0CN00 que RG

va anar VAIP3S0 anar VMIP3S0 va AQ0MS00

derrocar derrocar VMN0000

Pedro DESCONEGUDA

Sánchez DESCONEGUDA

, DESCONEGUDA

Com veiem, hi ha alguns problemes, com per exemple:

- Les paraules que estan a la primera posició d'una oració, i per tant estan escrites en majúscules, tot i estar al diccionari, les etiqueta com a desconegudes. Per solucionar això, primer buscarem al diccionari les paraules tal i com apareixen al text, si no les troba, llavors les buscarà passada a minúscula, i si tot i així no la troba, la marcarpa com a desconeguda.
- Els signes de puntuació els marca com a desconeguts, ja que són tokens que no apareixen al diccionari morfològic. La solució és senzilla i consisteix a incloure els signes de puntuació al diccionari morfològic. Al diccionari2-cat.txt hem afegit les següents entrades:

" " Fe

' ' Fe

. . Fp

, , Fc

; ; Fx

: : Fd

((Fpa

)) Fpt

[[Fca

]] Fct

Veiem la nova implementació al programa-5-5.py.

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

diccionari={}
arxiu_diccionari=codecs.open("diccionari2-cat.txt","r",encoding="utf-8")

for entrada in arxiu_diccionari:
    entrada=entrada.rstrip()
    camps=entrada.split(" ")
```

```
    forma=camps[0]
    lema=camps[1]
    etiqueta=camps[2]
    if forma in diccionari:
        diccionari[forma]=diccionari.get(forma,"")+" "+lema+" "+etiqueta
    else:
        diccionari[forma]=lema+" "+etiqueta

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

corpus = PlaintextCorpusReader(".", 'noticia.txt',word_tokenizer=tokenitzador,sent_tokenizer=segmentador)

for forma in corpus.words():
    if forma in diccionari:
        info=diccionari[forma]
    elif forma.lower() in diccionari:
        info=diccionari[forma.lower()]
    else:
        info="DESCONEGUDA"
    print(forma+" "+info)
```

que dóna com a sortida:

```
Després després RG
d' de SP
un un DI0MS0 un PI0MS00
debat debat NCMS000 debatre VMIP3S0 debatre VMM02S0
llampec llampec NCMS000
lluny lluny AQ0MS00 lluny RG
de de NCFS000 de SP
l' el DA0CS0 el PP3CSA0
esperpèntic esperpèntic AQ0MS00
comitè comitè NCMS000
federal federal AQ0CS00
que que CS que PR0CN00 que RG
va anar VAIP3S0 anar VMIP3S0 va AQ0MS00
derrocar derrocar VMN0000
Pedro DESCONEGUDA
```

Sánchez DESCONEGUDA

Encara queden problemes per resoldre, com els noms propis, algun aspecte malament tokenitzat, etc. També poden aparèixer paraules desconegudes que siguin correctes, però que no estiguin recollides al diccionari morfològic, etc. En propers apartats d'aquest mateix capítol anirem presentant solucions a aquestes qüestions.

5.3. Etiquetatge morfosintàctic

5.3.1. Introducció

En aquest mòdul estudiarem la tasca anomenada etiquetatge morfosintàctic (en anglès part-of-speech tagging o POS-tagging). Aquesta tasca consisteix a assignar a cada paraula d'un text una categoria gramatical i altra informació addicional (com poden ser diverses subcategories, el lema associat, etc.) Aquesta és una tasca fonamental en el processament del llenguatge natural, tot i que no està exempta de problemes que no estan encara totalment resolts.

El llenguatge natural és ambigu des de molts punts de vista, i també ho és en el morfosintàctic. Una determinada forma (com per exemple casa) pot tenir diverses interpretacions morfosintàctiques, pot ser un substantiu comú femení singular (amb lema casa) i també una forma de present o d'imperatiu del verb casar. Els etiquetadors morfosintàctics hauran d'intentar donar la interpretació adequada segons el context on apareix una determinada paraula; per tant hauran de desambiguar les diferents possibilitats.

En el mòdul veurem diverses tècniques que ens permetran etiquetar textos des del punt de vista morfosintàctic i que intentaran desambiguar (amb major o menor èxit) les diferents possibilitats.

5.3.2. Etiquetatge morfosintàctic vs. anàlisi morfològica

A l'apartat anterior van estudiar la tasca anomenada anàlisi morfològica, que consisteix a assignar a cada paraula d'un text totes les possibles anàlisis morfològiques. A continuació podem veure l'anàlisi morfològica, duta a terme amb l'analitzador Freeling, de l'oració: Avui fa sol però demà plourà.

Veiem que diverses paraules presenten ambigüitat: fa pot ser tant una forma del verb fer, com un substantiu, la nota musical fa; sol, pot ser tant un adjectiu, com un nom, com diverses formes del verb soler; però pot ser tant una conjunció com un substantiu (Dificultat, objecció. Trobar peròs en tot. [Font: DIEC]); i demà pot ser tant un adverbi com un substantiu.

Un analitzador morfosintàctic ofereix la mateixa sortida, però desambiguada, és a dir, tria una de les possibilitats de cada paraula. Veiem ara l'anàlisi morfosintàctica feta per Freeling de la mateixa oració:

Com podem observar, en general la tria la fa correcta, excepte en el cas de sol que l'etiqueta com a adjectiu.

L'etiquetatge morfosintàctic és una tasca bàsica per a moltes tasques de processament del llenguatge natural. Si volem fer una anàlisi sintàctica d'una oració, un pas previ és conèixer la categoria gramatical de cada paraula. Disposar de textos etiquetats a escala morfosintàctica és interessant per a molts estudis. Podem saber quins són els substantius més utilitzats en un corpus, veure totes les aparicions d'un verb independentment de la forma concreta, etc. L'etiquetatge morfosintàctic també es fa servir per a extreure els termes més rellevants d'un determinat document o conjunt de documents. L'etiquetatge també es fa servir per a la classificació de documents i recuperació d'informació.

5.3.4. Etiquetador per a l'anglès

NLTK proporciona un etiquetador per a l'anglès que funciona prou bé i que es pot fer servir fàcilment de manera directa, com al programa-5-6.py.

```
import nltk

oracio="They refuse to permit us to obtain the refuse permit"

paraules = nltk.tokenize.word_tokenize(oracio)

analisi=nltk.pos_tag(paraules)

print(analisi)
```

que dóna la següent sortida:

```
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Fixeu-vos que la paraula *permit* l'ha etiquetat correctament com a verb i com a substantiu. Fixeu-vos que les etiquetes les dona amb un etiquetari (tagset), o conjunt d'etiquetes, determinat. En aquest cas concret es fan servir el tagset WSJ (Wall Street Journal), que és el següent:

```
Tag   Function

CC   coordinating conjunction

CD   cardinal number

DT   determiner

EX   existential ``there''

FW   foreign word

IN   preposition

JJ   adjective

JJR   adjective, comparative

JJS   adjective, superlative

MD   modal

NN   non-plural common noun

NNP   non-plural proper noun

NNPS   plural proper noun

NNS   plural common noun

of   the word ``of''

PDT   pre-determiner

POS   posessive

PRP   pronoun

puncf   final punctuation (period, question mark and exclamation mark)

punc   other punction

RB   adverb

RBR   adverb, comparative

RBS   adverb, superlative
```

```
RP   particle
TO   the word ``to''
UH   interjection
VB   verb, base form
VBD   verb, past tense
VBG   verb, gerund or present participle
VBN   verb, past participle
VBP   verb, non-3rd person
VBZ   verb, 3rd person
WDT   wh-determiner
WP   wh-pronoun
WRB   wh-adverb
sym   symbol
2   ambiguously labelled
```

Com veurem quan comencem a construir etiquetadors per a altres llengües, com el català, les etiquetes que farem servir seran diferents. Hi ha una proposta d'etiquetari universal (universal tagset). A continuació podem observar aquest etiquetari.

```
Tag   Meaning   English   Examples
ADJ   adjective   new, good, high, special, big, local
ADP   adposition   on, of, at, with, by, into, under
ADV   adverb   really, already, still, early, now
CONJ   conjunction   and, or, but, if, while, although
DET   determiner, article   the, a, some, most, every, no, which
NOUN   noun   year, home, costs, time, Africa
NUM   numeral   twenty-four, fourth, 1991, 14:24
PRT   particle   at, on, out, over per, that, up, with
PRON   pronoun   he, their, her, its, my, I, us
VERB   verb   is, say, told, given, playing, would
.   punctuation marks   . , ; !
X   other   ersatz, esprit, dunno, gr8, univeristy
```

NLTK ens permet convertir les etiquetes d'un determinat etiquetari al l'etiquetari universal. Ho podem veure al programa-5-6b.py.

```
import nltk
oracio="They refuse to permit us to obtain the refuse permit"
paraules = nltk.tokenize.word_tokenize(oracio)
analisi=nltk.pos_tag(paraules)

for ana in analisi:
    forma=ana[0]
```

```
    etiqueta=ana[1]
    universal=nltk.tag.mapping.map_tag('en-ptb', 'universal', etiqueta)
    print(forma,etiqueta,universal)
```

que ens dona la següent sortida:

```
They PRP PRON
refuse VBP VERB
to TO PRT
permit VB VERB
us PRP PRON
to TO PRT
obtain VB VERB
the DT DET
refuse NN NOUN
permit NN NOUN
```

5.4.1. L'etiquetador per defecte

En aquest apartat presentem un etiquetador molt simple, que l'únic que fa és etiquetar totes les paraules amb una etiqueta determinada, és a dir, etiqueta totes les paraules amb la mateixa etiqueta. Per determinar quina etiqueta triarem el que farem primer és calcular quina és l'etiqueta més freqüent. Per a aconseguir això farem ús de corpus ja etiquetats: per a l'anglès farem servir el Brown Corpus i per al català el CESS_CAT.

Per a l'anglès:

En un intèrpret interactiu fem, per calcular l'etiqueta més freqüent:

```
>>> import nltk
>>> from nltk.corpus import brown
>>> tags=[tag for (word,tag) in brown.tagged_words()]
>>> nltk.FreqDist(tags).max()
'NN'
```

I per definir un etiquetador per defecte fem:

```
>>> from nltk.tokenize import word_tokenize
>>> tokens=word_tokenize(oracio)
>>> from nltk.tokenize import word_tokenize
>>> oracio="they refuse to permit us to obtain the refuse permit"
>>> tokens=word_tokenize(oracio)
>>> default_tagger=nltk.DefaultTagger('NN')
>>> default_tagger.tag(tokens)
```

[('they', 'NN'), ('refuse', 'NN'), ('to', 'NN'), ('permit', 'NN'), ('us', 'NN'), ('to', 'NN'), ('obtain', 'NN'), ('the', 'NN'), ('refuse', 'NN'), ('permit', 'NN')]

Com podeu observar, etiqueta totes les paraules amb 'NN'.

Per al català:

```
>>> from nltk.corpus import cess_cat
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> tags=[tag for (word,tag) in cess_cat.tagged_words()]
>>> nltk.FreqDist(tags).max()
'sps00'
```

Ens pot sorprendre que l'etiqueta més freqüent en català sigui la corresponent a la preposició ja que esperaríem que fos també la corresponent a substantiu. El que passa és que en l'etiquetari català tenin posicions per la categoria i diverses subcategoritzacions (gènere i nombre). Si modifiquem les dues darreres línies per fer que miri només la primera posició de l'etiqueta, obtindrem que la categoria més freqüent és 'n' (substantiu).

```
>>> tags=[tag[0] for (word,tag) in cess_cat.tagged_words()]
>>> nltk.FreqDist(tags).max()
'n'
```

Podríem fer també un etiquetador per defecte del català indicant que l'etiqueta per defecte fons 'n'. Però realment preferiríem indicar l'etiqueta completa més freqüent per als substantius.

```
>>> tags=[tag for (word,tag) in cess_cat.tagged_words()]
>>> nltk.FreqDist(tags).most_common(10)
[('sps00', 64225), ('ncfs000', 30917), ('ncms000', 29527), ('Fc', 26964), ('da0fs0', 17322), ('np00000', 17237), ('Fp', 16875), ('vmn0000', 15715), ('ncmp000', 14797), ('cc', 14403)]
```

Que ens dona que els noms comuns femenins singulars són lleugerament més freqüents que els noms comuns masculins singulars. Ara podem definir un etiquetador per defecte per al català:

```
>>> oracio="Avui fa sol però demà plourà"
>>> tokens=word_tokenize(oracio)
>>> default_tagger=nltk.DefaultTagger('ncfs000')
>>> default_tagger.tag(tokens)
[('Avui', 'ncfs000'), ('fa', 'ncfs000'), ('sol', 'ncfs000'), ('però', 'ncfs000'), ('demà', 'ncfs000'), ('plourà', 'ncfs000')]
```

Ja ens podem imaginar que aquest etiquetador no funcionarà gaire bé. En el següent apartat aprendrem a avaluar etiquetadors i podrem veure la precisió d'aquest etiquetador. També pot servir pels casos que tinguem una paraula desconeguda ja que li podrem assignar l'etiqueta més freqüent (sense comptar la de preposició, ja que sent una categoria tancada és pràcticament impossible que sigui desconeguda).

5.4.2. L'etiquetador per unigrames

L'etiquetador per defecte estudiat en l'apartat anterior etiqueta totes les paraules amb l'etiqueta més freqüent en tot el corpus. En aquest apartat i els propers estudiarem una sèrie d'etiquetadors anomenats genèricament etiquetadors per n-grames. Un n-grama és una combinació d'n elements. En general aquests etiquetadors aprenen a partir del corpus tenint en compte un context de n paraules. En el cas de l'etiquetador per unigrames l'únic que tenim en compte és la paraula per etiquetar mateixa, sense cap context. L'etiquetador per unigrames etiquetarà cada paraula amb l'etiqueta més frequent per a aquella paraula.

Amb l'NLTK crear un etiquetador per unigrames és molt senzill. Primer ho farem per a l'anglès i després per al català.

Per a l'anglès

```
>>> import nltk
>>> from nltk.corpus import brown
>>> tagged_sents=brown.tagged_sents()
>>> unigram_tagger=nltk.UnigramTagger(tagged_sents)
>>> oracio="they refuse to permit us to obtain the refuse permit"
>>> tokens=nltk.word_tokenize(oracio)
>>> unigram_tagger.tag(tokens)
```

[('they', 'PPSS'), ('refuse', 'VB'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PPO'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'AT'), ('refuse', 'VB'), ('permit', 'VB')]

Si ens fixem en el resultat, veiem que *refuse* l'etiqueta les dues vegades que apareix com a verb, ja que aquesta és l'etiqueta més freqüent per a aquesta paraula.

Per al català

Farem el mateix, però aquesta vegada en un programa (programa-5-7.py):

```
import nltk
from nltk.corpus import cess_cat
from nltk.tokenize import word_tokenize
tagged_sents=cess_cat.tagged_sents()
unigram_tagger=nltk.UnigramTagger(tagged_sents)
oracio="avui fa sol però demà plourà"
tokens=word_tokenize(oracio)
analisi=unigram_tagger.tag(tokens)
print(analisi)
```

que ens dona la sortida:

[('avui', 'rg'), ('fa', 'vmip3s0'), ('sol', 'aq0ms0'), ('però', 'cc'), ('demà', 'rg'), ('plourà', None)]

També podem entrenar un etiquetador amb un corpus propi. Ara farem servir el Wikicorpus, que és un corpus format per textos de la Vikipèdia i etiquetats amb Freeling. Com que aquest corpus ja té uns anys, descarregarem

els textos i el tornarem a etiquetar amb una versió més nova de Freeling. Podeu descarregar aquest corpus dels fitxers d'aquest capítol (recordeu descomprimir-lo un cop descarregat). Observem el format del corpus:

```
S' es P0300000 0.999814
hi hi PP3CN000 1
poden poder VMIP3P0 0.470339
representar representar VMN0000 1
nombres nombre NCMP000 0.454513
enters enter AQ0MP0 0.366457
o o CC 0.999266
decimals decimal AQ0CP0 0.566499
. . Fp 1
```

És a dir, forma, lema, etiqueta i probabilitat de l'etiqueta. Haurem de crear un codi capaç de llegir aquest corpus i crear les tagged_sents com a llistes de tagged_words on cada tagged_word és una tupla forma, etiqueta. Ho podem veure al programa-5-8.py:

```
import codecs
import nltk

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_words=[]
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

unigram_tagger=nltk.UnigramTagger(tagged_sents)
oracio="avui fa sol però demà plourà"
```

```
tokens=nltk.tokenize.word_tokenize(oracio)
analisi=unigram_tagger.tag(tokens)
print(analisi)
```

i proporciona la següent sortida:

```
[('avui', 'RG'), ('fa', 'VMIP3S0'), ('sol', 'NCMS000'), ('però', 'CC'), ('demà', 'RG'), ('plourà', 'VMIF3S0')]
```

5.4.3. L'etiquetador per bigrames

L'etiquetador per unigrames no té en compte el context d'aparició de les paraules, i les etiqueta sempre amb l'etiqueta més freqüent per a aquesta paraula. En aquest apartat construirem un etiquetador per bigrames que té en compte la pròpia paraula a etiquetar i la paraula anterior:

Per a l'anglès

```
>>> import nltk
>>> from nltk.corpus import brown
>>> tagged_sents=brown.tagged_sents()
>>> bigram_tagger=nltk.BigramTagger(tagged_sents)
>>> oracio="they refuse to permit us to obtain the refuse permit"
>>> tokens=word_tokenize(oracio)
>>> tokens=nltk.tokenize.word_tokenize(oracio)
>>> bigram_tagger.tag(tokens)
>>> bigram_tagger.tag(tokens)
[('they', 'PPSS'), ('refuse', 'VB'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PPO'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'AT'), ('refuse', 'NN'), ('permit', 'NN')]
```

Fixem-nos que ara pot etiquetar el segon refuse com a nom, ja que té en compte el context immediat.

Per al català

Modifiquem el programa-5-8.py per obtenir el programa-5-9.py:

```
import codecs
import nltk

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
```

```
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_words=[]
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

bigram_tagger=nltk.BigramTagger(tagged_sents)
oracio="avui fa sol però demà plourà"
tokens=nltk.tokenize.word_tokenize(oracio)
analisi=bigram_tagger.tag(tokens)
print(analisi)
```

Que ofereix ara la següent sortida:

```
[('avui', None), ('fa', None), ('sol', None), ('però', None), ('demà', None), ('plourà', None)]
```

És a dir, que no ha pogut etiquetar cap paraula. Això és degut a un fenomen conegut com a dispersió de dades. Hi ha molts més unigrames que bigrames en un corpus. Si entrenem un etiquetador amb bigrames, necessitarem un corpus més gran per poder trobar suficients evidències de cada bigrama de l'oracio a analitzar, com que no sempre és possible disposar de grans corpus, es pot recórrer a la tècnica coneguda com a *backoff*. A aquesta tècnica també se la coneix com a combinació d'etiquetadors. Al programa-5-10.py combinem un etiquetador de bigrames amb un d'unigrames.

```
import codecs
import nltk

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_words=[]
```

```
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

unigram_tagger=nltk.UnigramTagger(tagged_sents)
bigram_tagger=nltk.BigramTagger(tagged_sents,backoff=unigram_tagger)
oracio="avui fa sol però demà plourà"
tokens=nltk.tokenize.word_tokenize(oracio)
analisi=bigram_tagger.tag(tokens)
print(analisi)
```

i ens ofereix ara la següent sortida:

```
[('avui', 'RG'), ('fa', 'VMIP3S0'), ('sol', 'AQ0MS0'), ('però', 'CC'), ('demà', 'RG'), ('plourà', 'VMIF3S0')]
```

i com veiem, ha pogut etiquetar totes les paraules, ja que amb l'etiquetador de bigrames no ha pogut, però sí fent servir el d'unigrames.

5.4.4. L'etiquetador per trigrames

L'etiquetador per trigrames etiqueta una paraula tenint en compte el context format per les dues paraules anteriors. En aquest cas, el problema de la dispersió de dades serà encara més pronunciat. En el programa-5-11.py implementem un etiquetador per trigrames que es combina amb un de bigrames i a la seva vegada per un de trigrames. Fixeu-vos que en aquest programa hem canviat la frase a analitzar i hem fet servir un tokenitzador per expressions regular que ens permet tractar satisfactòriament els apòstrofs:

```
import codecs
import nltk

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_words=[]
    else:
```

```
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

unigram_tagger=nltk.UnigramTagger(tagged_sents)
bigram_tagger=nltk.BigramTagger(tagged_sents,backoff=unigram_tagger)
trigram_tagger=nltk.TrigramTagger(tagged_sents,backoff=bigram_tagger)
oracio="l'àcid desoxiribonucleic (ADN o DNA) és un àcid nucleic que conté les instruccions genètiques utilitzades en el desenvolupament i funcionament de tots els éssers vius coneguts, així com en alguns virus, des d'un punt de vista químic, l'ADN es compon de dos llargs polímers d'unitats simples anomenades nucleòtids, amb un tronc compost de sucres i grups fosfats units per enllaços èster"
tokenitzador=nltk.tokenize.RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')
tokens=tokenitzador.tokenize(oracio)
analisi=trigram_tagger.tag(tokens)
print(analisi)
```

que proporciona la següent anàlisi:

```
[("l'", 'DA0CS0'), ('àcid', 'NCMS000'), ('desoxiribonucleic', 'AQ0MS0'), ('(', 'Fpa'), ('ADN', 'NP00000'), ('o', 'CC'), ('DNA', 'NP00000'), (')', 'Fpt'), ('és', 'VSIP3S0'), ('un', 'DI0MS0'), ('àcid', 'NCMS000'), ('nucleic', 'AQ0MS0'), ('que', 'PR0CN000'), ('conté', 'VMIP3S0'), ('les', 'DA0FP0'), ('instruccions', 'NCFP000'), ('genètiques', 'AQ0FP0'), ('utilitzades', 'VMP00PF'), ('en', 'SPS00'), ('el', 'DA0MS0'), ('desenvolupament', 'NCMS000'), ('i', 'CC'), ('funcionament', 'NCMS000'), ('de', 'SPS00'), ('tots', 'DI0MP0'), ('els', 'DA0MP0'), ('éssers', 'NCMP000'), ('vius', 'AQ0MP0'), ('coneguts', 'NCMP000'), (',', 'Fc'), ('així', 'RG'), ('com', 'CS'), ('en', 'SPS00'), ('alguns', 'DI0MP0'), ('virus', 'NCMN000'), (',', 'Fc'), ('des', 'SPS00'), ("d'", 'SPS00'), ('un', 'DI0MS0'), ('punt', 'NCMS000'), ('de', 'SPS00'), ('vista', 'NCCS000'), ('químic', 'AQ0MS0'), (',', 'Fc'), ("l'", 'DA0CS0'), ('ADN', 'NP00000'), ('es', 'P0300000'), ('compon', 'VMIP3S0'), ('de', 'SPS00'), ('dos', 'Z'), ('llargs', 'AQ0MP0'), ('polímers', 'NCMP000'), ("d'", 'SPS00'), ('unitats', 'NCFP000'), ('simples', 'AQ0CP0'), ('anomenades', 'NCFP000'), ('nucleòtids', 'NCMP000'), (',', 'Fc'), ('amb', 'SPS00'), ('un', 'DI0MS0'), ('tronc', 'NCMS000'), ('compost', 'NCMS000'), ('de', 'SPS00'), ('sucres', 'NCMP000'), ('i', 'CC'), ('grups', 'NCMP000'), ('fosfats', 'NCMP000'), ('units', 'VMP00PM'), ('per', 'SPS00'), ('enllaços', 'NCMP000'), ('èster', None)]
```

On si ens fixem, hi ha algunes paraules, com èster que no les pot etiqueta, tot i que hi ha paraules força especialitzades que sí que és capaç d'etiquetar.

5.4.5. Tractament de paraules desconegudes: diccionari morfològic, etiquetador per afixos i etiquetador per defecte

En el programa anterior hem vist que un etiquetador entrenat amb un corpus no serà capaç d'etiquetar paraules que no apareguin en el corpus. Per solucionar això farem servir 3 estratègies:

- entrenar un etiquetador per unigrames a partir d'un diccionari morfològic, que en aquest cas serà el del propi Freeling i que podeu descarregar d'aquest apartat.
- entrenar un etiquetador per afixos que el que fa és fer servir totes les paraules del diccionari morfològic (i en el seu defecte podria fer-se servir les pròpies paraules del corpus), per aprendre les etiquetes més freqüents segons les terminacions de les paraules.

- i si tot això falla, fer servir un etiquetador per defecte, que faci servir l'etiqueta més freqüent en el nostre corpus d'aprenentatge

Veiem primer aquests tres entrenaments per separat i després ho combinarem tot en un únic etiquetador.

Etiquetador per unigrames a partir d'un diccionari morfològic

Farem servir aquesta part de codi (programa-5-12.py):

```
import codecs
import nltk

entrada=codecs.open("diccionari-cat.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
cont=0
for linia in entrada:
    cont+=1
    if cont==10000:
        break
    linia=linia.rstrip()
    camps=linia.split(" ")
    forma=camps[0]
    lema=camps[1]
    etiqueta=camps[2]
    tupla=(forma,etiqueta)
    tagged_words.append(tupla)
tagged_sents.append(tagged_words)

unigram_tagger_diccionari=nltk.UnigramTagger(tagged_sents)
```

Fixeu-vos que contem les línies i quan arriba a la 10000 parem l'entrenament per a que no trigui massa. Per fer les proves serà suficient i en el moment de fer l'entrenament definitiu eliminarem (comentarem) aquestes línies de codi de manera que faci l'entrenament amb tot el diccionari. Aquest programa no ofereix cap sortida.

Entrenament d'un etiquetador per afixos utilitzant el diccionari morfològic

Podem trobar la implementació en el programa-5-13.py que és exactament igual que l'anterior però canvia la darrera línia i ara és:

```
affix_tagger=nltk.AffixTagger(tagged_sents, affix_length=-3, min_stem_length=2)
```

que permet entrenar l'etiquetador per afixos tenint en compte els afixos amb tres caràcters sempre i quan l'arrel que quedi tingui 2 o més caràcters.

Determinació de l'etiqueta més freqüent i entrenament de l'etiquetador per defecte

El programa-5-14.py ens retorna les 10 etiquetes més freqüents:

```
import codecs
import nltk

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_words=[]
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)
tags=[]
for ts in tagged_sents:
    for wt in ts:
        tags.append(wt[1])

mft=nltk.FreqDist(tags).most_common(10)
print("Etiqueta més freqüent: ",mft)

default_tagger=nltk.DefaultTagger("NP00000")
```

Que ens dona a la sortida:

```
Etiquetes més freqüent:  [('SPS00', 284406), ('NP00000', 177296), ('NCMS000', 140239), ('NCFS000', 130773), ('Fc', 113652), ('Fp', 96173), ('DA0MS0', 87666), ('DA0FS0', 69641), ('CC', 64777), ('NCMP000', 62556)]
```

I tornem a tenir que la més freqüent (de les categories obertes) és el substantiu, concretament el nom propi. I llavors creem l'etiquetador per defecte fent servir aquesta etiqueta.

Posarem tot això junt en el següent apartat, o a més, aprendrem a emmagatzemar etiquetadors .

5.5. Emmagatzematge d'etiquetadors

En l'apartat anterior hem après a entrenar i combinar etiquetadors. Cada cop que volíem etiquetar una oració, entrenàvem un etiquetador i després etiquetàvem. Com que l'entrenament és lent, ens interessa entrenar una vegada i poder guardar l'etiquetador ja entrenat de manera que el puguem fer servir tantes vegades com vulguem.

En el programa-5-15.py entrenen i emmagatzem un etiquetador. Fixeu-vos en tots els etiquetadors diferents que entrenem, com els combinem, i com finalment emmagatzem el darrer, que de fet està combinat amb tota la resta. Fixeu-vos també com fem servir el mòdul pickle. I també tingueu en compte que hem comentat les línies que limiten el nombre d'entrades del diccionari morfològic que fa servir per entrenar. Si veieu que triga molt a entrenar, torneu a descomentar aquestes línies:

```
import codecs
import nltk
import pickle

entrada=codecs.open("catalanTagged_0_5000-utf-8.txt","r",encoding="utf-8")

tagged_words=[]
tagged_sents=[]
tagged_sents_per_unigrams=[]
cont=0
for linia in entrada:
    #cont+=1
    #if cont==10000:
    #    break
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
        if len(tagged_words)>0:
            tagged_sents.append(tagged_words)
            tagged_sents_per_unigrams.append(tagged_words)
            tagged_words=[]
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

if len(tagged_words)>0:
```

```
        tagged_sents.append(tagged_words)
        tagged_sents_per_unigrams.append(tagged_words)
        tagged_words=[]

    diccionari=codecs.open("diccionari-cat.txt","r",encoding="utf-8")

    cont=0
    for linia in diccionari:
        #cont+=1
        #if cont==10000:
        #    break
        linia=linia.rstrip()
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)
    tagged_sents_per_unigrams.append(tagged_words)

    default_tagger=nltk.DefaultTagger("NP00000")
    affix_tagger=nltk.AffixTagger(tagged_sents_per_unigrams, affix_length=-3,
min_stem_length=2,backoff=default_tagger)
    unigram_tagger_diccionari=nltk.UnigramTagger(tagged_sents_per_unigrams,backoff=affix_tagger)
    unigram_tagger=nltk.UnigramTagger(tagged_sents,backoff=unigram_tagger_diccionari)
    bigram_tagger=nltk.BigramTagger(tagged_sents,backoff=unigram_tagger)
    trigram_tagger=nltk.TrigramTagger(tagged_sents,backoff=bigram_tagger)

    sortida=open('etiquetador-cat.pkl', 'wb')
    pickle.dump(trigram_tagger, sortida, -1)
    sortida.close()
```

El que és important tenir en compte en el programa anterior és que estem proporcionant un corpus i un diccionari. El corpus ens proporciona informació de forma i etiqueta dins del context de l'oració. En canvi, el diccionari només ens dona informació sobre formes i les seves etiquetes d'una manera totalment fora de context, ja que la paraula que apareix al diccionari abans d'una altra només guarda una relació alfabètica. Per aquest motiu, en tots els entrenaments que suposin un context (bigrams i trigrams) només podem fer servir la informació que prové del corpus. En canvi, per als entrenaments que no es tingui en compte el context (afixos i unigrames) podem fer servir tant la informació que apareix en el corpus com la que apareix al diccionari. Fixeu-vos que al programa fem servir dues llistes per emmagatzemar la informació que fem servir per entrenar:

- tagged_sents_per_unigrams: on posem la informació del corpus i del diccionari
- tagged_sents: on només posem la informació del corpus

Les línies:

```
cont=0

for linia in diccionari:

    #cont+=1

    #if cont==10000:

    #    break
```

serveixen per limitar la informació que es carrega o bé del corpus o bé del diccionari, o de tots dos. Com que el programa triga molt a executar-se, podeu descomentar (treure el símbol "#") del davant de les línies.

En els arxius d'aquest capítol trobareu l'etiquetador-cat.pkl resultant per a que el pugeu fer servir en el següent programa sense esperar que es completi l'entrenament.

Ara a etiquetador-cat.pkl tenim un etiquetador que podem carregar sempre que vulguem de manera molt ràpida. Fixeu-vos en el programa-5-16:

```
import nltk

import pickle
import nltk

entrada=open('etiquetador-cat.pkl','rb')
etiquetador=pickle.load(entrada)
entrada.close()

oracio="l'àcid desoxiribonucleic (ADN o DNA) és un àcid nucleic que conté les instruccions genètiques utilitzades en el desenvolupament i funcionament de tots els éssers vius coneguts, així com en alguns virus, des d'un punt de vista químic, l'ADN es compon de dos llargs polímers d'unitats simples anomenades nucleòtids, amb un tronc compost de sucres i grups fosfats units per enllaços èster"
tokenitzador=nltk.tokenize.RegexpTokenizer('[ldsmLDSM]\'|\w+|[^\w\s]+')
tokens=tokenitzador.tokenize(oracio)
analisi=etiquetador.tag(tokens)
print(analisi)
```

Que ofereix l'anàlisi:

```
[("l'", 'DA0CS0'), ('àcid', 'NCMS000'), ('desoxiribonucleic', 'AQ0MS0'), ('(', 'Fpa'), ('ADN', 'NP00000'), ('o', 'CC'), ('DNA', 'NP00000'), (')', 'Fpt'), ('és', 'VSIP3S0'), ('un', 'DI0MS0'), ('àcid', 'NCMS000'), ('nucleic', 'AQ0MS0'), ('que', 'PR0CN000'), ('conté', 'VMIP3S0'), ('les', 'DA0FP0'), ('instruccions', 'NCFP000'), ('genètiques', 'AQ0FP0'), ('utilitzades', 'VMP00PF'), ('en', 'SPS00'), ('el', 'DA0MS0'), ('desenvolupament', 'NCMS000'), ('i', 'CC'), ('funcionament', 'NCMS000'), ('de', 'SPS00'), ('tots', 'DI0MP0'), ('els', 'DA0MP0'), ('éssers', 'NCMP000'), ('vius', 'AQ0MP0'), ('coneguts', 'NCMP000'), (',', 'Fc'), ('així', 'RG'), ('com', 'CS'), ('en', 'SPS00'), ('alguns', 'DI0MP0'), ('virus', 'NCMN000'), (',', 'Fc'), ('des', 'SPS00'), ("d'", 'SPS00'), ('un', 'DI0MS0'), ('punt', 'NCMS000'), ('de', 'SPS00'), ('vista', 'NCCS000'), ('químic', 'AQ0MS0'), (',', 'Fc'), ("l'", 'DA0CS0'), ('ADN', 'NP00000'), ('es', 'P0300000'), ('compon', 'VMIP3S0'), ('de', 'SPS00'), ('dos', 'Z'), ('llargs', 'AQ0MP0'), ('polímers', 'NCMP000'), ("d'", 'SPS00'), ('unitats', 'NCFP000'), ('simples', 'AQ0CP0'), ('anomenades', 'NCFP000'), ('nucleòtids', 'NCMP000'), (',', 'Fc'), ('amb', 'SPS00'), ('un', 'DI0MS0'), ('tronc', 'NCMS000'), ('compost', 'NCMS000'), ('de', 'SPS00'), ('sucres', 'NCMP000'), ('i', 'CC'), ('grups', 'NCMP000'), ('fosfats', 'NCMP000'), ('units', 'VMP00PM'), ('per', 'SPS00'), ('enllaços', 'NCMP000'), ('èster', 'NCMS000')]
```

5.6. Avaluació d'etiquetadors

En els apartats anteriors he après a crear etiquetadors estadístics. També hem après a fer-los servir per etiquetar textos i intuïtivament hem vist que funcionen força bé, tot i que poden etiquetar malament algunes paraules. Quan desenvolupem un etiquetador, ens interessaria saber quina precisió assolim. NLTK ens proporciona una manera molt senzilla d'avaluar etiquetadors. Comencem avaluant un etiquetador per a l'anglès (programa-5-17.py)

```
import nltk
from nltk.corpus import brown
brown_tagged_sents=brown.tagged_sents()
print("TOTAL ORACIONS:", len(brown_tagged_sents))
train_sents=brown_tagged_sents[:10000]
test_sents=brown_tagged_sents[56001:]
default_tagger=nltk.DefaultTagger("NN")
affix_tagger=nltk.AffixTagger(train_sents, affix_length=-3, min_stem_length=2)
unigram_tagger=nltk.UnigramTagger(train_sents, backoff=affix_tagger)
bigram_tagger=nltk.BigramTagger(train_sents, backoff=unigram_tagger)
trigram_tagger=nltk.TrigramTagger(train_sents, backoff=bigram_tagger)
precisio=trigram_tagger.evaluate(test_sents)
print("PRECISIO: ",precisio)
```

El programa primer ens indicarà el nombre total d'oracions del corpus. Fixeu-vos que fa un conjunt d'oracions d'entrenament (train_sents) amb les primeres 10000 oracions del corpus; i un de test amb les oracions finals, de la 56001 fins la final. Després de crear l'etiquetador amb el mètode evaluate avalua la precisió fent servir les oracions de test. El que fa el programa és etiquetar aquestes oracions i comparar-les amb les etiquetes reals. Si executeu el programa, obtindreu la següent sortida:

```
TOTAL ORACIONS: 57340
PRECISIO:  0.8937947951400866
```

Canvieu ara el nombre d'oracions per entrenar l'etiquetador i poseu el màxim (sense agafar oracions de test), és a dir, 56000. Si executeu ara el programa, la precisió passa a ser de:

```
PRECISIO:  0.9220420834770611
```

Veiem que quan més gran sigui el corpus d'entrenament, obtindrem una millor precisió.

Anem a avaluar l'etiquetador del català que hem entrenat i emmagatzemat en l'apartat anterior. Ho fem en el programa-5-18.py.

```
import nltk
import pickle
import codecs

#carreguem l'etiquetador
entrada=open('etiquetador-cat.pkl','rb')
etiquetador=pickle.load(entrada)
entrada.close()

#carreguem les oracions del corpus de test

entrada=codecs.open("wikicorpus-tagged-test.txt","r",encoding="utf-8")
```

```
tagged_words=[]
test_tagged_sents=[]
for linia in entrada:
    linia=linia.rstrip()
    if linia.startswith("<") or len(linia)==0:
        #nova linia
        if len(tagged_words)>0:
            test_tagged_sents.append(tagged_words)
            tagged_words=[]
    else:
        camps=linia.split(" ")
        forma=camps[0]
        lema=camps[1]
        etiqueta=camps[2]
        tupla=(forma,etiqueta)
        tagged_words.append(tupla)

precisio=etiquetador.evaluate(test_tagged_sents)
print("PRECISIO: ",precisio)
```

Si ens fixem, fem servir un fragment nou del wikicorpus ja etiquetat (wikicorpus-tagged-test.txt). Primer carreguem l'etiquetador emmagatzemat i després carreguem el nou corpus afegint-lo al test_tagged_sents, que són les que farem servir per avaluar al mètode evaluate. Aquest programa ens proporciona la següent sortida:

```
PRECISIO:  0.9585872792623847
```

Recordem que el corpus que estem fent servir és un corpus etiquetat automàticament amb Freeling, i per tant, el 95.8% de precisió que obtenim no és real. Hauríem de fer servir corpus etiquetats manualment, o bé etiquetats automàticament i revisats.