Etiquetatge morfosintàctic

Antoni Oliver

PID_00155228

Universitat Oberta de Catalunya
www.uoc.edu

Índex

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 d'utilització; 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.

Objectius

Els objectius bàsics que ha d'haver aconseguit l'estudiant una vegada treballats els continguts d'aquest mòdul són els següents:

1. Saber què és i per a què serveix un etiquetador morfosintàctic.
2. Conèixer els principals problemes que presenta la tasca de l'etiquetatge morfosintàctic i les principals tècniques per a solucionar-los.
3. Saber programar etiquetadors morfosintàctics senzills amb Python i NLTK.
4. Comprendre els conceptes d'entrenament i avaluació d'etiquetadors.
5. Conèixer alguns dels etiquetadors disponibles.

1. Els etiquetadors morfosintàctics

Un **etiquetador morfosintàctic** és una aplicació informàtica que assigna a cada paraula d'un text la seva categoria gramatical i altra informació addicional. Aquesta informació addicional pot consistir en certes subcategoritzacions i també en el lema associat a la paraula.

A continuació podem veure la sortida d'un etiquetador morfosintàctic que analitza la frase "Avui fa sol però demà plourà". La primera sortida que oferim a la figura 1 és sense desambiguar. Aquesta anàlisi sovint es coneix amb el nom d'*anàlisi morfològica* i dóna totes les possibles interpretacions de cada forma.

En el mòdul "Eines i recursos per al català i castellà" estudiarem més a fons l'analitzador Freeling i veurem que és molt més que un analitzador morfosintàctic.

Figura 1. Análisi morfològica

Avui	**fa**	**sol**	**però**	**demà**	**plourà**	**.**
avui RG 1	*fer* VMIP3S0 0.992188	*sol* NCMS000 0.6875	*però* CC 0.890152	*demà* RG 0.988095	*ploure* VMIF3S0 1	. Fp 1
	fa NCMS000 0.0078125	*sol* AQ0MS0 0.270833	*però* RG 0.106061	*demà* NCMS000 0.0119048		
		soler VMIP3S0 0.0208333	*però* NCMS000 0.00378788			
		soler VMM02S0 0.0208333				

A la figura 2 podeu veure la mateixa sortida desambiguada. Aquest tipus de sortida normalment rep el nom d'*anàlisi morfosintàctica*.

Tagset o etiquetari

Hi ha diferents etiquetaris o *tagsets*. Per a interpretar bé la sortida d'un etiquetador haurem de conèixer *a priori* l'etiquetari que fa servir.

Figura 2. Análisi morfosintàctica

Avui	**fa**	**sol**	**però**	**demà**	**plourà**	**.**
avui	*fer*	*sol*	*però*	*demà*	*ploure*	.
RG	VMIP3S0	AQ0MS0	CC	RG	VMIF3S0	Fp

Aquestes dues anàlisis s'han fet amb l'analitzador Freeling. Podeu accedir a una *demo* d'aquest analitzador a *http://garraf.epsevg.upc.es/freeling/demo.php*. Les etiquetes s'han expressat amb l'etiquetari (*tagset*) PAROLE (Eagles) que podeu trobar detallat a l'adreça *http://garraf.epsevg.upc.es/freeling/doc/userman/parole-ca.html*.

1.1. Per a què serveix l'etiquetatge morfosintàctic?

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.

L'etiquetatge morfosintàctic és, doncs, una tasca bàsica molt important i per aquest motiu l'estudiarem a fons en aquest mòdul.

1.2. Un primer etiquetador morfosintàctic en NLTK

En aquest subapartat presentem un primer etiquetador morfosintàctic per a l'anglès desenvolupat en NLTK i que es pot cridar de manera directa. En les línies següents podem veure com es pot cridar un etiquetador de l'anglès de l'NLTK en una sessió interactiva de Python.

En la resta del mòdul veurem com podem construir els nostres propis etiquetadors morfosintàctis.

```
>>> import nltk
>>> text=nltk.word_tokenize("They refuse to permit us to obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit','NN')]
```

2. Tècniques per a l'etiquetatge morfosintàctic

L'anàlisi morfosintàctica d'un text presenta diverses dificultats, entre les quals es poden destacar l'ambigüitat morfosintàctica i el tractament de paraules desconegudes. En aquest apartat considerem que podem resoldre satisfactòriament les tasques de *tokenització* i segmentació en oracions.

En el mòdul "Anàlisi textual i processament de corpus" hem estudiat la tasca de segmentació en unitats lèxiques (*tokenització*) i segmentació en oracions. Aquestes dues tasques són bàsiques i necessàries per a poder fer l'anàlisi morfosintàctica.

Com ja hem comentat, moltes paraules poden tenir diverses etiquetes associades. El nostre etiquetador morfosintàctic haurà de ser capaç de determinar quina etiqueta de les possibles és la que correspon.

Exemple

En la frase castellana "Yo bajo con el hombre bajo a tocar el bajo bajo la escalera" (frase extreta de la *demo* de l'analitzador Freeling), la forma *bajo* pot ser preposició, adjectiu, substantiu i verb. En aquesta frase hi ha altres formes ambigües: *hombre* (que pot ser substantiu i interjecció), *a* (que pot ser preposició i substantiu, el nom de la primera lletra de l'abecedari) i *la* (que pot ser determinant, pronom i substantiu, la nota musical).

Com veurem més endavant, els etiquetadors d'una manera o d'una altra treballen amb un diccionari de formes. Aquest diccionari conté *totes* les formes de la llengua amb les seves possibles etiquetes i sovint també el lema associat. Com sabeu, és impossible recollir absolutament totes les formes d'una llengua, i per aquest motiu el nostre etiquetador haurà de ser capaç de tractar les formes desconegudes. A banda de les paraules desconegudes, l'analitzador haurà de ser capaç de tractar noms propis no recollits en el diccionari i algunes paraules estrangeres que puguin sorgir esporàdicament dins del text.

Les tècniques per a solucionar aquests problemes es poden dividir en tres grans grups d'acord amb el tipus de coneixement que fan servir (Màrquez i Rodríguez, 1998): coneixement lingüístic, informació estadística o tècniques d'aprenentatge automàtic. També es poden trobar sistemes híbrids i també hi ha certs sistemes difícils de classificar.

En aquest mòdul presentarem diversos etiquetadors seguint una classificació menys estricta. Començarem presentant un analitzador morfològic senzill que funciona mitjançant consultes a un diccionari. Després veurem algunes tècniques de desambiguació que funcionen mitjançant regles escrites manualment. Després presentarem diverses tècniques que aprenen a desambiguar a partir de corpus anotats. En primer lloc tècniques estadístiques que funcionen a partir d'*n*-grames. A continuació presentarem un cas una mica especial, l'etiquetador de Brill (1995), ja que fa servir un conjunt de regles que es deriven automàti-

cament a partir d'un corpus anotat. Finalment presentarem algunes tècniques d'aprenentatge automàtic que es fan servir per a l'anàlisi morfosintàctica.

Durant el mòdul aprendrem a entrenar i fer servir etiquetadors amb l'NLTK. A més, aprendrem a avaluar els etiquetadors, és a dir, a determinar la precisió que obtenim per a diferents paràmetres.

3. Un analitzador morfològic basat en un diccionari

La primera aplicació que programarem serà un analitzador morfològic senzill que a partir d'un diccionari morfològic emmagatzemat en un fitxer de text serà capaç de fer anàlisis morfològiques de textos d'entrada. Un diccionari morfològic és un diccionari de formes amb els seus possibles lemes i etiquetes relacionats. A continuació podem veure un fragment d'un diccionari morfològic per al català:

```
remis remar VMSP2S0
remisa remís AQ0FS0
remisament remisament RG
remises remís AQ0FP0
remisos remís AQ0MP0
remissibilitat remissibilitat NCFS000
remissibilitats remissibilitat NCFP000
remissible remissible AQ0CS0
```

A partir d'un diccionari d'aquest estil podem fer un analitzador morfològic. Un possible codi que farà aquesta tasca (`programa-5-1.py`) és el que mostrem a continuació.

```
# -*- coding: utf-8 -*-
import nltk
fdiccionari=open("diccionari-cat.txt","r")
diccionari=dict()
while True:
    linia=fdiccionari.readline().rstrip()
    if not linia: break
    camps=linia.split(" ")
    forma=camps.pop(0)
    informacio=" ".join(camps)
    diccionari[forma]=informacio
fdiccionari.close()
frase="avui la meva dona fa vacances, però demà treballarà tot el dia."
expressioregular=r'''(?x)
[\wàèòéíóúïüçÀÈÒÉÍÓÚÏÜÇ]+'?|
[\,\;\.\:\!\?\(\)]
'''
tokens=nltk.regexp_tokenize(frase,expressioregular)
for token in tokens:
    print token+" "+diccionari[token]
```

Aquest programa treu per pantalla una sortida com aquesta:

```
avui avui RG
la el DA0FS0 ell PP3FSA00 la I la NCMS000
meva meu DP1FSS
dona dona I dona NCFS000
fa fa NCMS000 fer VMIP3S0
vacances vacança NCFP000
, , Fc
però però CC però NCMS000
demà demà NCMS000 demà RG
treballarà treballar VMIF3S0
tot tot DI0CS0 tot DI0MS0 tot NCMS000 tot PI0MS000 tot RG
el el DA0MS0 ell PP3MSA00
dia dia NCMS000
. . Fp
```

No proveu per ara el programa amb cap altra frase, perquè segurament us trobareu amb un seguit de problemes que comentarem més endavant. Com podeu observar, la sortida no està desambiguada. El programa simplement escriu tota la informació per a cada una de les frases. Abans de solucionar els diferents problemes que presenta aquest senzill programa explicarem cada una de les seves parts.

En primer lloc, amb la part de codi següent:

```
import nltk
fdiccionari=open("diccionari-cat.txt","r")
diccionari=dict()
```

importem l'NLTK, obrim en mode lectura el fitxer que conté el diccionari i declarem un diccionari (l'estructura de dades de Python) que s'anomena *diccionari*. Recordeu que en aquest tipus d'estructura de dades podem emmagatzemar informació associada a una clau. En el nostre cas la clau serà la forma i la informació que emmagatzemen serà la informació associada a la forma. En el fragment de codi següent llegim el fitxer.

```
while True:
    linia=fdiccionari.readline().rstrip()
    if not linia: break
    camps=linia.split(" ")
    forma=camps.pop(0)
    informacio=" ".join(camps)
    diccionari[forma]=informacio
fdiccionari.close()
```

La informació del diccionari està separada per espais en blanc; per això fem un *split* amb aquest separador. La *forma* l'obtenim fem un `pop(0)` de *camps*. Això el que fa és treure de la llista el primer element i assignar-lo a la variable `forma`. Per a obtenir la variable `informacio` ajuntem amb un `join` la llista *camps* fent servir un espai en blanc com a separador. Finalment assignem al diccionari la *forma* com a clau i la *informacio* com a valor.

En la resta del codi definim una frase que *tokenitzem* amb el tokenitzador d'expressions regulars de l'NLTK:

Aquest tipus de tokenitzador ja el vam veure en el mòdul "Anàlisi textual i processament de corpus".

```
frase="avui la meva dona fa vacances, però demà treballarà tot el dia."
expressioregular=r'''(?x)
[\wàèòéíóúïüçÀÈÒÉÍÓÚÏÜÇ]+'?|
[\,\;\.\:\!\?\(\)]
'''
tokens=nltk.regexp_tokenize(frase,expressioregular)
```

Ja finalment recorrem la llista de *tokens* i escrivim cada un dels *tokens* amb la informació emmagatzemada en el diccionari.

Com ja he comentat, aquest petit programa presenta diversos errors i problemes que anirem solucionant en les properes línies.

En primer lloc haureu observat que el programa triga molt a executar-se. En aquesta versió estem treballant amb un diccionari molt gran que està en un fitxer de text. Aquest diccionari conté 634.646 formes i la seva informació associada. Cada cop que l'executem hem de llegir tot el fitxer per a poder muntar el diccionari. Podem crear un fitxer en disc que sigui el diccionari mateix en un format de base de dades, que serà més ràpid de llegir. En primer lloc executarem el programa següent (`crearbddiccionari.py`) per a crear la base de dades:

```
import anydbm
diccionari = anydbm.open('diccionari.db', 'c')
fdiccionari=open("diccionari-cat.txt","r")
while True:
    linia=fdiccionari.readline().rstrip()

    if not linia: break
    camps=linia.split(" ")
    forma=camps.pop(0)
    informacio=" ".join(camps)
    diccionari[forma]=informacio
fdiccionari.close()
diccionari.close()
```

Aquest programa només caldrà executar-lo un cop i tornar-lo a executar només si per algun motiu modifiquem el diccionari. A continuació oferim la versió modificada de l'analitzador morfològic (`programa-5-2.py`).

```
# -*- coding: utf-8 -*-
import nltk
import anydbm
diccionari = anydbm.open('diccionari.db', 'c')
frase="avui la meva dona fa vacances, però demà treballarà tot el dia."
expressioregular=r'''(?x)
[\wàèòéíóúïüçÀÈÒÉÍÓÚÏÜÇ]+'?|
[\,\;\.\:\!\?\(\)]
'''
tokens=nltk.regexp_tokenize(frase,expressioregular)
for token in tokens:
    print token+" "+diccionari[token]
```

Ara ja hem aconseguit millorar una mica la velocitat del nostre programa*. Si ara modifiqueu la frase i poseu "Avui la meva...", és a dir, la primera paraula en majúscules (com hauria de ser) es produirà un error. Això passa perquè totes les entrades del diccionari estan en minúscules (si l'obriu observareu que fins i tot els noms propis estan en minúscules). Així doncs, el que haurem de fer és passar el *token* a minúscules just quan consultem el diccionari. Simplement cal fer la modificació següent (el programa complet modificat el podeu trobar a `programa-5-3.py`) a la darrera línia del programa.

***Potser la millora no és espectacular, però de passada hem simplificat una mica el programa d'anàlisi.**

```
print token+" "+diccionari[token.lower()]
```

El nostre analitzador encara presenta problemes. Poseu una frase com per exemple "M'he comprat un ordinador amb multiprocessador i connexió Wi-fi". Es produirà un error perquè *Wifi* és una paraula desconeguda per al nostre sistema. Estem treballant amb un diccionari molt gran però sempre apareixeran paraules desconegudes, noms propis que no són al diccionari, etc. Hem de protegir el nostre programa per a aquests casos. Abans de programar la solució, haurem de pensar què volem fer amb les paraules desconegudes. Una primera aproximació pot ser simplement assignar-li l'etiqueta "DESCONEGUDA". En el programa següent (`programa-5-4-py`) podem observar la implementació d'aquesta solució.

```
# -*- coding: utf-8 -*-
import nltk
import anydbm
diccionari = anydbm.open('diccionari.db', 'c')
frase="M'he comprat un ordinador amb multiprocessador i connexió Wifi."
```

```
expressioregular=r'''(?x)
[\wàèòéíóúïüçÀÈÒÉÍÓÚÏÜÇ]+'?|
[\,\;\.\:\!\?\(\)]
'''
tokens=nltk.regexp_tokenize(frase,expressioregular)
for token in tokens:
    if diccionari.has_key(token.lower()):
     print token+" "+diccionari[token.lower()]
    else:
        print token +" DESCONEGUDA"
```

Ara podem afinar una mica més encara i en comptes de dir que és una paraula desconeguda dir el següent:

- Si la paraula no comença en majúscules dir que és un substantiu comú masculí singular (NCMS).
- Si la paraula comença en majúscules dir que és un substantiu propi masculí singular (NPMS).

Us deixo la implementació d'aquesta solució com a exercici.

Com a conclusió, veiem que fer un programa que faci l'anàlisi morfosintàctica és relativament senzill si disposem d'un bon diccionari morfològic. Ara bé, el que realment ens interessa és obtenir una sortida desambiguada, és a dir, una sortida amb una única interpretació de cada forma del text.

4. Desambiguació mitjançant regles creades manualment

Els primers sistemes de desambiguació mitjançant regles creades manualment es van crear cap als anys seixanta del segle XX. Cal tenir molt present que els ordinadors que hi havia en aquells anys no tenien res a veure amb els actuals. Klein i Simmons (1963) descriuen un sistema de desambiguació per a l'anglès capaç d'etiquetar 1.250 paraules per minut en un IBM 7090 (uns dels primers ordinadors que feien servir transistors en comptes de tubs de buit). Els autors afirmen que el sistema pot desambiguar de manera correcta el 90% de les paraules d'un text. Un aspecte important a tenir en compte respecte a la precisió d'un determinat etiquetador és el nombre total d'etiquetes, és a dir, el nombre total de categories i subcategories que té en compte. El sistema que estem descrivint assoleix una precisió elevada, però el nombre total d'etiquetes és reduït, 30 en total.

Mida de l'etiquetari

La mida de l'etiquetari o *tagset* és un aspecte important a tenir en compte quan avaluem un etiquetador. Com més petita sigui la mida més fàcil serà assolir nivells de precisió elevats.

Ordinador IBM 7090
Font: *http://commons.wikimedia.org/wiki/File:IBM_7090_computer.jpg)*

Un dels primers sistemes a gran escala per a l'anglès és el TAGGIT (Greene i Rubin, 1971). Aquest etiquetador es va fer servir per a fer un primer etiquetatge del Brown Corpus. Es basa en regles de context i fa servir un total de 71 etiquetes i un total de 3.300 regles. Amb aquest etiquetador es va aconseguir desambiguar el 77% del Brown Corpus.

Les gramàtiques de restriccions* (Karlsson i altres, 1995) proporcionen un formalisme eficaç per a escriure regles de desambiguació morfosintàctica i d'anàlisi sintàctica superficial. L'analitzador del català CATCG desenvolupat a la Universitat Pompeu Fabra (Alsina i altres, 2002) disposa d'un mòdul de desambiguació morfològica anomenat DeMCat que fa servir un formalisme de gramàtiques de restriccions. El nombre total de regles és del voltant de 1.000. L'estratègia bàsica d'aquests tipus de desambiguadors és seleccionar o eliminar certes etiquetes d'acord amb les restriccions imposades pel context limítrof. Les regles expressen una etiqueta destí sobre la qual s'actua, un operador que indica si se selecciona o s'eliminen certes etiquetes i un context que especifica les paraules o etiquetes limítrofes que han d'apareixer perquè s'apliqui la regla. Les posicions del context s'indiquen amb nombres enters, positius si es refereix a la dreta de l'objectiu i negatius si es refereix a l'esquerra. El zero indica la mateixa posició destí.

***En anglès, *Constraint Grammar*.**

Per exemple, la següent regla indica que s'elimini l'etiqueta "VERB" si també té l'etiqueta "NOM" i la paraula anterior té l'etiqueta "DETerminant" (i pot tenir-ne d'altres) i l'anterior a aquesta és una "PREPosició" de manera no ambigua (és a dir, que només té l'etiqueta "PREP").

```
(Rule1)
REMOVE (VERB) IF (0 NOM) (-1 DET) (-2C PREP)
```

Una característica típica de les gramàtiques desenvolupades manualment és el seu elevat grau de lexicalització, és a dir, moltes regles fan referència no únicament a etiquetes sinó també a paraules (lemes o fins i tot formes concretes). Al voltant del 40% de les 1.100 regles d'ENGCG (Voutilainen i Heikkilä, 1993) s'expressen en funció de formes i no d'etiquetes.

5. Desambiguació mitjançant tècniques estadístiques

Les tècniques de desambiguació mitjançant tècniques estadístiques es basen en la construcció d'un model del llenguatge a partir de corpus etiquetats prèviament.

L'etiquetatge estadístic assumeix que cada paraula que volem analitzar és coneguda i que té associat un conjunt finit d'etiquetes. El conjunt d'etiquetes de cada paraula es pot obtenir d'un diccionari, d'un analitzador morfològic o bé del corpus etiquetat d'aprenentatge mateix. Quan una o més paraules de la frase per analitzar té associada més d'una etiqueta, el mètode estadístic que fem servir ens ha de permetre determinar la seqüència òptima d'etiquetes ($E = e_1,e_2,e_3,...e_n$) donada una seqüència de paraules ($P = p_1,p_2,p_3,...,p_n$).

Per a solucionar aquest problema sovint es modela el problema mitjançant el **model del canal sorollós*** (Shannon, 1948). Aquest model suposa que els senyals o les dades observades (en el nostre cas la seqüència de les paraules de la frase) són el resultat d'emetre el senyal o les dades que volem obtenir (en el nostre cas la seqüència d'etiquetes) a través d'un canal que produeix una distorsió o soroll.

*En anglès, *noisy channel model.*

$$e_1,e_2,e_3,...,e_n \rightarrow \text{canal sorollós} \rightarrow pe_1,p_2,p_3,...;\ p_n$$

La seqüència d'etiquetes òptima si coneixem la seqüència de paraules correspon a la maximització de la probabilitat condicional:

$$\hat{T} = P(e_1,e_2,e_3,...,e_n|p_1,p_2,p_3,...,t_n)$$

El teorema de Bayes sobre les probabilitats dels esdeveniments A i B afirma que:

$$P(A|B)P(B) = P(B|A)P(A)$$

i per tant

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Si ara fem que $P(P) = pe_1,p_2,p_3,...;\ p_n$ i que $P(E) = e_1,e_2,e_3,...,e_n$ fent servir el teorema de Bayes obtenim que la seqüència d'etiquetes més probable és:

$$\hat{T} = argmax\frac{P(E)P(P|E)}{P(P)}$$

Amb una seqüència de paraules $p_1,p_2,p_3,...,t_n$ determinada $P(P)$ és constant i es pot eliminar, amb el que queda:

$$\hat{T} = argmaxP(E)P(P|E)$$

Aquests paràmetres es poden obtenir directament del corpus anotat, ja que $P(E)$ és la probabilitat d'una determinada seqüència d'etiquetes i $P(P|E)$ és la probabilitat d'una determinada seqüència de paraules donada una determinada seqüència d'etiquetes.

Lectura recomanada

Per a aprofundir en el tema de l'estadística i probabilitat aplicades a la lingüística és recomanable llegir *The Linguist's Guide to Statistics - Don't Panic* (1997), de Brigitte Krenn i Christer Samuelsson. Aquest llibre es pot descarregar d'Internet.

5.1. L'etiquetador per defecte

En aquest subapartat programarem un etiquetador molt simple, que l'únic que fa és etiquetar totes les paraules amb l'etiqueta més freqüent del nostre corpus d'entrenament. Ja ens podem imaginar que aquest etiquetador no funcionarà gaire bé, però ens pot servir per a establir un límit inferior (*baseline*) per a la precisió de la resta d'etiquetadors que anem estudiant en aquest mòdul. Totes les millores que anem introduint en els nostres etiquetadors hauran de millorar aquesta precisió.

Anem a descobrir quina és l'etiqueta més freqüent per a l'anglès a partir del Brown Corpus i per al català a partir del corpus Cess Cat (en el cas del català farem servir les etiquetes simplificades). Per a fer això podem fer (el càlcul pot trigar una mica):

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

Com podeu observar, l'etiqueta més freqüent en anglès és "NN" i en català (amb l'etiquetari simplificat) és "N".

L'NLTK ens permet avaluar ràpidament la precisió dels etiquetadors que anem dissenyant. En les línies següents podeu veure la manera de fer-ho.

Nota

L'avaluació pot trigar una bona estona, ja que la mida dels corpus és relativament gran, tot i que ara per a l'anglès estem limitant el corpus a la secció *news*; i per al català a les 1.000 primeres oracions).

```
>>> default_tagger=nltk.DefaultTagger('NN')
>>> brown_tagged_sents=brown.tagged_sents(categories='news')
>>> default_tagger.evaluate(brown_tagged_sents)
0.13089484257215028
```

```
>>> default_tagger=nltk.DefaultTagger('N')
>>> cess_cat_tagged_sents=cess_cat.tagged_sents(simplify_tags=True)
>>> default_tagger.evaluate(cess_cat_tagged_sents[0:1000])
0.25654112022740488
```

Com podeu observar les precisions obtingudes són molt baixes, del 13% en anglès i del 25% en català.

5.2. L'etiquetador per unigrames

L'etiquetador per defecte estudiat en el subapartat 5.1. etiqueta totes les paraules amb l'etiqueta més freqüent en tot el corpus. En les propers subapartats estudiarem una sèrie d'etiquetadors anomenats genèricament etiquetadors per *n-grames*. 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.

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

```
>>> import nltk
>>> from nltk.corpus import brown
>>> brown_tagged_sents=brown.tagged_sents(categories='news')
>>> unigram_tagger=nltk.UnigramTagger(brown_tagged_sents)
>>> unigram_tagger.evaluate(brown_tagged_sents)
0.9349006503968017
```

Veiem que la precisió ara és molt alta. El que passa és que hem fet una mica de trampa. Hem entrenat l'etiquetador amb un corpus i l'hem avaluat amb el mateix corpus. Això no es pot fer, ja que juguem amb avantatge. El que farem ara és entrenar el corpus amb el 90% de les oracions i avaluar-lo amb el 10% restant. Primer hem de saber quantes oracions té el fragment de corpus que estem fent servir. Ho podeu saber de la manera següent:

Corpus d'entrenament i d'avaluació

No es pot avaluar un etiquetador amb un fragment del mateix corpus que s'ha fet servir per a entrenar-lo. El que es fa és dividir el corpus en una part més gran per a entrenar el corpus i una altra part de més petita per a avaluar-lo.

```
>>> mida=len(brown_tagged_sents)
>>> mida
4623
>>> mida*0.9
4160.6999999999998
>>> mida*0.1
462.30000000000001
```

Així doncs, ara entrenarem el corpus amb les 4.161 primeres oracions i l'avaluarem amb les 462 darreres oracions.

```
>>> train_sents=brown_tagged_sents[:4160]
>>> test_sents=brown_tagged_sents[4160:]
>>> unigram_tagger=nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
0.81202033290142528
```

En aquests exemples hem treballat només amb la secció *news* del corpus Brown. Podeu repetir l'exercici amb tot el corpus, però tot el procés serà més lent i haureu de tenir una mica de paciència.

Ara repetirem aquests experiments amb l'etiquetador per unigrames amb el corpus català. El primer que farem serà mirar quantes oracions té el corpus per a decidir una mida de corpus d'entrenament i de corpus d'avaluació.

```
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> mida=len(cess_cat_tagged_sents)
>>> mida
17104
```

El que farem perquè no trigui massa és establir un corpus d'entrenament amb les 5.000 primeres oracions i un de prova amb les 1.000 següents:

```
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> unigram_tagger=nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
0.84834750521418256
```

Podeu veure com etiqueta un parell de frases qualssevol del corpus (però que no formin part de l'entrenament); per exemple la 9.016 i la 9.022:

```
>>> unigram_tagger.tag(cess_cat_sents[9016])
[('El', 'da0ms0'), ('president', 'ncms000'), ('va', 'vaip3s0'),
('presentar', 'vmn0000'), ('despr\xe9s', 'rg'), ('el', 'da0ms0'),
('Govern', 'np0000o'), ('en','sps00'), ('un', 'di0ms0'),
('gran', 'aq0cs0'), ('acte', 'ncms000'), ('a', 'sps00'),
(l·l'", 'da0cs0'), ('Auditori', None), ('.', 'Fp')]
>>> unigram_tagger.tag(cess_cat_sents[9022])
[('Els', 'da0mp0'), ('paquets', 'ncmp000'), ('portaven', 'vmii3p0'),
('una', 'di0fs0'), ('etiqueta', 'ncfs000'), ('a', 'sps00'),
('la', 'da0fs0'), ('part', 'ncfs000'), ('exterior', 'aq0cs0'),
('que', 'pr0cn000'), ('els', 'da0mp0'), ('descrivia', None),
('com_a', 'sps00'), ('productes', 'ncmp000'), ('arom\xe0tics', None),
('i', 'cc'), ('comptaven', None), ('tamb\xe9', 'rg'), ('amb', 'sps00'),
('instruccions', 'ncfp000'), ("d'", 'sps00'), ('\xfas', 'ncms000'),
('que', 'pr0cn000'), ('la', 'da0fs0'), ('policia', 'nccs000'),
('va', 'vaip3s0'), ('qualificar','vmn0000'), ('de', 'sps00'),
('"', 'Fe'), ('c\xednicament', None),('perilloses', None), ('"', 'Fe'),
('.', 'Fp')]
```

Com veiem, les paraules *Auditori*, *aromàtics*, *comptaven*, *cínicament* i *perilloses* les etiqueta amb "None", és a dir, no les ha pogudes etiquetar. Això es deu al fet que aquestes paraules no apareixien en el corpus d'entrenament.

Podeu provar què passa si augmenteu el corpus d'entrenament (compte de no incloure frases que estiguin en el corpus d'avaluació). Augmenta significativament la precisió?

En aquest cas hem treballat amb les etiquetes completes. També serà interessant observar què passa si treballem amb les etiquetes simplificades:

```
>>> cess_cat_tagged_sents=cess_cat.tagged_sents(simplify_tags=True)
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> unigram_tagger=nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
0.87221241777635172
```

Com calia esperar, la precisió ha augmentat, ja que treballem amb un conjunt d'etiquetes més reduït.

5.3. Tractament de paraules desconegudes: l'etiquetador per afixos

Un problema important per als etiquetadors estadístics és determinar l'etiqueta d'una paraula desconeguda, és a dir, d'una paraula que no aparegui en el

corpus d'entrenament. Una possibilitat de tractament de les paraules desconegudes és fixar-nos en les terminacions de les paraules conegudes (Calberger i Viggo, 1999).

Amb NLTK podem crear fàcilment un etiquetador per afixos, que trobi l'etiqueta més probable per a les paraules que continguin aquest afix. Aquest etiquetador es pot fixar tant en els prefixos (si la mida en caràcters s'especifica mitjançant un nombre enter positiu) com en els sufixos (si la mida en caràcters s'especifica mitjançant un nombre enter negatiu). En l'exemple següent creem un etiquetador per sufixos per a l'anglès, amb una mida de sufix de 3 i una arrel mínima de 2 (aquesta arrel mínima indica que no es faci servir el sufix si almenys l'arrel resultant no consta de 2 caràcters com a mínim). Podeu observar que la precisió no és gaire elevada (0,247). Posteriorment creem un etiquetador per unigrames i fem *backoff* amb el de sufixos (és a dir, les paraules que no pot etiquetar amb el d'unigrames s'etiqueten amb el de sufixos) i veiem que la precisió augmenta (0,863) i que millora una mica la d'unigrames únicament (0,848).

```
>>> import nltk
>>> from nltk.corpus import brown
>>> brown_tagged_sents=brown.tagged_sents(categories='news')
>>> train_sents=brown_tagged_sents[:4160]
>>> test_sents=brown_tagged_sents[4160:]
>>> affix_tagger=nltk.AffixTagger(train_sents, affix_length=-3, min_stem_length=2)
>>> affix_tagger.evaluate(test_sents)
0.24668593640984751
>>> unigram_tagger=nltk.UnigramTagger(train_sents, backoff=affix_tagger)
>>> unigram_tagger.evaluate(test_sents)
0.86305192863550284
```

Podem fer el mateix per al català i observar les precisions que s'assoleixen i com s'etiqueten les oracions 9016 i 9022 del corpus:

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> cess_cat_sents=cess_cat.sents()
>>> affix_tagger=nltk.AffixTagger(train_sents, affix_length=-3, min_stem_length=2)
>>> affix_tagger.evaluate(test_sents)
0.24727258142146638
>>> unigram_tagger=nltk.UnigramTagger(train_sents, backoff=affix_tagger)
>>> unigram_tagger.evaluate(test_sents)
0.88348307396117443
```

```
>>> unigram_tagger.tag(cess_cat_sents[9016])[('El', 'da0ms0'),
('president', 'ncms000'), ('va', 'vaip3s0'), ('presentar', 'vmn0000'),
('despr\xe9s', 'rg'), ('el', 'da0ms0'), ('Govern', 'np0000o'), ('en','sps00'),
('un', 'di0ms0'), ('gran', 'aq0cs0'), ('acte', 'ncms000'),
('a', 'sps00'), (l·l'", 'da0cs0'), ('Auditori', 'ncms000'), ('.', 'Fp')]
>>> unigram_tagger.tag(cess_cat_sents[9022])
[('Els', 'da0mp0'), ('paquets', 'ncmp000'), ('portaven', 'vmii3p0'),
('una', 'di0fs0'), ('etiqueta', 'ncfs000'), ('a', 'sps00'), ('la', 'da0fs0'),
('part', 'ncfs000'), ('exterior', 'aq0cs0'), ('que', 'pr0cn000'),
('els', 'da0mp0'), ('descrivia', 'vaii3s0'), ('com_a', 'sps00'),
('productes', 'ncmp000'), ('arom\xe0tics', 'aq0mp0'), ('i', 'cc'),
('comptaven', 'vmii3p0'), ('tamb\xe9', 'rg'), ('amb', 'sps00'),
('instruccions', 'ncfp000'), ("d'", 'sps00'), ('\xfas', 'ncms000'),
('que', 'pr0cn000'), ('la', 'da0fs0'), ('policia', 'nccs000'), ('va', 'vaip3s0'),
('qualificar', 'vmn0000'), ('de', 'sps00'), ('"', 'Fe'), ('c\xednicament', 'ncms000'),
('perilloses', 'ncfp000'), ('"', 'Fe'), ('.', 'Fp')]
```

Observeu els resultats de l'avaluació: l'etiquetador per unigrames sol assolia una precisió de 0,848 i ara combinant-lo amb un de sufixos s'arriba a una precisió de 0,883. Les paraules que no era capaç d'etiquetar l'etiquetador per unigrames sol (*Auditori*, *aromàtics*, *comptaven*, *cínicament* i *perilloses*) s'etiqueten correctament ara?

5.4. L'etiquetador per *n*-grames

L'etiquetador per *n*-grames és una generalització de l'etiquetador per unigrames. En el subapartat 5.2. hem vist com l'etiquetador per unigrames tenia en compte l'etiqueta més freqüent de la paraula, sense tenir en compte per a res el context d'aparició.

L'etiquetador per *n*-grames té en compte l'etiqueta de la paraules mateixa i les etiquetes de les n - 1 paraules precedents. Així, l'etiquetador per bigrames té en compte l'etiqueta d'una paraula i de l'anterior, i l'etiquetador per trigrames l'etiqueta d'una paraula i les etiquetes de les dues paraules anteriors.

5.4.1. L'etiquetador per bigrames

L'etiquetador per bigrames és un etiquetador que s'entrena observant l'etiqueta d'una paraula i de la paraula anterior. Amb l'NLTK és molt senzill crear i avaluar etiquetadors per bigrames.

En les línies de codi següents definim un etiquetador per bigrames i l'entrenem amb les primeres 5.000 oracions del corpus i l'avaluem amb les 1.000 oracions següents:

Nota

Els exemples següents els farem només per al català, però pot ser un bon exercici repetir-los per a l'anglès i observar les diferències que es produeixen en la precisió.

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> bigram_tagger=nltk.BigramTagger(train_sents)
>>> bigram_tagger.evaluate(test_sents)
0.14407187550136372
```

Com podeu veure, la precisió en aquest cas ha baixat molt. L'explicació és senzilla: a mesura que augmentem l'ordre de *n* es troben menys exemples en el corpus. Hi ha menys combinacions de dues paraules que paraules i això fa que siguin molt pocs bigrames els que es repeteixen després en el corpus d'avaluació.

Observem ara com s'etiqueten les oracions 9016 i 9022 amb l'etiquetador per bigrames:

```
>>> bigram_tagger.tag(cess_cat_sents[9016])
[('El', 'da0ms0'), ('president', 'ncms000'), ('va', 'vaip3s0'),
('presentar', 'vmn0000'), ('despr\xe9s', 'rg'), ('el', 'da0ms0'),
('Govern', 'np0000o'), ('en','sps00'), ('un', 'di0ms0'), ('gran', 'aq0cs0'),
('acte', None), ('a', None), (l·l'", None), ('Auditori', None), ('.', None)]
>>> bigram_tagger.tag(cess_cat_sents[9022])
[('Els', 'da0mp0'), ('paquets', 'ncmp000'), ('portaven', None), ('una', None),
('etiqueta', None), ('a', None), ('la', None), ('part', None), ('exterior', None),
('que', None), ('els', None), ('descrivia', None), ('com_a', None),
('productes', None), ('arom\xe0tics', None), ('i', None), ('comptaven', None),
('tamb\xe9', None), ('amb', None), ('instruccions', None), ("d'", None),
('\xfas', None), ('que', None), ('la', None), ('policia', None), ('va', None),
('qualificar', None), ('de', None), ('"', None), ('c\xednicament', None),
('perilloses', None), ('"', None), ('.', None)]
```

Fixeu-vos com moltes paraules queden sense etiquetar (etiqueta `None`).

Dispersió de dades

A mesura que creix l'ordre *n* dels etiquetadors per *n*-grames s'accentua el problema de la dispersió de dades, ja que necessitem corpus de mida molt més gran per a poder tenir suficients exemples d'aprenentatge.

Aquest problema sovint s'anomena **dispersió de dades** o *data sparseness*. Per a poder entrenar un etiquetador per bigrames que sigui efectiu necessitem disposar d'un corpus molt més gran, per a poder trobar més bigrames i poder assolir precisions més elevades. Fixeu-vos com augmenta la precisió si fem servir les primeres 15.000 oracions per a entrenar i les següents 1.000 per a avaluar.

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> train_sents=cess_cat.tagged_sents()[:15000]
>>> test_sents=cess_cat.tagged_sents()[15000:16000]
>>> bigram_tagger=nltk.BigramTagger(train_sents)
>>> bigram_tagger.evaluate(test_sents)
0.19304948385437715
```

Observeu que, tot i triplicar la mida del corpus d'entrenament, la precisió que s'assoleix és encara molt baixa.

5.4.2. L'etiquetador per trigrames

L'etiquetador per trigrames té en compte la paraula actual i dues paraules anteriors. Si avaluem per separat aquest etiquetador també tindrà una precisió molt baixa.

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> trigram_tagger=nltk.TrigramTagger(train_sents)
>>> trigram_tagger.evaluate(test_sents)
0.081220920904861216
```

I ara el resultat d'etiquetar les oracions 9016 i 9022 només amb l'etiquetador per trigrames:

```
>>> trigram_tagger.tag(cess_cat_sents[9016])
[('El', 'da0ms0'), ('president', 'ncms000'), ('va', 'vaip3s0'),
('presentar', None), ('despr\xe9s', None), ('el', None), ('Govern', None),
('en', None), ('un',None), ('gran', None), ('acte', None), ('a', None),
(l·l'", None), ('Auditori', None), ('.', None)]
>>> trigram_tagger.tag(cess_cat_sents[9022])
[('Els', 'da0mp0'), ('paquets', None), ('portaven', None), ('una', None),
('etiqueta', None), ('a', None), ('la', None), ('part', None),
('exterior', None), ('que', None), ('els', None), ('descrivia', None),
('com_a', None), ('productes', None), ('arom\xe0tics', None), ('i', None),
('comptaven', None), ('tamb\xe9', None), ('amb', None), ('instruccions', None),
("d'", None), ('\xfas', None), ('que', None), ('la', None), ('policia', None),
('va', None), ('qualificar', None), ('de', None), ('"', None),
('c\xednicament', None), ('perilloses', None), ('"', None), ('.', None)]
```

En aquest cas el problema de la dispersió de dades és encara més accentuat.

5.4.3. Més sobre combinació d'etiquetadors

En subapartats anteriors ja hem combinat un etiquetador d'unigrames amb un etiquetador de sufixos. El que feia aquesta combinació era etiquetar primer amb unigrames i en cas de no trobar cap etiqueta per a una paraula provar-la d'etiquetar amb sufixos. Aquesta estratègia rep el nom de *backoff*. Ara combinarem més etiquetadors: trigrames, bigrames, unigrames, afixos i l'etiquetador per defecte. En una sessió interactiva de Python feu el següent:

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> train_sents=cess_cat_tagged_sents[:5000]
>>> test_sents=cess_cat_tagged_sents[5000:6000]
>>> t_defecte=nltk.DefaultTagger('ncms000')
>>> t_sufixos=nltk.AffixTagger(train_sents, affix_length=-3,
      min_stem_length=2,backoff=t_defecte)
>>> t_unigrames=nltk.UnigramTagger(train_sents, backoff=t_sufixos)
>>> t_bigrames=nltk.BigramTagger(train_sents, backoff=t_unigrames)
>>> t_trigrames=nltk.TrigramTagger(train_sents, backoff=t_trigrames)
>>> t_trigrames=nltk.TrigramTagger(train_sents, backoff=t_bigrames)
>>> t_trigrames.evaluate(test_sents)
0.894713621049254
```

Veiem que per a aquesta combinació d'etiquetadors hem obtingut la precisió més elevada. Ara provarem si la precisió augmenta significativament si tripliquem la mida del corpus d'entrenament:

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> train_sents=cess_cat.tagged_sents()[:15000]
>>> test_sents=cess_cat.tagged_sents()[15000:16000]
>>> t_defecte=nltk.DefaultTagger('ncms000')
>>> t_sufixos=nltk.AffixTagger(train_sents, affix_length=-3,
      min_stem_length=2,backoff=t_defecte)
>>> t_unigrames=nltk.UnigramTagger(train_sents, backoff=t_sufixos)
>>> t_bigrames=nltk.BigramTagger(train_sents, backoff=t_unigrames)
>>> t_trigrames=nltk.TrigramTagger(train_sents, backoff=t_bigrames)
>>> t_trigrames.evaluate(test_sents)
0.91906983578854562
```

Observeu que la precisió ha augmentat. També observeu que el temps d'entrenament de cada etiquetador és més elevat. En general, si disposem de corpus etiquetats grans interessarà fer servir corpus d'entrenament de gran mida.

Com que els temps d'entrenament seran elevats, caldrà buscar algun mecanisme que ens permeti emmagatzemar els etiquetats de manera que no hàgim d'entrenar-los cada cop que necessitem etiquetar un text.

5.5. Emmagatzematge d'etiquetadors

Com hem vist, l'entrenament d'un etiquetador pot trigar una bona estona, sobretot si fem servir corpus grans. Un cop entrenat l'etiquetador ens interessarà emmagatzemar-lo d'alguna manera, de manera que quan hàgim d'etiquetar algun text ens estalviem el procés d'entrenament. El procés és molt senzill i el descrivim en les línies següents*.

***Suposeu que venim de la mateixa sessió interactiva del subapartat 5.4. i que, per tant, els etiquetadors ja estan creats.**

```
>>> from cPickle import dump
>>> sortida=open('etiquetador.pkl', 'wb')
>>> dump(t_trigrames, sortida, -1)
>>> sortida.close()
```

Un cop emmagatzemat l'etiquetador el podem carregar en qualsevol moment fent:

```
>>> import nltk
>>> from cPickle import load
>>> entrada=open('etiquetador.pkl','rb')
>>> etiquetador=load(entrada)
>>> entrada.close()
```

Si volem que el nostre etiquetador ens etiqueti una determinada frase podem fer:

```
>>> frase="El meu amic Joan va anar a casa seva i es va trobar una sorpresa"
>>> tokens=frase.split()
>>> etiquetador.tag(tokens)
[('El', 'da0ms0'), ('meu', 'dp1mss'), ('amic', 'ncms000'), ('Joan', 'ncms000'),
('va', 'vaip3s0'), ('anar', 'vmn0000'), ('a', 'sps00'), ('casa', 'ncfs000'),
('seva', 'dp3fs0'), ('i', 'cc'), ('es', 'p0000000'), ('va', 'vaip3s0'),
('trobar','vmn0000'), ('una', 'di0fs0'), ('sorpresa', 'ncfs000')]
```

5.6. Més sobre etiquetatge de paraules desconegudes

En el subapartat 5.3. hem comentat una estratègia de tractament de paraules desconegudes que consisteix a analitzar els sufixos de les paraules del corpus i etiquetar les paraules desconegudes segons el seu sufix. Aquesta estratègia

té en compte únicament els sufixos, sense analitzar els contextos d'aparició. En aquest subapartat analitzarem una estratègia de tractament de les paraules desconegudes que té en compte el seu context d'aparició. Per a veure com es comporten les paraules desconegudes no podem fer servir directament el corpus d'entrenament, ja que en aquest corpus totes les paraules són conegudes. El que farem serà modificar el corpus d'entrenament de manera que algunes paraules conegudes es transformin en desconegudes, però mantenint l'etiqueta que tenien originàriament.

```
>>> import nltk
>>> from nltk.corpus import cess_cat
>>> cess_cat_tagged_sents=cess_cat.tagged_sents()
>>> paraules_corpus=cess_cat.words()
>>> vocabulari=nltk.FreqDist(paraules_corpus)
>>> vocabulari1000=list(vocabulari)[:1000]
>>> mapping=nltk.defaultdict(lambda: 'DESC')
>>> for v in vocabulari1000:
...     mapping[v]=v
>>> cess_cat_mod=[]
>>> for frase in cess_cat_tagged_sents:
...     frasemod=[(mapping[p],e) for (p,e) in frase]
...     cess_cat_mod.append(frasemod)
```

Ara disposem de dues versions del `cess_cat`, la normal i la modificada. La modificada canvia les paraules que no siguin de les 1.000 més freqüents per "DESC", però mantenint l'etiqueta original. Podem observar les dues versions del corpus, si fem:

```
>>> cess_cat_tagged_sents[1]
[('La', 'da0fs0'), ('sent\xe8ncia', 'ncfs000'), (',', 'Fc'), ('a', 'sps00'),
('la', 'da0fs0'), ('qual', 'pr0cs000'), ('ha', 'vaip3s0'), ('tingut', 'vmp00sm'),
('acc\xe9s', 'ncms000'), ('Intra-ACN', 'np0000o'), (',', 'Fc'),
('desestima', 'vmm02s0'), ('els', 'da0mp0'), ('recursos', 'ncmp000'),
('interposats', 'aq0mpp'),('pels', 'spcmp'), ('processats', 'ncmp000'),
(',', 'Fc'), ('Albert_Bram\xf3n','np0000p'), (',', 'Fc'), ('president', 'ncms000'),
('del', 'spcms'), ('Col\xb7legi_de_Veterinaris_de_Girona', 'np0000o'),
('en', 'sps00'), ('el', 'da0ms0'), ('moment', 'ncms000'), ('dels', 'spcmp'),
('fets', 'ncmp000')...
>>> cess_cat_mod[1]
[('La', 'da0fs0'), ('sent\xe8ncia', 'ncfs000'), (',', 'Fc'), ('a', 'sps00'),
('la', 'da0fs0'), ('qual', 'pr0cs000'),  ('ha', 'vaip3s0'), ('tingut', 'vmp00sm'),
('acc\xe9s', 'ncms000'), ('DESC', 'np0000o'), (',', 'Fc'), ('DESC', 'vmm02s0'),
('els', 'da0mp0'), ('recursos', 'ncmp000'), ('DESC', 'aq0mpp'), ('pels', 'spcmp'),
('DESC', 'ncmp000'), (',', 'Fc'), ('DESC', 'np0000p'), (',', 'Fc'),
('president', 'ncms000'), ('del', 'spcms'), ('DESC', 'np0000o'), ('en', 'sps00'),
('el', 'da0ms0'), ('moment', 'ncms000'), ('dels', 'spcmp'), ('fets', 'ncmp000')...
```

Ara podríem entrenar un etiquetador de bigrames o de trigrames amb el corpus modificat per a entrenar-lo a etiquetar les paraules marcades com a desconegudes segons el seu context. De fet, en crearem un de trigrames amb *backoff* de bigrames i aquest a la seva vegada amb *backoff* d'unigrames. Això ho farem per a les primeres 5.000 oracions del corpus i les 1.000 següents les farem servir per a avaluar.

```
>>> train_corpus_mod=cess_cat_mod[:5000]
>>> test_corpus_mod=cess_cat_mod[5000:6000]
>>> unigram_desc_tagger=nltk.UnigramTagger(train_corpus_mod)
>>> bigram_des_tagger=nltk.BigramTagger(train_corpus_mod,backoff=unigram_desc_tagger)
>>> bigram_desc_tagger=nltk.BigramTagger(train_corpus_mod,backoff=unigram_desc_tagger)
>>> trigram_desc_tagger=nltk.TrigramTagger(train_corpus_mod,backoff=bigram_desc_tagger)
>>> trigram_desc_tagger.evaluate(test_corpus_mod)
0.78750200545483717
```

La precisió que hem obtingut no és exactament la precisió en l'etiquetatge de les paraules desconegudes (les que tenen com a forma "DESC" en el corpus modificat), sinó la precisió total de l'etiquetador. En les línies següents podem observar el codi* que permet calcular la precisió assolida només comptant les paraules desconegudes:

***El codi complet del programa es troba a** `desconegudes2.py`.

```
>>> corpus_etiquetat=[]
>>> for frase in test_corpus_mod:
...     paraules=[]
...     for (paraula,etiqueta) in frase:
...             paraules.append(paraula)
...     trigram_desc_tagger.tag(paraules)
...     corpus_etiquetat.append(frase)
contfrase=0
contdesc=0
correctes=0
for frase in corpus_etiquetat:
    conttoken=0
    for (paraula, etiqueta) in frase:
        if (paraula=="DESC"):
            contdesc+=1
            (paraula2,etiqueta2)=test_corpus_mod[contfrase][conttoken]

            if (etiqueta2==etiqueta):
                print paraula, paraula2, etiqueta, etiqueta2
                correctes+=1
        conttoken+=1
    contfrase+=1
print correctes
```

```
2339
print contdesc
6910
precisio=100*(float(correctes)/contdesc)
print precisio
33.8494934877
```

Veieu que la precisió és baixa (33,85) però superior a la precisió de l'etiquetador per afixos (24,67). Si repetim l'experiment, però fent servir les etiquetes simplificades*, la precisió augmenta notablement (la precisió total augmenta de 0,78 a 0,88 i la de les desconegudes de 0,34 a 0,66).

***Trobareu el codi del programa a** `desconegudes2.py`.

Altres tècniques

Es poden aplicar altres tècniques similars per al tractament de les paraules desconegudes. Per exemple, es pot entrenar el corpus normalment, i quan etiquetem, si ens trobem amb una paraula desconeguda considerar que pot tenir totes les etiquetes corresponents a les categories obertes (substantius, adjectius i verbs) si comença amb minúscula, o bé les corresponents a noms propis si comença amb majúscula.

6. L'etiquetador de Brill

L'etiquetador de Brill (1995) és un cas especial. Aquest etiquetador fa servir un conjunt de regles, però aquestes regles es deriven automàticament a partir d'un corpus anotat. El que fa aquest etiquetador és etiquetar cada paraula amb la seva categoria més freqüent i després comparar aquest etiquetatge amb l'etiquetatge manual i aprendre una sèrie de regles de transformació que facin que l'etiquetatge automàtic millori. Aquesta mena d'aprenentatge basat en regles de transformació adquirides automàticament rep el nom d'*aprenentatge basat en transformacions**.

**En anglès, *transformation-based learning.*

NLTK implementa l'etiquetador de Brill i disposa d'una *demo* del seu funcionament. En una sessió interactiva de Python, escriviu el següent:

```
>>> import nltk
>>> nltk.tag.brill.demo()
Loading tagged data...
Done loading.
Training unigram tagger:
    [accuracy: 0.833114]
Training bigram tagger:
    [accuracy: 0.838631]
Training Brill tagger on 1600 sentences...
Finding initial useful rules...
    Found 9695 useful rules.

           B      |
   S   F   r   O  |        Score = Fixed - Broken
   c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
   o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
   r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
   e   d   n   r  |  e
------------------+-------------------------------------------------------
  11  15   4   0  | WDT -> IN if the tag of words i+1...i+2 is 'DT'
  10  12   2   0  | IN -> RB if the text of the following word is
                  |   'well'
   9   9   0   0  | WDT -> IN if the tag of the preceding word is
                  |   'NN', and the tag of the following word is 'NNP'
   7  10   3   0  | WDT -> IN if the tag of words i+1...i+2 is 'NNS'
   6   8   2   0  | RBR -> JJR if the tag of words i+1...i+2 is 'NNS'
   5   5   0   0  | WDT -> IN if the tag of the preceding word is
```

```
                |   'NN', and the tag of the following word is 'PRP'
   4   4   0   1  | WDT -> IN if the tag of words i+1...i+3 is 'VBG'
   3   3   0   0  | RB -> IN if the tag of the preceding word is 'NN',
                |   and the tag of the following word is 'DT'
   3   3   0   0  | NNS -> NN if the text of the preceding word is
                |   'one'
   3   3   0   0  | RP -> RB if the text of words i-3...i-1 is 'were'

Brill accuracy: 0.839243
Done; rules and errors saved to rules.yaml and errors.out.
```

Si observeu el que surt per pantalla veureu que el primer que es fa és entrenar un etiquetador per unigrames i un per bigrames amb *backoff* d'unigrames. Les precisions que assoleixen aquests etiquetadors són de 0,833 el d'unigrames i de 0,838 el de bigrames. L'etiquetador de bigrames es farà servir com a etiquetador previ. A partir de l'etiquetatge del corpus amb aquest etiquetador previ i del corpus d'entrenament mateix s'aniran derivant les regles de transformació que facin que es millorin els resultats. A la taula que hi apareix podeu observar les millors regles que ha derivat. Al final podem observar que la precisió obtinguda amb l'etiquetador de Brill és de 0,839.

A més de la *demo*, NLTk permet entrenar i fer servir etiquetadors de Brill. A continuació veurem com es pot entrenar un etiquetador de Brill i com es pot avaluar; primer ho farem per a l'anglès (`brill1.py`).

```
import nltk
from nltk.corpus import brown

brown_train = brown.tagged_sents(categories='news')[0:3000]
brown_test = brown.tagged_sents(categories='news')[3000:4000]

from nltk.tag.brill import *
templates = [
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,1)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (2,2)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,2)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,3)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,1)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (2,2)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,2)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,3)),
    ProximateTokensTemplate(ProximateTagsRule, (-1, -1), (1,1)),
    ProximateTokensTemplate(ProximateWordsRule, (-1, -1), (1,1)),
    ]
unigram_tagger = nltk.UnigramTagger(brown_train)
precisio=unigram_tagger.evaluate(brown_test)
```

```
print "PRECISIO UNIGRAMS: ", precisio

trainer = FastBrillTaggerTrainer(initial_tagger=unigram_tagger,
    templates=templates, trace=3,
    deterministic=True)
brill_tagger = trainer.train(brown_train, max_rules=10)
precisio=brill_tagger.evaluate(brown_test)
print "PRECISIO BRILL: ", precisio
```

Fixeu-vos que es defineixen una sèrie de plantilles* (*templates*) per a crear les regles de transformació. També limitem el nombre màxim de regles per derivar, en aquest cas a 10 (`max_rules=10`). Si executem aquest programa apareixerà la informació següent per pantalla:

***Podeu trobar més informació sobre aquestes plantilles a la documentació de l'NLTK.**

```
PRECISIO UNIGRAMS:  0.785423128046
Training Brill tagger on 3000 sentences...
Finding initial useful rules...
    Found 52007 useful rules.

           B      |
   S   F   r   O  |        Score = Fixed - Broken
   c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
   o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
   r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
   e   d   n   r  |  e
------------------+-------------------------------------------------------
 208 208   0   0  | TO -> IN if the tag of the following word is 'AT'
  70  70   0   2  | TO -> IN if the tag of the following word is 'NP'
  66 116  50   5  | NP -> NP-TL if the tag of the following word is
                  |   'NN-TL'
  59  77  18   3  | VB -> NN if the tag of words i-2...i-1 is 'AT'
  53  86  33   1  | TO -> IN if the tag of words i+1...i+2 is 'NNS'
  53  54   1   1  | VBN -> VBD if the tag of the preceding word is
                  |   'NP'
  46  81  35   2  | NN -> VB if the tag of the preceding word is 'TO'
  45  45   0   1  | NN -> VB if the tag of the preceding word is 'MD'
  38  38   0   0  | VBN -> VBD if the tag of the preceding word is
                  |   'PPS'
  36  39   3   0  | VBD -> VBN if the tag of words i-2...i-1 is 'BEDZ'
PRECISIO BRILL:  0.794505981391
```

Ara modificarem el nombre de regles a 100 i verem que la precisió augmenta lleugerament, a 0,802. Si posem el nombre màxim de regles a 1.000 observem que la precisió passa a 0,805. Podem també crear un etiquetador de Brill per al català i avaluar-lo (`brill2.py`).

```
import nltk
from nltk.corpus import cess_cat

cess_cat_train = cess_cat.tagged_sents()[0:3000]
cess_cat_test = cess_cat.tagged_sents()[3000:4000]

from nltk.tag.brill import *
templates = [
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,1)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (2,2)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,2)),
    SymmetricProximateTokensTemplate(ProximateTagsRule, (1,3)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,1)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (2,2)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,2)),
    SymmetricProximateTokensTemplate(ProximateWordsRule, (1,3)),
    ProximateTokensTemplate(ProximateTagsRule, (-1, -1), (1,1)),
     ProximateTokensTemplate(ProximateWordsRule, (-1, -1), (1,1)),
    ]
unigram_tagger = nltk.UnigramTagger(cess_cat_train)
precisio=unigram_tagger.evaluate(cess_cat_test)
print "PRECISIO UNIGRAMS: ", precisio

trainer = FastBrillTaggerTrainer(initial_tagger=unigram_tagger,
    templates=templates, trace=3,
    deterministic=True)
brill_tagger = trainer.train(cess_cat_train, max_rules=10)
precisio=brill_tagger.evaluate(cess_cat_test)
print "PRECISIO BRILL: ", precisio
```

Si executem el programa obtindrem la informació per pantalla següent:

```
PRECISIO UNIGRAMS:  0.830442652625
Training Brill tagger on 3000 sentences...
Finding initial useful rules...
    Found 45318 useful rules.

           B      |
   S   F   r   O  |        Score = Fixed - Broken
   c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
   o   x   k   h  |  u     Broken = num tags changed correct -> incorrect
   r   e   e   e  |  l     Other = num tags changed incorrect -> incorrect
   e   d   n   r  |  e
------------------+-------------------------------------------------------
 196 200   4   0  | pr0cn000 -> cs if the tag of words i-3...i-1 is
                  |   'vaip3s0'
```

```
 166 167   1   2  | pr0cn000 -> cs if the tag of words i-2...i-1 is
                  |   'vmip3s0'
  49  57   8   2  | pr0cn000 -> cs if the tag of words i-2...i-1 is
                  |   'vmn0000'
  49  49   0   0  | aq0msp -> vmp00sm if the text of the preceding
                  |   word is 'ha'
  35  36   1   9  | di0fs0 -> pi0fs000 if the tag of the following
                  |   word is 'sps00'
  34  34   0  12  | di0ms0 -> pi0ms000 if the tag of the following
                  |   word is 'spcmp'
  34  37   3   1  | pr0cn000 -> cs if the tag of words i-2...i-1 is
                  |   'vsip3s0'
  28  58  30   6  | p0000000 -> p0300000 if the tag of words i-2...i-1
                  |   is 'sn.e-SUJ'
  28  28   0   0  | pr0cn000 -> cs if the tag of the preceding word is
                  |   'vmip3p0'
  28  28   0   0  | sps00 -> cs if the tag of the following word is
                  |   'vaip3s0'
PRECISIO BRILL:  0.837650156327
```

En aquest programa també hem fet servir un màxim de 10 regles i passem del 0,83 de precisió de l'etiquetador d'unigrames a 0,837. Amb 100 regles la precisió passa a ser de 0,843 i amb 1.000 de 0,844.

Les precisions que assolim no són gaire espectaculars. Teniu en compte que l'etiquetador de Brill treballa amb un etiquetador previ, que en els programes `brill1.py` i `brill2.py` és simplement un etiquetador per unigrames. Podem fer servir un etiquetador que presenti una precisió més elevada com a etiquetador inicial.

Exercici

Experimenteu amb diferents mides de corpus d'entrenament i de nombre màxim de regles i observeu com varia la precisió de l'etiquetador. Proveu també de fer servir l'etiquetari simplificat i observeu com afecta aquest fet la precisió del sistema.

7. Tècniques d'aprenentatge automàtic aplicades a l'etiquetatge morfosintàctic

Sota el nom de tècniques d'aprenentatge automàtic aplicades a l'etiquetatge morfosintàctic agruparem una sèrie de tècniques que fan servir informació que va més enllà dels *n*-grames dels etiquetadors estadístics. Alguns d'aquests algorismes són capaços de treballar amb una gran quantitat d'atributs sense gaire problemes amb la dispersió de dades. Recordeu que quan entrenàvem etiquetadors basats en bigrames i en trigrames, la precisió d'aquests etiquetadors sense combinar era baixa perquè no es trobaven prou dades en el corpus. Les tècniques que presentarem en aquest apartat pateixen menys aquests problemes.

L'aprenentatge automàtic o artificial* és una àrea de la intel·ligència artificial que estudia algorismes que poden aprendre a partir de l'experiència pròpia o per mitjà de reorganitzar el coneixement de què disposen (Màrquez i altres, 2001).

*En anglès, *machine learning*.

Lectura recomanada

Són moltes les tècniques que es poden incloure en aquesta classificació i en aquest mòdul n'inclourem només unes poques. Si voleu consultar una llista més exhaustiva, podeu consultar l'obra de Màrquez (2001).

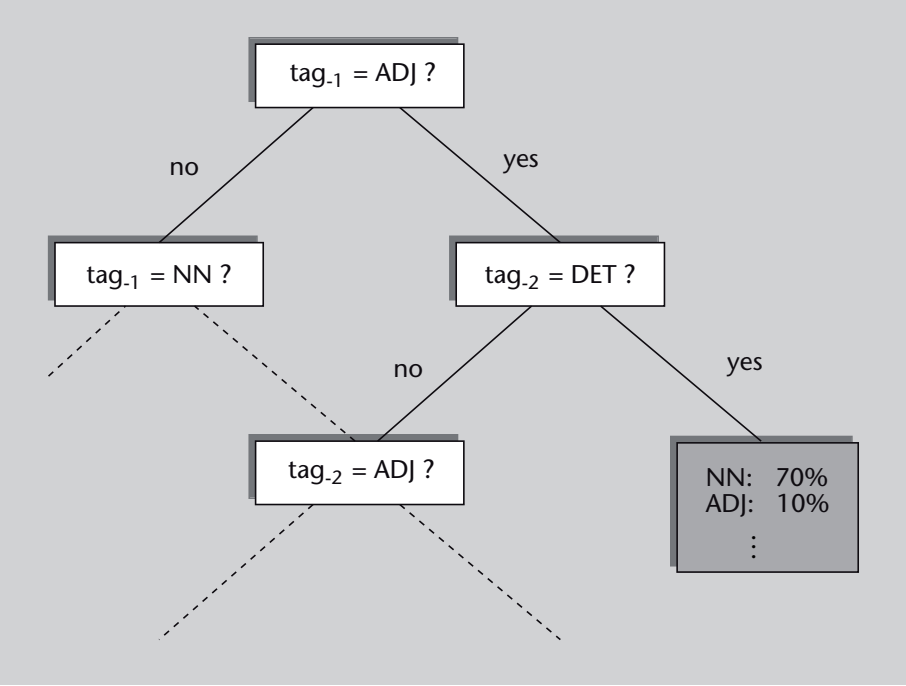
7.1. Arbres de decisió

Un arbre de decisió permet extreure i representar regles de classificació a partir d'un conjunt d'exemples mitjançant una estructura jeràrquica seqüencial. A la figura 3 podem veure un exemple de fragment d'arbre de decisió extret de (Schmid, 1994). En la part superior es planteja si l'etiqueta precedent és "ADJ"; si no ho és, es planteja si és "NN". En canvi, si ho és, es planteja si l'etiqueta de dues posicions abans és "DET" i si ho és, determina que l'etiqueta corresponent és "NN" amb un 70% de probabilitat i "ADJ" amb un 10%.

NLTK implementa diverses classes per a treballar amb arbres de decisió. No entrarem en detall, però podem executar una *demo* fent:

```
>>> import nltk
>>> nltk.classify.decisiontree.demo()
Training classifier...
best stump for   5000 toks uses endswith(vowel)=False err=0.2556
best stump for   1802 toks uses has(y)=False         err=0.3152
best stump for   1612 toks uses startswith(k)=False  err=0.3033
```

Figura 3. Fragment d'un arbre de decisió




```
...

best stump for    113 toks uses (default)           err=0.0619
best stump for      9 toks uses has(a)=False        err=0.2222
best stump for      8 toks uses count(m)=2          err=0.0000
Testing classifier...
Accuracy: 0.7960
endswith(vowel)=False? ............................. male
  has(y)=False? ..................................... male
    startswith(k)=False? ............................ male
      count(v)=2? .................................... male
        has(c)=False? ............................... female
        else: ....................................... male
      else: ......................................... male

...

      else: ......................................... female
        has(a)=False? ............................... female
        else: ....................................... male
    else: ........................................... female
      count(m)=2? .................................... male
      else: ......................................... female

if endswith(vowel) == False:
  if has(y) == False:
    if startswith(k) == False:
      if count(v) == 2:
```

```
        if has(c) == False: return 'female'
        if has(c) != False: return 'male'
      if count(v) != 2:
        if count(l) == 3:

...

      if startswith(c) != False:
        if has(a) == False: return 'female'
        if has(a) != False: return 'male'
    if has(k) != False:
      if count(m) == 2: return 'male'
      if count(m) != 2: return 'female'
```

El que fa aquesta *demo* és intentar classificar correctament en masculí o femení una llista de 7.944 noms propis anglesos, dels quals 2.943 són masculins i 5.001 són femenins.

En el subapartat 9.2. podeu veure una implementació pràctica d'aquests tipus d'etiquetadors.

7.2. Màquines de vectors de suport

Les **màquines de vectors de suport** o *support vector machines* són un conjunt d'algorismes d'aprenentatge automàtic que es basen en la creació d'uns espais de moltes dimensions i en la inducció de separacions lineals o hiperplans que separin els conjunts de mostres positius i negatius amb un marge màxim entre les dues classes.

A la figura 4 (extreta de l'obra de Nakagawa i altres (2001)) podeu veure que hi ha molts hiperplans que poden separar el conjunt de dades positives (figura de l'esquerra) i negatives d'entrenament. Les SVM troben l'hiperplà òptim que maximitza el marge (la distància entre l'hiperplà i els punts més propers) (figura de la dreta).

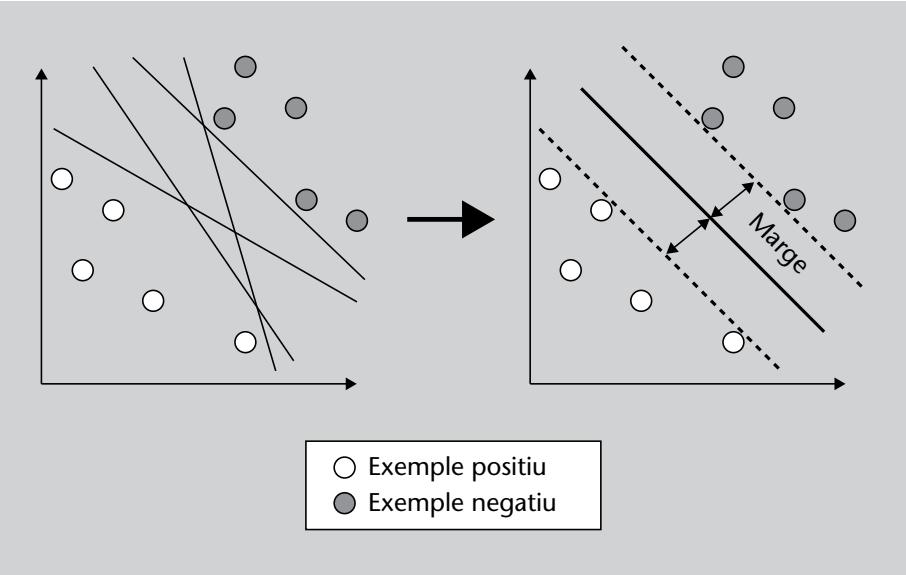
En el subapartat 9.3. podeu provar un etiquetador que funciona mitjançant aquesta tècnica d'aprenentatge (Giménez i Màrquez, 2004).

7.3. Combinació de classificadors

Algunes tècniques d'aprenentatge automàtic es basen en la combinació de diversos classificadors. S'ha demostrat que sota algunes condicions, la combinació de classificadors funciona millor que el millor dels classificadors combinats. Normalment la combinació es porta a terme mitjançant algun tipus de funció de votació.

Hi ha diverses aplicacions d'aquesta tècnica a l'etiquetatge morfosintàctic. En l'obra d'Halteren i altres (1998) s'entrenen 4 etiquetadors que fan servir

Figura 4. Maximització del marge en les SVM



tècniques ben conegudes (models ocults de Markov, transformacions basades en memòria, regles de transformació i entropia màxima) amb el mateix corpus. Després es fan servir diverses estratègies de votació i classificadors. Qualsevol d'aquestes combinacions milloren el millor dels etiquetadors.

Brill i Wu (1998) han observat que els errors que cometen tres etiquetadors ben coneguts són complementaris i que aquesta complementarietat es pot aprofitar per a combinar-los i obtenir un etiquetador amb més precisió. Els etiquetadors que combinen són:

- Un etiquetador per trigrames combinat amb un etiquetador per sufixos per a tractar les paraules desconegudes.
- Un etiquetador basat en transformacions (l'etiquetador de Brill que ja hem estudiat).
- Un etiquetador basat en entropia màxima.

Per a combinar els etiquetadors experimenten amb dues tècniques:

- **Votació simple**: s'agafa l'etiqueta més votada. En cas de desacord s'agafa l'etiqueta que dóna l'etiquetador basat en entropia màxima, ja que és el que ha demostrat tenir una precisió més elevada.
- **Ús de contextos**. Per a determinar l'etiqueta d'una paraula fa servir la paraula anterior, la paraula mateixa i la paraula següent i la sortida de cada etiquetador per a la paraula anterior, la paraula mateixa i la paraula següent. Tot això fa que es disposi d'un conjunt d'entrenament que permet obtenir les probabilitats de les etiquetes correctes que apareixen en aquests contextos.

A l'obra de Màrquez i altres (1998) es presenta una aplicació de la combinació d'etiquetadors que serveix per a entrenar etiquetadors estadístics amb corpus

relativament petits fent servir la tècnica de *bootstrapping*. El procediment s'inicia fent servir un corpus etiquetat manualment de mida petita per a entrenar un parell d'etiquetadors. Llavors s'etiqueta automàticament un corpus més gran i es retenen aquells fragments en què els dos etiquetadors donen les mateixes etiquetes. Aquest corpus nou s'afegeix a l'anterior per a reentrenar els etiquetadors. Aquest procés es repeteix fins que ja no s'assoleixen millores en la precisió.

8. Avaluació pràctica d'etiquetadors

En els apartats anteriors hem après a entrenar i avaluar etiquetadors. Sovint hem anat fent proves per veure com afecta la mida del corpus o altres paràmetres la precisió de l'etiquetador. En aquest apartat aprendrem a fer aquest tipus de proves d'una manera automàtica i a graficar els resultats. La idea és fer un programa que vagi variant els paràmetres i entrenant els etiquetadors i després aprofitar els resultats per a fer-ne un gràfic.

El primer exemple que mostrem (`avaluacio1.py`), extret directament de l'obra de Bird i altres (2009) avalua un etiquetador que funciona mitjançant la creació d'un diccionari a partir del corpus Brown i amb *backoff* d'un etiquetador per defecte que assigna la categoria "NN". L'etiquetador fa servir una sèrie de mides del corpus d'entrenament donades per la funció

```
sizes = 2 ** pylab.arange(15)
```

que genera la següent llista de 15 elements

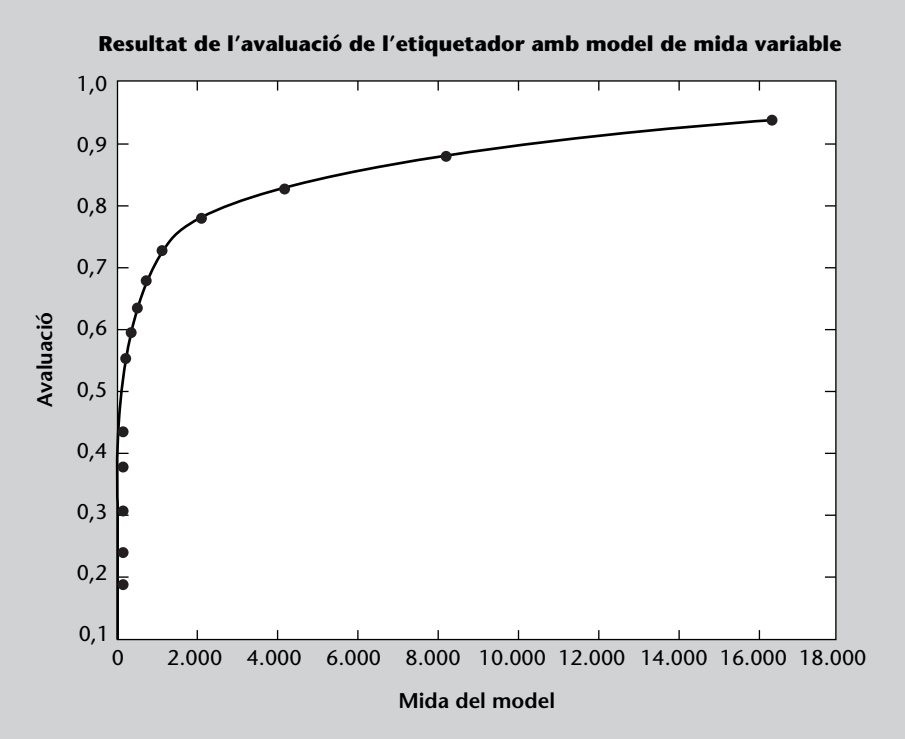
```
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384]
```

A la figura 5 es pot observar que la precisió al principi augmenta molt ràpidament, però que més endavant aquesta augmenta d'una manera molt més lenta.

```
import nltk
from nltk.corpus import brown

def performance(cfd, wordlist):
    lt=dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model=lt, backoff=nltk.DefaultTagger('NN'))
    precisio=baseline_tagger.evaluate(brown.tagged_sents(categories='news'))
    print precisio
    return precisio

def display():
    import pylab
    words_by_freq = list(nltk.FreqDist(brown.words(categories='news')))
    cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
```

```
    sizes = 2 ** pylab.arange(15)
    print sizes
    perfs = [performance(cfd, words_by_freq[:size]) for size in sizes]
    pylab.plot(sizes, perfs, '-bo')
    pylab.title('Lookup Tagger Performance with Varying Model Size')
    pylab.xlabel('Model Size')
    pylab.ylabel('Performance')
    pylab.show()

display()
```

Figura 5. Resultat de l'avaluació de l'etiquetador



Podem aprofitar aquest programa per a avaluar algun dels etiquetadors que hem estudiat en altres apartats. Per exemple, podem avaluar més extensament l'etiquetador presentat al subapartat 5.4.3. amb el programa següent (`avaluacio2.py`):

```
import nltk
from nltk.corpus import cess_cat

def precisio(mida):
    train_sents=cess_cat.tagged_sents()[:mida]
    test_sents=cess_cat.tagged_sents()[mida:(mida+1000)]
    t_defecte=nltk.DefaultTagger('ncms000')
    t_sufixos=nltk.AffixTagger(train_sents, affix_length=-3,
       min_stem_length=2,backoff=t_defecte)
```

```
    t_unigrames=nltk.UnigramTagger(train_sents, backoff=t_sufixos)
    t_bigrames=nltk.BigramTagger(train_sents, backoff=t_unigrames)
    t_trigrames=nltk.TrigramTagger(train_sents, backoff=t_bigrames)
    precisio=t_trigrames.evaluate(test_sents)
    print precisio
    return precisio

def nombre_paraules(mida):
    sents=cess_cat.sents()[:mida]
    paraules=0
    for frase in sents:
        paraules+=len(frase)
    return paraules


def display():
    import pylab
    mides = 100*(2 ** pylab.arange(8))
    print mides
    mides_paraules= [nombre_paraules(mida) for mida in mides]
    print mides_paraules
    precisions = [precisio(mida) for mida in mides]

    pylab.plot(mides_paraules, precisions, '-bo')
    pylab.title('Precisio Etiquetador per Trigrames')
    pylab.xlabel('Mida del model')
    pylab.ylabel('Precisio')
    pylab.show()

display()
```

A la figura 6 podeu veure el comportament de la precisió per a aquest etiquetador. Observeu que el comportament és similar al del cas anterior.

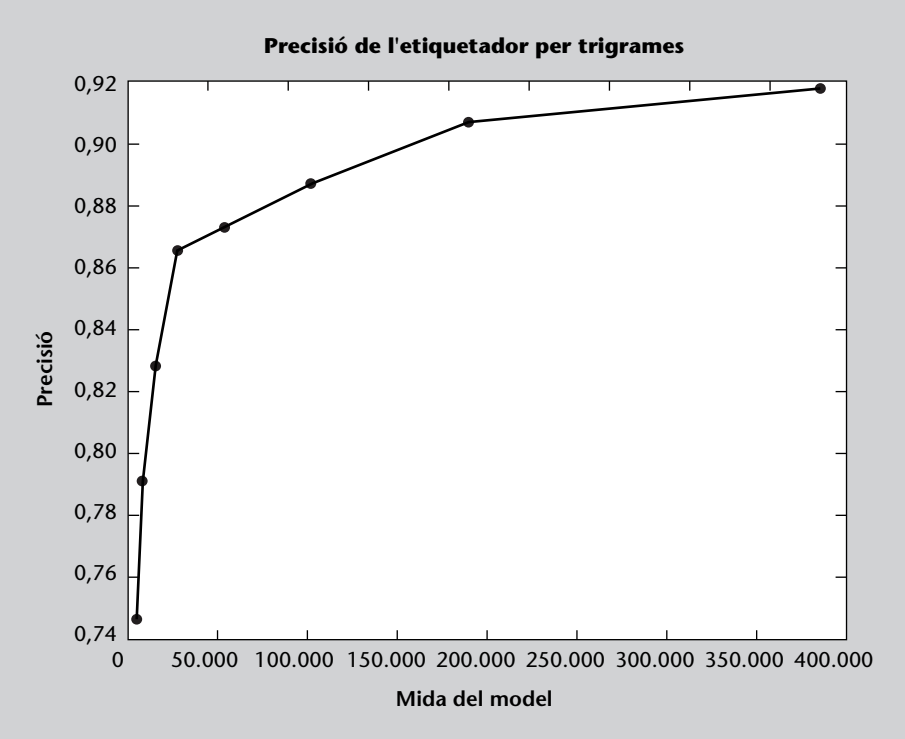
Amb Pylab podem fer gràfics molt més complexos. En el programa següent (`avaluació3.py`) creem un gràfic més complex, on es grafiquen la precisió de diferents etiquetadors per a diferents mides del corpus d'entrenament.

Adreça web recomanada

Podeu trobar tota la informació del paquet Pylab a *http://matplotlib.sourceforge.net/index.html*.

```
import nltk
from nltk.corpus import cess_cat

def precisio(mida):
    train_sents=cess_cat.tagged_sents()[:mida]
    test_sents=cess_cat.tagged_sents()[mida:(mida+1000)]
    t_defecte=nltk.DefaultTagger('ncms000')
    t_sufixos=nltk.AffixTagger(train_sents, affix_length=-3, min_stem_length=2)
```

Figura 6. Resultat de l'avaluació de l'etiquetador per trigrames



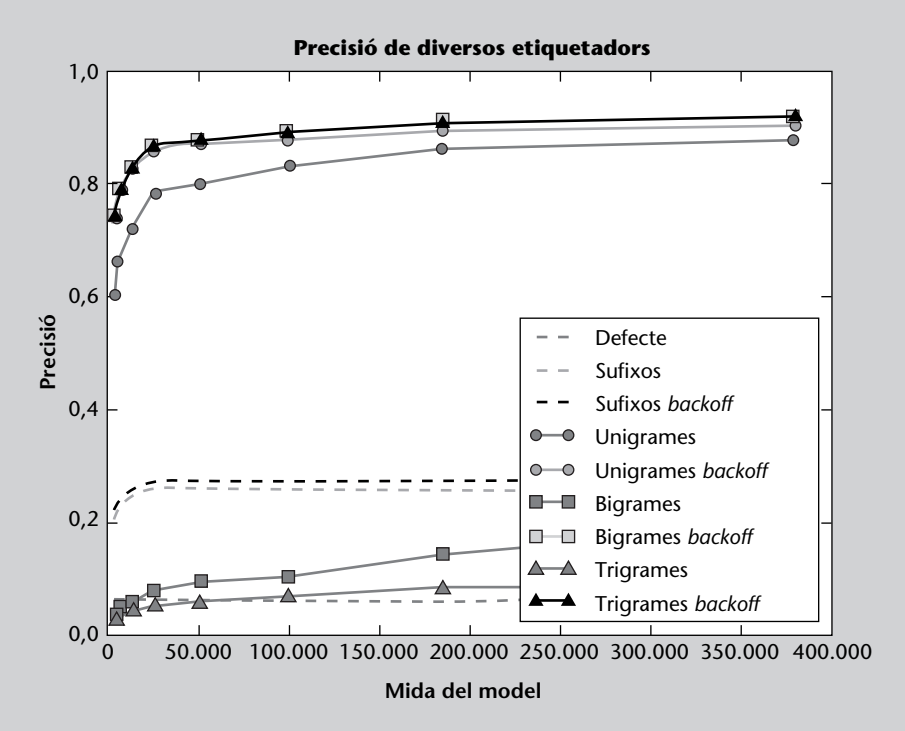

```
    t_sufixos_b=nltk.AffixTagger(train_sents, affix_length=-3,
       min_stem_length=2,backoff=t_defecte)
    t_unigrames=nltk.UnigramTagger(train_sents)
    t_unigrames_b=nltk.UnigramTagger(train_sents, backoff=t_sufixos_b)
    t_bigrames=nltk.BigramTagger(train_sents)
    t_bigrames_b=nltk.BigramTagger(train_sents, backoff=t_unigrames_b)
    t_trigrames=nltk.TrigramTagger(train_sents)
    t_trigrames_b=nltk.TrigramTagger(train_sents, backoff=t_bigrames_b)
    precisio_d=t_defecte.evaluate(test_sents)
    precisio_s=t_sufixos.evaluate(test_sents)
    precisio_sb=t_sufixos_b.evaluate(test_sents)
    precisio_1=t_unigrames.evaluate(test_sents)
    precisio_1b=t_unigrames_b.evaluate(test_sents)
    precisio_2=t_bigrames.evaluate(test_sents)
    precisio_2b=t_bigrames_b.evaluate(test_sents)
    precisio_3=t_trigrames.evaluate(test_sents)
    precisio_3b=t_trigrames_b.evaluate(test_sents)
    print precisio_d, precisio_s, precisio_sb, precisio_1, precisio_1b,
      precisio_2, precisio_2b, precisio_3, precisio_3b
    return precisio_d, precisio_s, precisio_sb, precisio_1, precisio_1b,
      precisio_2, precisio_2b, precisio_3, precisio_3b

def nombre_paraules(mida):
    sents=cess_cat.sents()[:mida]
    paraules=0
    for frase in sents:
```

```
        paraules+=len(frase)
    return paraules


def display():
    import pylab
    mides = 100*(2 ** pylab.arange(8))
    print mides
    mides_paraules= [nombre_paraules(mida) for mida in mides]
    print mides_paraules

    graficar=[]
    precisions = [precisio(mida) for mida in mides]
    g0=[]
    g1=[]
    g2=[]
    g3=[]
    g4=[]
    g5=[]
    g6=[]
    g7=[]
    g8=[]
    for x in range(9):
        for y in range(8):
            if (x==0): g0.append(precisions[y][x])
            if (x==1): g1.append(precisions[y][x])
            if (x==2): g2.append(precisions[y][x])
            if (x==3): g3.append(precisions[y][x])
            if (x==4): g4.append(precisions[y][x])
            if (x==5): g5.append(precisions[y][x])
            if (x==6): g6.append(precisions[y][x])
            if (x==7): g7.append(precisions[y][x])
            if (x==8): g8.append(precisions[y][x])
    print g0
    print g1
    print g2
    print g3
    print g4
    print g5
    print g6
    print g7
    print g8
    pylab.plot(mides_paraules,g0,'r--')
    pylab.plot(mides_paraules,g1,'g--')
    pylab.plot(mides_paraules,g2,'b--')
    pylab.plot(mides_paraules,g3,'-ro')
    pylab.plot(mides_paraules,g4,'-go')
```

```
    pylab.plot(mides_paraules,g5,'-rs')
    pylab.plot(mides_paraules,g6,'-gs')
    pylab.plot(mides_paraules,g7,'-r^')
    pylab.plot(mides_paraules,g8,'-b^')
    pylab.title('Precisio diversos etiquetadors')
    pylab.xlabel('Mida del model')
    pylab.ylabel('Precisio')
    leg = pylab.legend(('defecte', 'sufixos', 'sufixos backoff', 'unigrames',
        'unigrames backoff', 'bigrames', 'bigrames backoff', 'trigrames',
        'trigrames backoff'), 'lower right', shadow=True)

    pylab.show()

display()
```

A la figura 7 podem observar els resultats.

Figura 7. Resultat de l'avaluació de diversos etiquetadors



9. Alguns etiquetadors disponibles

A Internet podem trobar una sèrie d'etiquetadors que estan disponibles sota diferents llicències, algunes lliures. En aquest apartat en citem alguns i detallem la metodologia bàsica que fan servir.

Nota

Pot resultar un exercici interessant descarregar algun d'aquests etiquetadors i intentar executar-los en el vostre ordinador.

9.1. Freeling

Freeling (Carreras i altres, 2004) és un analitzador lingüístic desenvolupat per la Universitat Politècnica de Catalunya. Treballarem molt a fons aquest analitzador en el mòdul "Eines i recursos per al català i castellà" i aprendrem a instal·lar-lo i a fer diferents tipus d'anàlisis.

Adreça web

Es pot trobar tota la informació sobre Freeling a *http://www.lsi.upc.edu/~nlp/freeling/*.

9.2. Tree Tagger

El TreeTagger és la implementació de l'etiquetador per arbres de decisió que vam veure en el subapartat 7.1. Aquest etiquetador es pot descarregar de l'adreça *http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/* i es pot fer servir lliurement per a docència i recerca, però no per a tasques comercials. D'aquesta plana es poden descarregar els programes d'entrenament i d'etiquetatge i els fitxers de paràmetres per a anglès, alemany, italià, holandès, castellà, grec, búlgar i rus. A la plana indicada hi ha instruccions detallades de com cal instal·lar els programes sota diversos sistemes operatius.

Per a Linux és molt senzill; només cal seguir els passos següents:

1) Descarregar l'etiquetador corresponent a la vostra arquitectura, segurament PC, a un directori que us vagi bé.

2) Descarregar els *scripts* d'etiquetatge al mateix director.

3) Descarregar l'*script* d'instal·lació.

4) Descarregar els paràmetres per a les llengües desitjades.

Un cop fet això cal anar al directori corresponent i executar l'*script* d'instal·lació:

```
sh install-tagger.sh
```

Per a executar-lo es pot fer:

```
echo "Hello, good moring" | cmd/tree-tagger-english
reading parameters ...
tagging ...
Hello UH Hello
,,,
good JJ good
moring NN <unknown>
 finished.
```

Si volem analitzar el text d'un arxiu podem fer:

```
cat russian.txt | cmd/tree-tagger-russian
reading parameters ...
tagging ...
Слушало Vmis-sna-p слушать
 ,,,
слышало Vmis-sna-p слыш
ать
вещее Afpnsnf вещий
мое P--nsn мой
все P---pn все
эти P---pn этот
речи Ncfsgn речь
еще R еще
за Sp-a за
месяц Ncmsan месяц
! SENT !
```

La instal·lació en Windows no és complicada. Es descarrega la versió per Windows i es descomprimeix l'arxiu *zip*. Si es vol incloure l'etiquetador en el *path* de l'ordinador (és a dir, que es pugui executar des de qualsevol directori), seguiu les instruccions detallades que us donen a la web. Si l'únic que voleu és provar-lo i fer-hi un ús esporàdic, podeu fer el següent:

1) Un cop descomprimit l'arxiu copieu tota la carpeta `TreeTagger` al directori arrel C:.

2) Si no teniu un intèrpret de Perl al vostre ordinador, instal·leu-lo.

3) Descarregueu els paràmetres de les llengües desitjades i descomprimiu-lo preferentment al directori `C:\TreeTagger\lib`.

Adreça web recomanada

Podeu obtenir gratuïtament un intèrpret de Perl de la pàgina *http://www.activestate.com*.

Per a etiquetar un text primer haurem de tenir-lo en un fitxer que tingui un *token* per línia. Si tenim un fitxer que es diu `text.txt` i que conté:

```
Hello
,
good
morning
```

I des d'una pantalla del símbol de sistema fem (des del directori on hi ha el fitxer):

```
C:\TreeTagger\bin\tree-tagger.exe C:\TreeTagger\lib\english.par
   text.txt -token -lemma
```

Obtindrem l'anàlisi desitjada. Si tenim un fitxer de text sense tokenitzar podem tokenitzar-lo amb el programa `tokenize.pl`. Si el fitxer `text.txt` conté "Hello, good morning" sense tokenitzar podem fer:

```
C:\TreeTagger\cmd\tokenize.pl < text.txt > text2.txt
```

i en `text2.txt` tindrem la versió tokenitzada. Ara ja podem analitzar aquesta versió com hem explicat. Això ho podem fer en un únic pas fent:

```
perl C:\TreeTagger\cmd\tokenize.pl < text.txt | C:\TreeTagger\bin\tree-tagger.exe
C:\TreeTagger\lib\english.par -token -lemma
```

Per a aquells usuaris poc acostumats a fer servir la línia de comandes s'ha creat una interfície gràfica que en facilita l'ús i que podeu veure-la a la figura 8.

Adreça web recomanada

La interfície gràfica de Tree Tagger es pot descarregar de la plana *http://www.smo.uhi.ac.uk/~oduibhin/oideasra/interfaces/winttinterface.htm*. A la mateixa plana de descàrrega hi ha instruccions per a instal·lar aquesta interfície gràfica.

9.3. SVMTool

SVMTool és un etiquetador que es pot descarregar des de l'adreça web següent: *http://www.lsi.upc.edu/~nlp/SVMTool/*. En aquesta plana trobarem els programes necessaris (hi ha diferents versions) i els models per al català, castellà i anglès. Nosaltres hem descarregat la versió 1.2.2 (Perl) i els models per al català. Les instruccions que mostro aquí funcionen bé per a un sistema operatiu Linux, tot i que és possible executar l'SVMTool sota Windows.

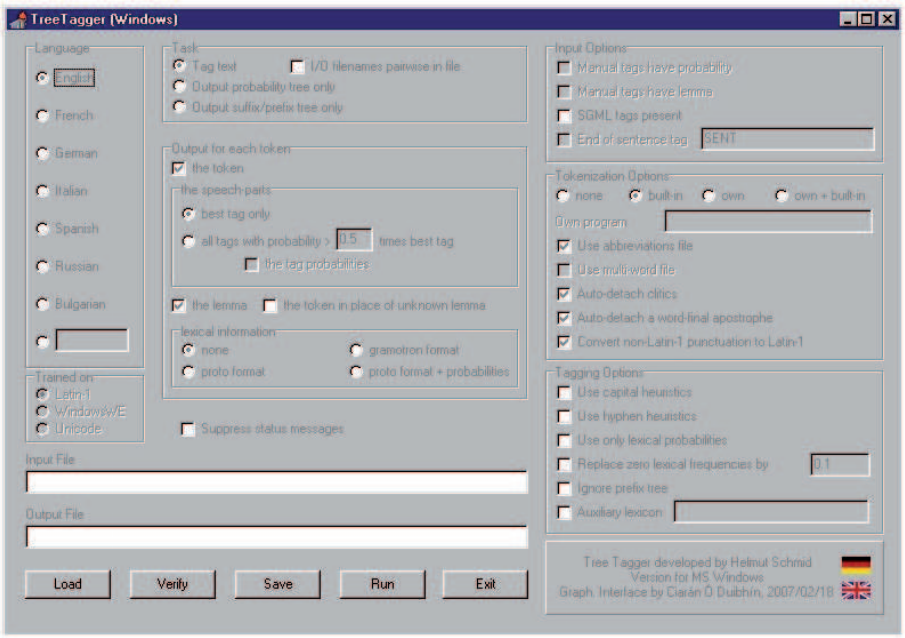
A més de l'etiquetador també es distribueixen tots els programes necessaris per a entrenar-lo.

1) Descarreguem SVMTool.

2) Descomprimim i instal·lem fent:

```
gunzip SVMTool.v1.2.2.tar.gz
tar -xvf SVMTool.v1.2.2.tar
```

Figura 8. Interfície gràfica del Tree Tagger per a Windows




```
cd SVMTool-1.2.2/
perl Makefile.pl
make
sudo make install
```

3) En el subdirectori `bin` hi haurà el programa d'anàlisi.

4) Ara cal descarregar els models corresponents a les llengües que desitgem, i els descomprimim.

5) Per a poder executar l'analitzador necessitem disposar del text per analitzar en un format d'una paraula per línia.

```
Hola
,
avui
fa
un
dia
molt
bonic
,
però
diuen
que
demà
plourà
.
```

6) Si fem (des del subdirectori `bin`, i donant la ruta al model `../../cat/3LB.CAT`):

```
perl SVMTagger -V 2 ../../cat/3LB.CAT < text.txt > sortida.txt
-------------------------------------------------------------------------------
SVMTool v1.2.2. Copyright (C) 2004 Jesus Gimenez and Lluis Marquez.
-------------------------------------------------------------------------------
MODEL = ../../cat/3LB.CAT
T = 0 :: S = LR :: K = 0 :: U = 0
-------------------------------------------------------------------------------
READING DICTIONARY <../../cat/3LB.CAT.DICT>...
[DONE]
READING MODELS < DIRECTION = left-to-right :: MODEL = ambiguous context >
(1) READING MODELS (weights and biases)
    FOR KNOWN WORDS <../../cat/3LB.CAT.M0.LR.MRG>...
(2) READING MODELS (weights and biases)
    FOR UNKNOWN WORDS <../../cat/3LB.CAT.UNK.M0.LR.MRG>...
TAGGING < DIRECTION = left-to-right >
.1 sentences [DONE]
BENCHMARK TIME:  0 wallclock secs ( 0.02 usr +  0.00 sys =  0.02 CPU)
===============================================================================
      START-UP: 1.81 secs
===============================================================================
       TAGGING: 0.02 secs
-------------------------------------------------------------------------------
  F.EXTRACTION: 0 secs
           SVM: 0.01 secs
       PROCESS: 0.01 secs
===============================================================================
OVERALL_TIME = START-UP_TIME + TAGGING_TIME = 1.81 secs + 0.02 secs = 1.83 secs
===============================================================================
```

7) En el fitxer `sortida.txt` tindrem l'anàlisi:

```
Hola NP
, Fc
avui RG
fa VMI
un DI
dia NC
molt RG
bonic AQ
, Fc
però AQ
diuen VMI
que CS
demà VMS
plourà AQ
. Fp
```

Resum

En aquest mòdul hem après què és, per a què serveix i com podem construir etiquetadors morfosintàctics. També hem vist com es pot aprendre a desambiguar morfosintàcticament a partir de corpus anotats i com es poden avaluar els etiquetadors. Hi ha moltes més tècniques a part de les presentades en aquest mòdul, i podeu trobar interessant llegir algun dels articles que us proposo per a les tècniques següents:

- *n*-grames (Weischedel i altres, 1993)
- Models ocults de Markov (Charniak i altres, 1993)
- Model de màxima entropia (Ratnaparkhi, 1996)
- Aprenentatge basat en memòria (Daelemans i altres, 1996)

També hem vist que les paraules desconegudes representen un problema important per als etiquetadors i hem vist diferents tècniques per a tractar aquest problema. Qui vulgui aprofundir en aquestes tècniques pot llegir:

- Sufixos i context proper (Thede, 1998)
- Terminacions i capitalització (Weischedel i altres, 1993)
- Interpolació lineal d'un model de sufixos de mida fixa (Brants, 2000), (Dzeroski i altres, 2000)
- Mesures de similitud paradigmàtica (Cucerzan i Yarowsky, 2000)
- Mètodes basats en regles (Mikheev, 1997)
- Arbres de decisió (Orphanos i Christodoulakis, 1999)
- Màquines de vectors de suport (Nakagawa i altres, 2001)

Bibliografia

Alsina, A.; Badia, T.; Boleda, G.; Bott, S.; Gil, À.; Quixal, M. i Valentín, O. (2002). «CATCG: a general purpose parsing tool applied». A: «Proceedings of the International Conference on Language Resources and Evaluation», volum III (pàgs. 1130–1134).

Bird, S.; Klein, E. i Loper, E. (2009). *Natural Language Processing with Python - Analyzing text with the Natural Language Toolkit*. O'Reilly Media.

Brants, T. (2000). «TnT - A Statistical Part-of-Speech Tagger». A: «Proceedings of the 6th Applied NLP Conference», (pàgs. 224–231).

Brill, E. (1995). «Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging». A: *Computational Linguistics*, volum 21(4): pàgs. 543–565.

Brill, E. i Wu, J. (1998). «Classifier Combination for Improved Lexical Disambiguation». A: «Proceedings of the joint 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL», (pàgs. 191–195). Montréal, Canada.

Calberger, J. i Viggo, K. (1999). «Implementing an Efficient Part-of-Speech Tagger». A: *Software-Practice and Experience*, volum 29: pàgs. 815–832.

Carreras, X.; Chao, I.; Padró, L. i Padró, M. (2004). «FreeLing: An Open-Source Suite of Language Analyzers». A: «Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)».

Charniak, E.; Hendrickson, C.; Jacobson, N. i Perkowitz, M. (1993). «Equations for Part-of-Speech Tagging». A: «Proceedings of the 11th National Conference on Artificial Intelligence», (pàgs. 784–789).

Cucerzan, S. i Yarowsky, D. (2000). «Language Independent, Minimally Supervised Induction of Lexical Probabilities». A: «Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)», (pàgs. 270–277).

Daelemans, W.; Zavrel, J.; Berck, P. i Gillis, S. (1996). «MBT: A Memory-Based Part of Speech Tagger-Generator». A: «Proceedings of the 4th Workshop on Very Large Corpora», (pàgs. 14–27). Copenhagen, Denmark.

Dzeroski, S.; Erjavec, T. i Zavrel, J. (2000). «Morphosyntactic Tagging of Slovene: Evaluating Taggers and Tagsets». A: «Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC)», (pàgs. 1099–1104).

Giménez, J. i Màrquez, L. (2004). «SVMTool: A general POS tagger generator based on Support Vector Machines». A: «Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)», Lisbon, Portugal.

Greene, B. i Rubin, G. (1971). *Automatic Grammatica Tagging of English*. Providence: Brown University.

Halteren, H.; Zavrel, J. i Daelemans, W. (1998). «Improving Data Driven Wordclass Tagging by System Combination». A: «Proceedings of the joint 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL», (pàgs. 491–497). Montréal, Canada.

Karlsson, F.; Voutilainen, A.; Heikkilä, J. i Anttila, A. (1995). *Constraint Grammar. A Language-Independent System for Parsing Unresticted Text*. Berlin / New York: Mauton de Gruyter.

Klein, S. i Simmons, R. (1963). «A computational approach to gramamtical coding of English words». A: *Journal of the ACM*, volum 10(3): pàgs. 334–347.

Màrquez, L. (2001). «Aprendizaje automático y procesamiento del lenguaje antural». Ponència en el curs: "La Ingeniería Lingüística en la Sociedad de la Información". Fundación Duques de Soria (http://www.lsi.upc.es/ lluism/drafts/ml-nlp.ps.gz).

Màrquez, L.; Padró, L. i Rodríguez, H. (1998). «Improving Tagging Accuracy by Using Voting Taggers». A: «Proceedings of the 2nd Conference on Natural Language Processing and Industrial Applications, NLP+IA / TAL+AI», (pàgs. 149–155). New Brunswick, Canada.

Màrquez, L.; Padró, L. i Rodríguez, H. (2001). *Mètodes robustos en l'anàlisi del llenguatge. El processament de text no restringit*, capítol 3. Universitat Oberta de Catalunya.

Màrquez, L. i Rodríguez, H. (1998). *Part-of-Speech Tagging Using Decision Trees*. Lecture Notes on Computer Science. Berlin / heidelberg: Springer.

Mikheev, A. (1997). «Automatic Rule Induction for Unknown-Word Guessing». A: *Computational Linguistics*, volum 23(3): pàgs. 405–423.

Nakagawa, T.; Kudoh, T. i Matsumoto, Y. (2001). «Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines». A: «Proceedings of the 6th Natural Language Processing Pacific Rim Symposium», (pàgs. 325–331).

Orphanos, G. i Christodoulakis, D. (1999). «POS Disambiguation and Unknown Word Guessing with Decision Trees». A: «Proceedings of the 9th Conference of European Chapter of the Association for Computational Linguistics (EACL)», (pàgs. 134–141).

Ratnaparkhi, A. (1996). «A Maximum Entropy Model for Part-of-Speech Tagging». A: «Proceedings of Conference on Empirical Methods in Natural Language Processing», (pàgs. 133–142).

Schmid, H. (1994). «Probabilistic Part-of-Speech Tagging Using Decision Trees». A: «Proceedings of th Conference on New Methods in Language Processing», Manchester.

Shannon, C. (1948). «A mathematical theory of communication». A: *Bell System Technical Journal*, volum 27: pàgs. 398–403, 623–656.

Thede, S. (1998). «Predicting Part-of-Speech Information about Unknown Words using Statistical Methods». A: «Proceedings of the joint 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, COLING-ACL», (pàgs. 1505–1507).

Voutilainen, A. i Heikkilä, J. (1993). «An English Constraint Grammar (ENGCG): a surface-sytntactic parser of English».

Weischedel, R.; Meeter, M.; Schwartz, R.; Ramshaw, L. i Palmucci, J. (1993). «Coping with ambiguity and unknown words through probabilistic models». A: *Computational Linguistics*, volum 19(2): pàgs. 359–382.