



link.nltk

un workbench linguistico su Plone

Giovanni Toffoli

Incontro presso IASI-LEKS del 2 Luglio 2010

Che cosa è link.nltk

link.nltk è un'estensione per Plone che include:

- **un workbench per creare, raffinare e valutare risorse linguistiche**
- **funzioni utili alla analisi, indicizzazione, classificazione, annotazione dei contenuti testuali del CMS.**

link.nltk integra:

- **la libreria Python NLTK (= Natural Language ToolKit), anch'essa open source**
- **risorse linguistiche per l'Italiano (corpora, lessici, ecc.), un po' meno open ma disponibili a certe condizioni.**

Incontro presso IASI-LEKS del 2 Luglio 2010

Prerequisiti di link.nltk

link.nltk richiede:

- **Plone versione 3.1+**
(in realtà è sviluppato e testato con Plone 3.3.5)
- **la libreria NLTK (www.nltk.org),**
completa delle risorse linguistiche campione contenute nell'appendice `nltk_data`
- **alcune estensioni di Plone,**
appartenenti alla suite `z3c` (= Zope 3 collective)
- in particolare `z3c.forms` - usate per costruire la UI.

Le risorse linguistiche di link.nltk

Le risorse linguistiche attualmente usate in link.nltk sono:

- **itWaC, un corpus "bilanciato", estratto dal web con una metodologia ben documentata, che specifica i criteri di esplorazione del web e di campionamento dei documenti**
- **morph-it, un lessico morfologico di libero dominio, sviluppato dal gruppo di Baroni e Zanchetta**
- **regole di tokenizzazione assiemate da me a partire da un più ampio repertorio tratto da una nota didattica di Baroni**
- **regole per il chunking di frasi nominali, definite da me; sono da migliorare (estendere e raffinare).**

itWaC e fratelli

itWaC è stato costruito recentemente da 2 team dell'Università di Trento-Rovereto (Baroni) e Bologna-Forlì (Zanchetta).

itWaC è distribuito sotto forma di 21 file di testo, in formato pseudo-XML, per un totale di:

- **circa 30 GByte di testo annotato**
- **oltre 1,5 miliardi di token (parole); ogni token è corredato di POS-tag e lemma.**

itWaC è "fratello" di ukWaC e deWaC, corpora sviluppati all'interno dell'iniziativa "The WaCky Wide Web" - WaCky in breve - dove il suffisso WaC sta per "Web as a Corpus".

Incontro presso IASI-LEKS del 2 Luglio 2010

I moduli di elaborazione di link.nltk

Attualmente link.nltk include:

- moduli di elaborazione, che estendono le API di NLTK e/o le specializzano per supportare le risorse linguistiche italiane
- moduli di interazione, che espongono le funzioni del workbench

I moduli di elaborazione attualmente sono 5:

- corpora, lexicons
- tokenizers, taggers, chunkers.

L'interfaccia utente (UI) del workbench include le funzioni:

- corpus, lexicon
- tokenizer, postagger, termextractor

Incontro presso IASI-LEKS del 2 Luglio 2010

L'interfaccia utente di link.nltk

In linea di principio i moduli di elaborazione dovrebbero essere parametrici rispetto a

- **lingua, formato del corpus, tipo di tagger**
- **canale di input per il corpus, il lessico, le regole di tokenizzazione, l'istanza di tagger pre-addestra**

Il workbench dovrebbe riconfigurare dinamicamente la sua UI in base ai parametri via via impostati. Per esempio

- **se si fissa la lingua (es: Italiano) ..**
- **.. dovrebbero sparire le opzioni non consistenti con tale scelta: i corpora non italiani, i lessici non italiani, i tipi di tagger che eventualmente non supportassero l'Italiano, ecc.**

Il lessico

Il lessico morph-it è interessante perché

- è un lessico morfologico molto ampio
- è di libero dominio
- analizzando un lessico, si estrae il tagset
- analizzando un lessico, si estrae l'insieme dei lemmi
- analizzando un lessico, si estrae l'insieme dei tag associati ad ogni parola distinta.

Ma le frequenze si possono estrarre solo da un corpus.

Quindi un lessico è utile, mi sembra, più a fini didattici che pratici.

I corpora

I corpora sono la materia prima alla base di tutto.

Da un corpus si può ricavare un lessico.

Da un corpus si può estrarre quel che serve a disambiguare le parole (token) e a identificare i chunk di interesse:

- **le probabilità assolute dei tag**
- **le probabilità a priori dei tag associabili ad una data parola**
- **le probabilità condizionate di tutti i tipi immaginabili, per lo più con riferimento ad un contesto.**

I corpora in NLTK

Le risorse linguistiche di NLTK, contenute nell'appendice `nltk_data`, coprono in modo sparso numerose lingue.

Alcuni corpora contengono testi "grezzi", alcuni sono annotati con POS-tag, pochi sono "parsed", molti sono classificati per genere, argomento, ecc.

NLTK definisce delle classi che

- **implementano tutti i tipi di corpora contenuti in `nltk_data`**
- **hanno metodi per estrarre da un corpus: liste di file, di token, di tagged token, di sentences, di tagged sentences; a volte tali liste possono essere filtrate per genere, argomento, ecc.**

NLTK fornisce anche classi "astratte" che possono essere specializzate per trattare nuovi formati di corpus.

Incontro presso IASI-LEKS del 2 Luglio 2010

I corpora in NLTK (continua)

Una feature molto interessante delle classi di cui sopra è che la maggior parte dei metodi restituiscono non delle liste di oggetti, ma degli "iterable", cioè delle liste *virtuali*.

Vediamo che significa da un esempio; supponiamo che

- esista un metodo `tokens()` che mi riporta tutti i token (1,5 miliardi !!) del corpus `itWaC`
- io scriva il codice Python, contenente l'operatore di "slicing":

```
tt = itwac.tokens(); tt50 = tt[50:100]
```

Se il metodo `tokens()` mi riportasse una vera e propria lista, sul mio notebook dovrei attendere circa 1 ora.

Invece il corpus viene letto solo fino a dove serve, e così il codice di cui sopra richiede meno di 1 secondo.

I corpora dell'iniziativa WaCky

In `link.nltk` ho creato un paio di classi, derivate dalle classi astratte di NLTK, che mi consentono di leggere e analizzare tutti i corpora del progetto WaCky.

Per ora ho sperimentato con il primo dei 21 file di itWaC:

- circa 1,5 MByte di testo annotato in formato pseudo-XML
- 3.334.016 frasi
- 93.150.779 token:
è il totale delle parole e dei segni di interpunzione.

I tokenizer

NLTK fornisce bell'e fatti alcuni tokenizer per l'Inglese e alcune altre lingue; non per l'Italiano.

Fornisce inoltre funzioni di utilità per costruire un tokenizer a partire da una sequenza di "regular expression" passate come input.

Con tali funzioni

- **ho creato facilmente una prima versione di tokenizer per l'Italiano ..**
- **.. a partire da un insieme di regular expression che ho trovato in una dispensa di Baroni di alcuni anni fa.**

I tokenizer (continua)

Uno dei maggiori problemi dei tokenizer: riconoscere se un punto (".")

- **rappresenta la fine di una frase, o è un separatore di cifre, decimali, o fa parte di una sigla.**

lo per ora

- **uso un insieme di regular expression ridotto rispetto a Baroni**
- **dato che la maggior parte delle sue serve a riconoscere decine e decine di sigle diverse, italiane o provenienti da altre lingue.**

Altro problema

- **quali euristiche usare per riconoscere i nomi propri?**

I tagger in NLTK

I POS-tagger costituiscono un argomento complesso.

Sono un elemento essenziale nell'estrazione dei termini, sia quando si costruisce una terminologia, sia quando la si usa per classificare un documento o estrarne informazione.

NLTK fornisce

- **linee guida per la costruzione di tagger basati sull'apprendimento da corpora**
- **alcune classi astratte da cui partire per creare tagger specifici di tipo "sequenziale", che nel fare il tagging di un testo lo scandiscono da sinistra a destra, considerando ad ogni passo una piccola "finestra" intorno alla parola da annotare.**

Addestrare un tagger con itWaC - primi passi

La funzione "postagger" di link.nltk consente di

- creare un nuovo tagger addestrandolo con un corpus
- effettuare download e upload di un tagger addestrato in locale e/o del testo da taggare
- usare un tagger addestrato per eseguire il tagging di un testo - in prospettiva, di un intero corpus.

Ho cominciato con l'addestrare, con il primo file di itWaC, un "UnigramTagger", cioè un tagger che analizza un token alla volta, trascurando il contesto; si basa cioè solo sulla probabilità a priori che una certa parola rappresenti una certa parte del discorso (POS = part of speech).

Addestrare un tagger con itWaC - primi risultati

Il training dello UnigramTagger ha richiesto oltre 1 ora (sul mio notebook), ma i risultati sono incoraggianti: vedi esempio allegato a mia e-mail di qualche giorno fà.

Per la maggior parte gli errori sono dovuti al non uso di informazione e conoscenza sul contesto.

Per esempio

- **il tagger non ha riconosciuto la frase nominale "festa cittadina" perché nel corpus la parola "cittadina" è usata più spesso come nome che non come aggettivo**
- **d'altra parte sappiamo che, se occorre dopo un nome, "cittadina" è presumibilmente un aggettivo.**

Addestrare un tagger con itWaC - problemi

Dopo UnigramTagger, ho tentato di addestrare un BigramTagger.

Un "BigramTagger" è un tagger che, nell'annotare un token, per ciascuno dei tag candidati considera la probabilità condizionale che esso segua il tag con cui è stato GIÀ' annotato il token precedente.

Quindi non cerca di fare un *look-ahead*.

A questo punto il programma, dopo circa 3 ore di elaborazione, è andato in *memory error*: stava usando già da un bel pezzo la memoria virtuale del mio notebook.

Addestrare un tagger con itWaC - soluzioni

Per superare, almeno in parte, il problema di memoria e prestazioni ho sfruttato 2 feature di NLTK:

- lo "slicing" (vi ho accennato a proposito dei corpora)
- il meccanismo di "backoff".

Lo *slicing* si applica alla lettura di un corpus:

- a tentativi individuo span, un range di frasi del corpus, inferiore alla dimensione del primo file, e tale da non superare la memoria disponibile
- con lo span individuato, diciamo, 1 o 2 milioni di frasi, (anziché 3 milioni e oltre) addestro il tagger.

Addestrare un tagger con itWaC - il backoff

Per ogni tipo di tagger, si può specificare un tagger da usare come ripiego (backoff) se il primo non ha informazioni sufficienti per decidere; il meccanismo è ricorsivo:

- un BigramTagger può "ripiegare" su un UnigramTagger
- un UnigramTagger può "ripiegare" su un DefaultTagger che, per l'Italiano, assegnerà il tag NOUN (il più frequente in assoluto) ad una parola mai incontrata nel corpus di training.

Addestrare un tagger con itWaC - esempio

Un esempio: "festa cittadina"

- se la parola "cittadina" è stata incontrata nel corpus dopo parole con tag NOUN, BigramTagger avrà calcolato la relativa probabilità condizionale e potrà fare una scelta
- altrimenti BigramTagger demanderà il tagging a UnigramTagger
- se nel corpus la parola "cittadina" c'è, UnigramTagger gli assegnerà il tag con la maggiore probabilità a priori
- se nel corpus la parola "cittadina" proprio non c'è, a sua volta UnigramTagger demanderà il tagging al DefaultTagger, che gli assegnerà il tag NOUN: in questo caso sbaglia, ma 60 volte su 100 ci azzecca.

I tagger - come continuare?

Tutto il discorso per me è partito dal TreeTagger:

- reimplementare la sua strategia, per poterla meglio controllare
- usare come backoff un tagger basato sui suffissi: per l'Italiano dovrebbe essere molto efficace.

Qualche domanda:

- quanto è difficile reimplementare (emulare) il TreeTagger?
- di quanta memoria hanno bisogno gli algoritmi di training?
- quale può essere la dimensione di un TreeTagger addestrato?
- è proprio conveniente il TreeTagger nel caso dell'Italiano?
- non conviene usare un UnigramTagger o un BigramTagger, e poi applicare delle regole di correzione tipo Brill tagger?

I chunker

Un chunker:

- è un tipo di analizzatore sintattico (parser) con obiettivi ridotti.

Certe tecniche di chunking assomigliano a quelle di tokenizzazione.

NLTK fornisce

- alcuni chunker bell'e fatti per l'Inglese e forse per altre lingue, non per l'Italiano
- funzioni di utilità per costruire un chunker a partire da un insieme ordinato di regole espresse con un formalismo analogo a quello delle "regular expression".

I chunker (continua)

Non dispongo di corpora italiani già analizzati in termini di chunk nominali; allora

- **non sono in grado di creare un chunker mediante training**
- **ho costruito a mano un primo insieme di regole di chunking.**

Funzionano già abbastanza bene; ma mi chiedo:

- **quali tipi di chunk nominali sono più utili per l'estrazione di termini?**
- **preposizioni, articoli, pronomi dimostrativi all'inizio di un chunk vanno eliminati?**
- **quando è che una frase preposizionale va considerata parte integrante del chunk nominale a cui è attaccata?**