Tools for the lab sessions

Ing. Roberto Tedesco, PhD
roberto.tedesco@polimi.it

arcslab
adaptable, relational and cognitive software environments

NLP – AA 17-18
Prof. L. Sbattella

Tools to download

- Text:
  - POS tagger: Stanford
  - Parser: Stanford
  - Chunker: TreeTagger
- Speech:
  - Praat
- Toolkit:
  - NLTK 3 (prerequisite: Python 2.7 / **3.x**)

To be presented today

- Text:
  - POS tagger: Stanford
  - Parser: Stanford
  - Chunker: TreeTagger
- Toolkit:
  - Python (needed as NLTK 3 is a Python library)
- You already knows Praat…

- These slides include a subset of `ES3-POS-chunker-parser.pdf` (the full version is already on the web site, just for the sake of completeness…)

POS TAGGING, PARSING, CHUNKING

Stanford POS tagger

- Stanford POS Tagger
- Entropy Maximization
  - Uses a CMM, basically a simplified CRF
- Java based

- demo

Chunking (aka shallow parsing)

- Identifying and classifying the flat, non-overlapping segments of a sentence
  - This set typically includes noun phrases, verb phrases, adjective phrases, and prepositional phrases
  - [$_{NP}$ The morning flight] [$_{PP}$ from] [$_{NP}$ Denver] [$_{VP}$ has arrived.]
- Leverages POS tagging
- Two approaches:
  - Finite-state rules able to catch phrase segments (FST)
  - Machine learning. We present this approach

Tags (Penn treebank corpus)

Tag	Description	Words	Example	%
NP	noun phrase	**DT+RB+JJ+NN + PR**	*the strange bird*	51
PP	prepositional phrase	**TO+IN**	*in between*	19
VP	verb phrase	**RB+MD+VB**	*was looking*	9
ADVP	adverb phrase	**RB**	*also*	6
ADJP	adjective phrase	**CC+RB+JJ**	*warm and cosy*	3
SBAR	subordinating conjunction	**IN**	*whether or not*	3
PRT	particle	**RP**	*up the stairs*	1
INTJ	interjection	**UH**	*hello*	0

CoNLL corpus

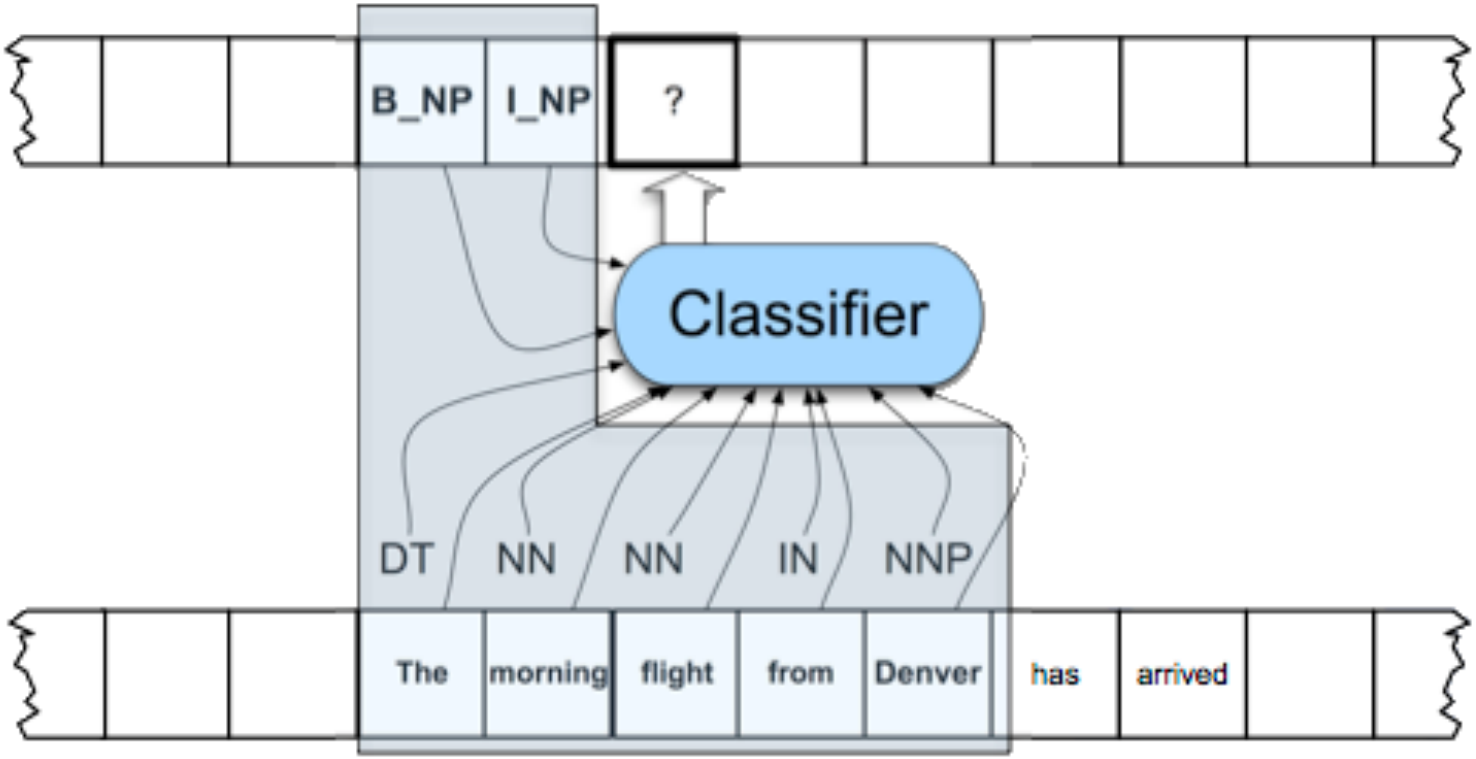
- To train stochastic chunkers
- token, POS, and chunk type
- IBO tagging, for chunk types:

  B_ begin of a chunk
  I_ inside the chunk
  O not part of a chunk

He	PRP	B_NP
reckons	VBZ	B_VP
the	DT	B_NP
current	JJ	I_NP
account	NN	I_NP
deficit	NN	I_NP
will	MD	B_VP
narrow	VB	I_VP
to	TO	B_PP
only	RB	B_NP
#	#	I_NP
1.8	CD	I_NP
billion	CD	I_NP
in	IN	B_PP
September	NNP	B_NP

Machine learning based chunking

- The chunker slides a context window over the sentence classifying words as it proceeds
- At this point the classifier is attempting to label *flights*

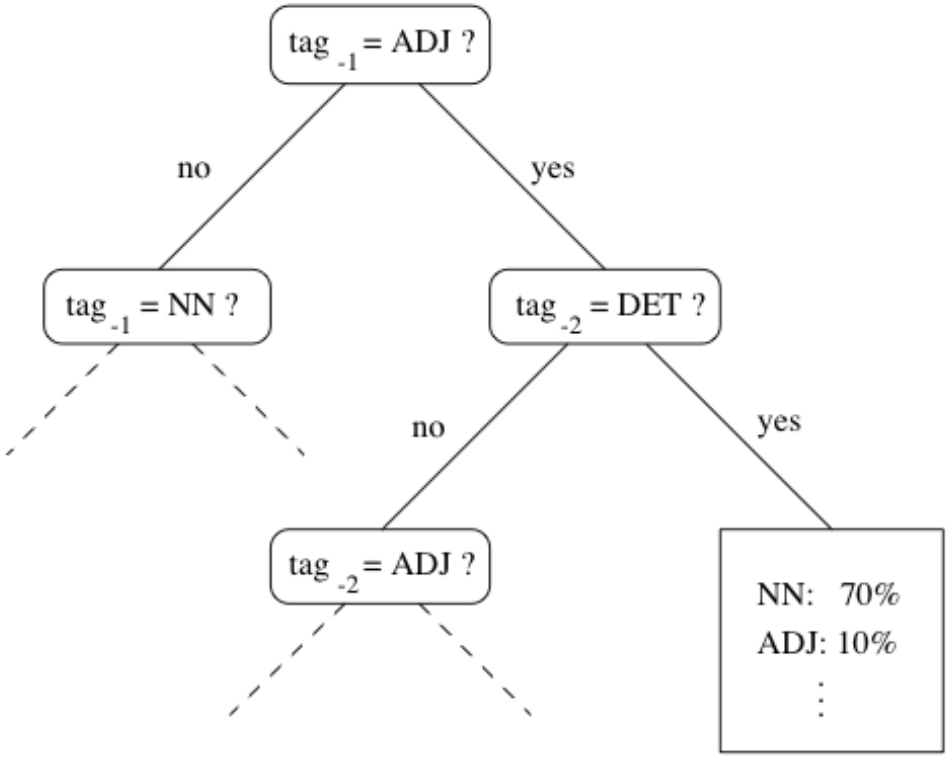


TreeTagger

- POS tagging
- Uses a 2nd order HMM; estimates transition probability by means of a model (not an n-gram)
- Uses a binary decision tree
  - Built from a training corpus of trigrams with POS's

demo

$$p(T^{(t)} \mid T^{(t-1)}, T^{(t-2)})$$

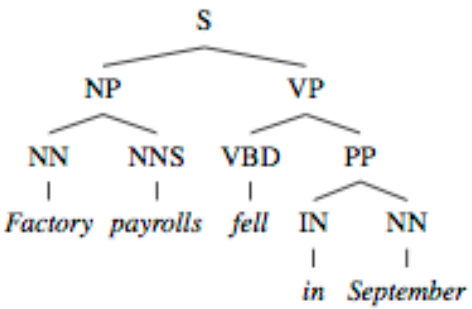


Stanford Parser

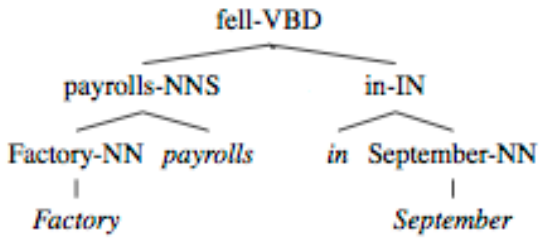
- Lexicalized PCFG are hard to train
  - Data sparsity
- Lexicalized parse tree can be seen as a combination of:
  - Unlexicalized parse tree
  - Dependency tree among words
- These trees can be trained separately
  - Reduce data sparsity

Stanford Parser

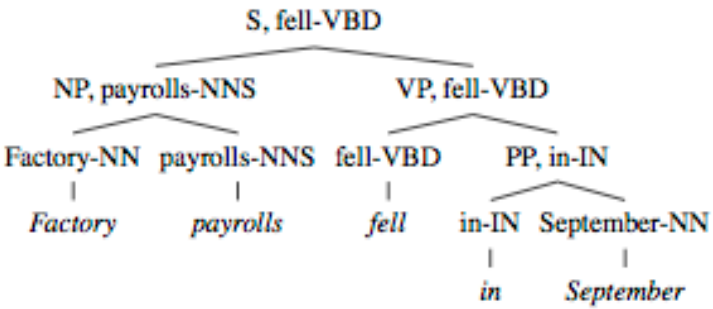
- Starting form a lexicalized Treebank:
  - Reconstruct unlexicalized PCFG
  - Extract headwords and build dependencies among them
- Parsing:
  - P(T): probability of a given unlexicalized parse tree
  - p(D): probability of a given dependency tree (sort of)
- Factored Lexicalized PCFG: p(T,D) = p(T)•p(D)



(a) PCFG Structure



(b) Dependency Structure



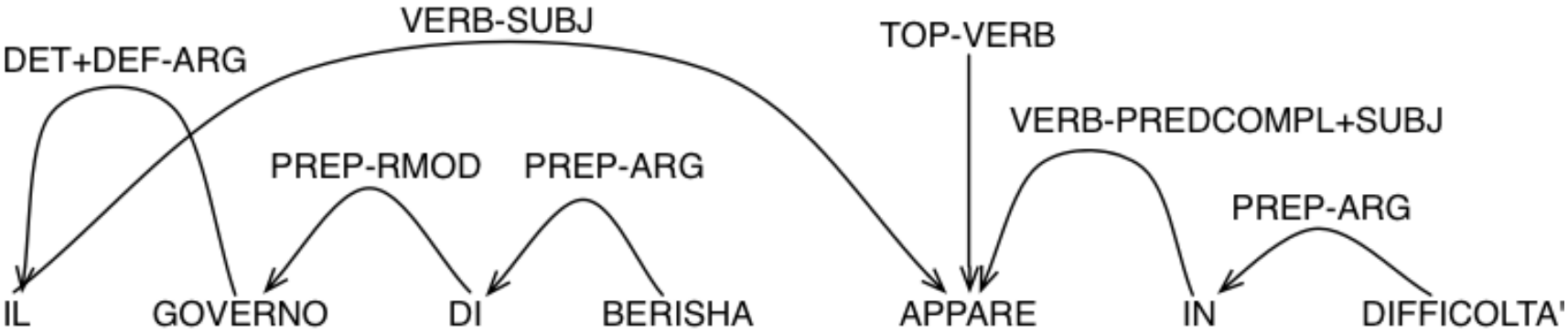
(c) Combined Structure

- Demo

Dependency Grammar: TUT

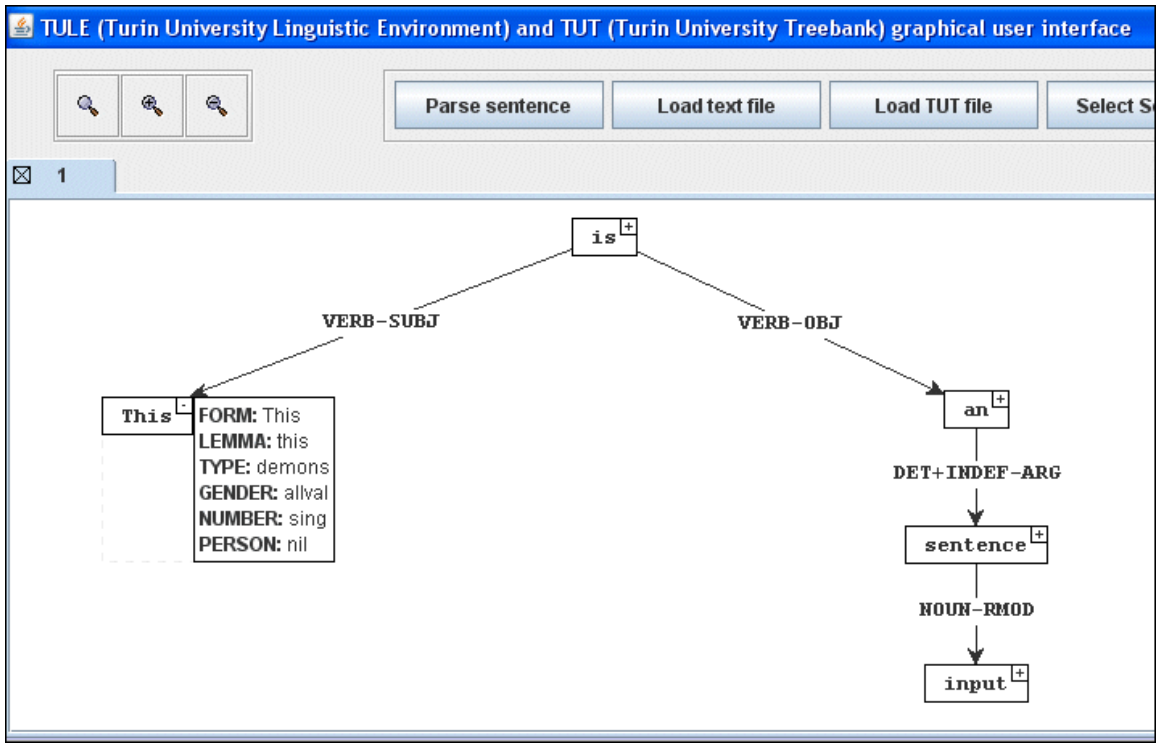
- TUT Treebank contains DG-tagged sentences

1 Il (IL ART DEF M SING) **[5;VERB-SUBJ]**
2 Governo (GOVERNO NOUN COMMON M SING) **[1;DET+DEF-ARG]**
3 di (DI PREP MONO) **[2;PREP-RMOD]**
4 Berisha (BERISHA NOUN PROPER) **[3;PREP-ARG]**
5 appare (APPARIRE VERB MAIN IND PRES INTRANS 3 SING) **[0;TOP-VERB]**
6 in (IN PREP MONO) **[5;VERB-PREDCOMPL+SUBJ]**
7 difficolta' (DIFFICOLTÀ NOUN COMMON F ALLVAL) **[6;PREP-ARG]**
8 . (#\. PUNCT) **[5;END]**



TULE

- Dependency grammar
- Client-server
  - Server: LISP-based parser
  - Client: Java-based GUI
- Multilanguage



REFERENCES

POS Tagging

- TreeTagger:
  - http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html
- Stanford
  - http://nlp.stanford.edu/software/index.shtml
- FreeLing (and parser, morpho analyzer, …)
  - http://nlp.lsi.upc.edu/freeling/

Shallow parsers

- CHAOS
  - http://ai-nlp.info.uniroma2.it/external/chaosproject/
- TreeTagger:
  - http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html
- Illinois
  - http://cogcomp.cs.illinois.edu/page/software_view/13

Full parsers

- Stanford parser
  - http://nlp.stanford.edu/software/lex-parser.shtml
- Charniak parser
  - http://www.cs.brown.edu/~ec/
- NLTK (actually, contains a lot of tools…)
  - http://www.nltk.org
- TULE
  - http://www.tule.di.unito.it/

Corpora/1

- Linguistic Data Consortium
  - http://www.ldc.upenn.edu
- European Language Resources Association (ELRA)
  - http://www.icp.grenet.fr/ELRA/
- Int. Computer Archive of Modern English (ICAME)
  - http://nora.hd.uib.no/icame.html
- Oxford Text Archive (OTA)
  - http://ota.ahds.ac.uk/
- Child Language Data Exchange System (CHILDES)
  - http://childes.psy.cmu.edu/

Corpora/2

- NLTK_lite (directory **`corpora`**)
  - Small samples of: Penn Treebank, Brown, ecc.
- Penn Treebank:
  - http://www.cis.upenn.edu/~treebank/
- Brown Corpus:
  - http://en.wikipedia.org/wiki/Brown_Corpus
- American National Corpus:
  - http://americannationalcorpus.org/
- British National Corpus:
  - http://www.natcorp.ox.ac.uk/
- Corpus e Lessico di Frequenza dell'Italiano Scritto:
  - http://alphalinguistica.sns.it/CoLFIS/CoLFIS_Presentazione.htm

A NANO INTRO TO PYTHON

Brief description of Python progs

- Official web site: `https://www.python.org`
- Object oriented
- Typed objects but un-typed variables
- Type constraints are not checked at compile time
- Whitespace indentation
  - An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block
  - Spaces or tab, but not the two at the same time
  - The “continuation operator”: ‘\’
- Functions and methods

Some Python hints

- String literals

```
a= “ciao”
a = 'ciao'
a = “ciao\n”
a = “”” ciao this is
an example with new lines “””
```

- No `main()`: code is executed from the beginning
- But, if functions are defined:

```
if __name__ == '__main__':
    my_entry_point()
```

  launches the function `my_entry_point()`
- Otherwise, the file is just a library of functions

Some Python hints

- `If` expression (as the "C" `x = cond ? a : b`)

```
a = idx - 10 if idx >= 10 else 0
```

- Tuples, lists, sets, dictionaries (i.e., hashtables)

```
t = ()
t = ('ciao', 1, 2.343)
l = []
l = ['a','f', 'a']
s = set()
s = {'a','f','a'}
```

no duplicates→ `{'a','f'}`

```
d = {}
d = d['pippo'] = 3
```

contains→ `{'pippo': 3}`

- Formatted output

```
print ("Example:%f3.2;%2d" % (59.058,1))
```

prints: `Example: 59.06; 1`

Some Python hints

- List comprehension

  $S = [f(x) : x \in X, \text{condition}(x) \text{ holds}]$

```
s = [v for v in 'ABCDABCD' if v not in 'CB']
print(s)  # generates ['A', 'D', 'A', 'D']
```

- Set comprehension

  $S = \{f(x) : x \in X, \text{condition}(x) \text{ holds}\}$

```
s = {v for v in 'ABCDABCD' if v not in 'CB'}
print(s)  # generates ['A', 'D']
```

- Dictionary comprehension

```
d = {n: n**2 for n in range(5)}
```

  variable `d` contains:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Some Python hints

- For.. in

```
list1 = [5, 6, 8, 3]
for a in list1:
    print(a)
```

- List comprehensions generates all the element
  - Memory consumption
- Generators return an object that will only generate the items when needed

```
doubles_list = [2 * n for n in range(100000)]
doubles_gen = (2 * n for n in range(100000))
for a in doubles_list:
    print(a)
for a in doubles_gen:
    print(a)
```

Some Python hints

- Functions with optional return

```
def name(parameter1, parameter2):
    .......
    return xyz
```

- Optional arguments (i.e.: with default values) and named argument

```
def open_file(host,filename="default.txt"):
    .......

open_file("111.111.111.111")
open_file("111.111.111.111","pippo.txt")
open_file(filename="pippo.txt", \
          host="111.111.111.111")
```

Some Python hints

- Some NLTK functions and methods make use of the Python `*args` and `**kwargs` idioms to allow arbitrary number of arguments to functions and methods
- The `*args` will give you all parameters as a tuple:

```
def foo(*args):
    for a in args:
        print(a)

>>> foo(1,2)
1
2
```

Some Python hints

- The `**kwargs` will give you all keyword arguments, except for those corresponding to a formal parameter, as a dictionary:

```
def foo(**kwargs):
    for a in kwargs:
        print(a, kwargs[a])

>>> foo(name='one', age=27)
age 27
name one
```

Some Python hints

- Classes

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def length(self):
        a=math.sqrt(self.x**2+self.y**2)
        return a
```

- New objects

```
v = Vector(3,5)
```

- Object methods

```
l = v.length()
```

- Class methods

```
Path.cwd()
```

Integer division

- In Python 2.7, the `/` operator performs integer division if inputs are integers
- If you want float division just use this special import at the beginning of the file:

```
from __future__ import division
```

- For Python 3.x, this is not required, as the `/` operator performs float division in any case

Character encoding

- Python 3.x works with Unicode by default
- Python 2.7 provides some support to Unicode, but it is not the default way of coding characters
- For details, see:
  `https://docs.python.org/3/howto/unicode.html`

NLTK

- NLTK, is a suite of libraries and programs for both symbolic and statistical NLP
- Based on the Python programming language
- It provides:
  - interfaces to over 50 corpora and lexical resources
  - a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning
  - wrappers for industrial-strength NLP libraries
- Available for Windows, OS X, and Linux

NLTK installation

- Current version: NLTK 3
- Install Python (version 2.7 or 3.x)
- Install NLTK:
  see: http://www.nltk.org/install.html
- Install NLTK Data:
  see: http://www.nltk.org/data.html
- The API:
  see: http://www.nltk.org/api/nltk.html
- The on-line book (updated for NLTK3, Python3):
  see: http://www.nltk.org/book/