NLP Chain

Giuseppe Castellucci
castellucci@ing.uniroma2.it
Web Mining & Retrieval a.a. 2013/2014

Outline

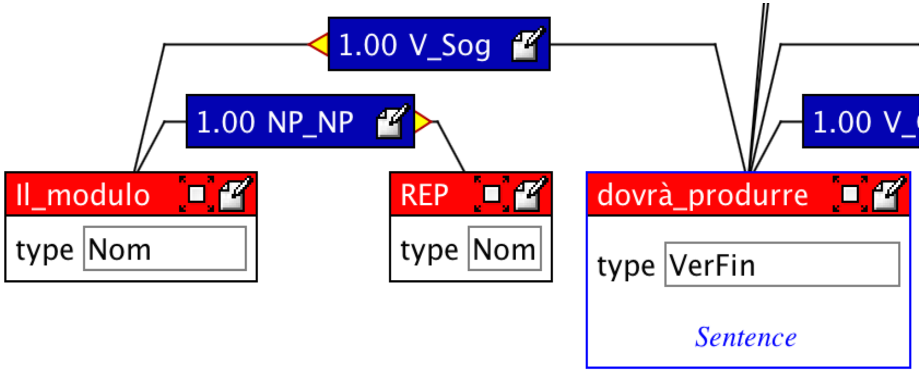
- NLP chains
- RevNLT
- Exercise

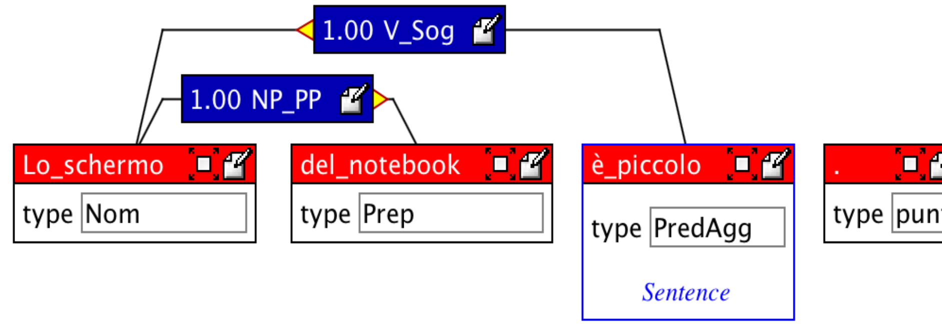
NLP chain

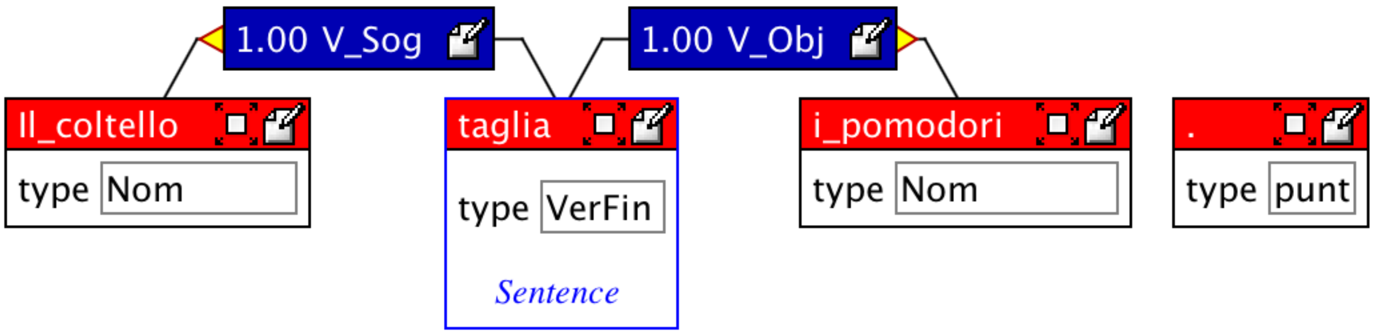
- Automatic analysis of texts
- At different levels
  - Token
  - Morphological
  - Grammatical
  - Syntax
  - Semantics
- Chain
  - reflects these levels in different processing modules

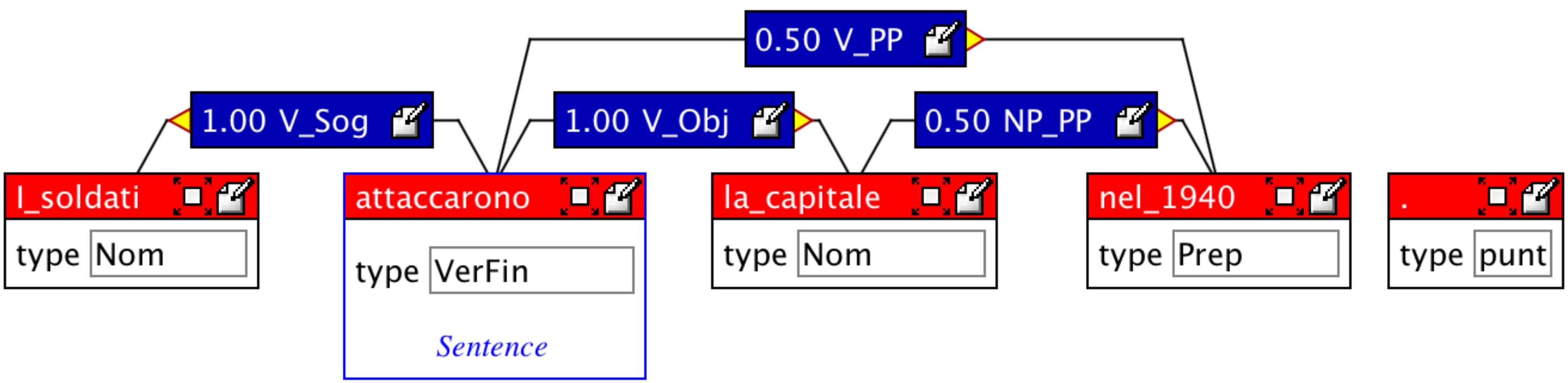
What is useful for?

- Find **patterns**
  - Verb-Noun, Noun-Adjective, etc…
  - Applications in (not limited to)
    - Requisite analysis
    - Object-Action-Tool discovery
- **Extract** information for ML algorithms
  - In higher level tasks linguistic features are very useful
    - Search, Semantic Role Labeling, Sentiment Analysis


1.00 V_Sog
1.00 NP_NP
1.00 V_
Il_modulo
type Nom
REP
type Nom
dovrà_produrre
type VerFin
Sentence

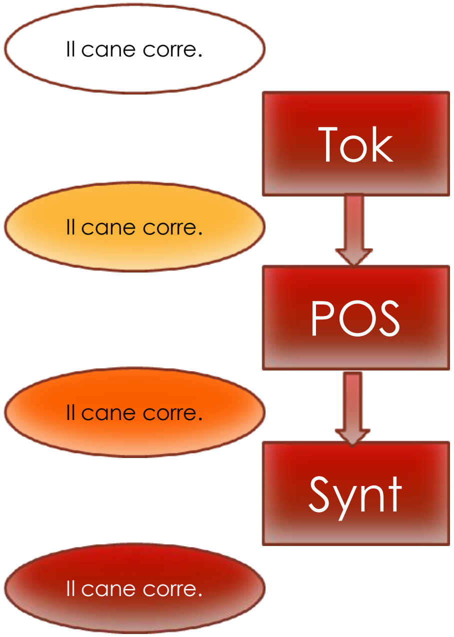

1.00 V_Sog
1.00 NP_PP
Lo_schermo
type Nom
del_notebook
type Prep
è_piccolo
type PredAgg
Sentence
.
type


1.00 V_Sog
1.00 V_Obj
Il_coltello
type Nom
taglia
type VerFin
Sentence
i_pomodori
type Nom
.
type punt


0.50 V_PP
1.00 V_Sog
1.00 V_Obj
0.50 NP_PP
I_soldati
type Nom
attaccarono
type VerFin
Sentence
la_capitale
type Nom
nel_1940
type Prep
.
type punt

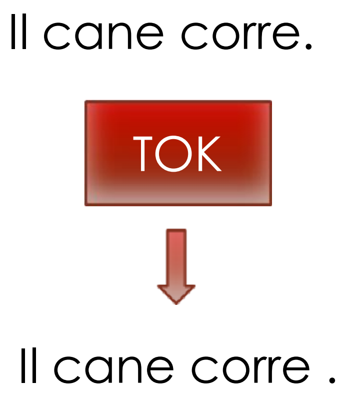
General flow

- Most general case
  - One Linguistic Level – One Module
- Each module takes in input a data structure
- And enrich it with new information
- Advantages
  - Problem decomposition
  - Each module is responsible of one step
  - Can use previously computed information
- Disadvantages
  - Any idea?
  - Error propagation
  - No joint inferences



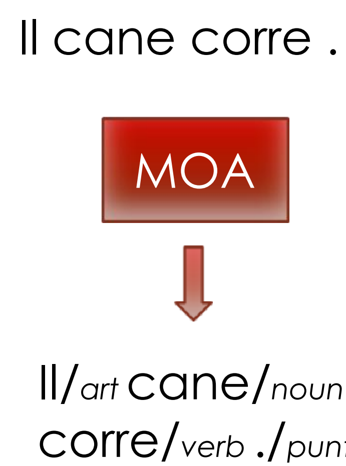
TOKenizer

- Input
  - A text
  - A paragraph
  - A sentence
- Output
  - List of tokens
- Each token represents the minimal unit for further processing
- It is not a whitespace splitter!
  - Dates
  - Numbers
  - Abbreviations
  - In web texts: emoticons, links etc.



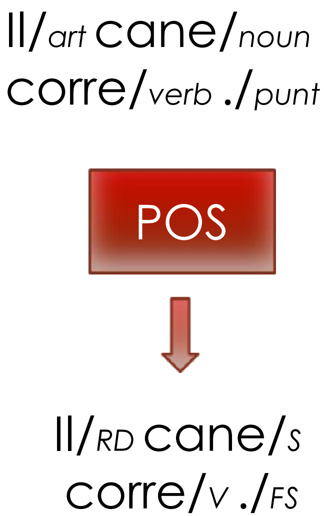
MOrphology Analyzer

- Input
  - Tokenized string
- Output
  - Enrich tokens with morphological information
    - Simplest case using a dictionary
  - Grammatical category, grammatical features
    - *Abbassiamo* is *verb 1.pers.plur.ind.pres*
- A token can have multiple morphological information
  - E.g. *diviso* is an *adjective* or a *verb*



POS tagger

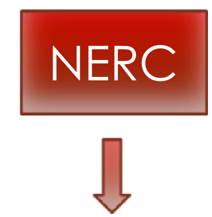
- Part-Of-Speech: linguistic category of words
- Input
  - Morphologically analyzed tokens
- Output
  - Assign to each token its grammatical category
  - Morphological information used as features
  - Context-dependent models



Named Entity Recognizer…

- … and Classifier
- Input
  - Tokens with morphological and grammatical information
- Output
  - Enriched tokens where a **named entity** is recognized and classified w.r.t. predefined classes
  - PERsons, ORGanizations, LOCations

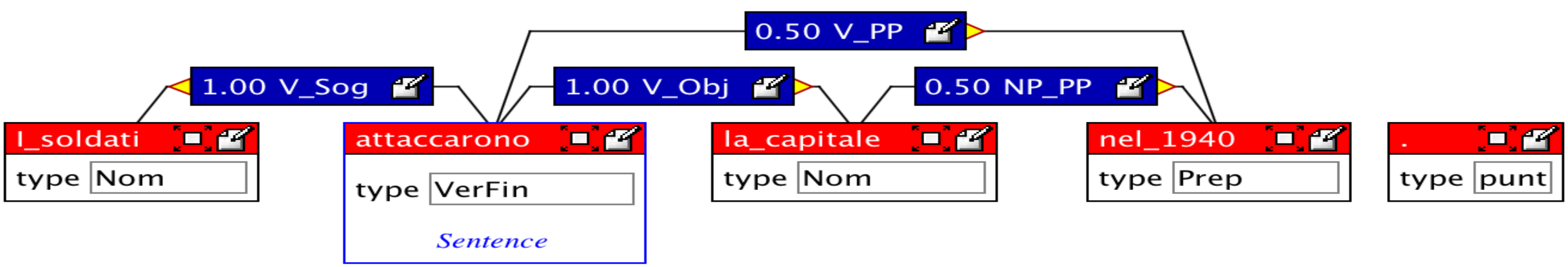
Giuseppe/SP ieri/B era/V a/E Roma/SP ./FS



Giuseppe/SP-PER ieri/B era/V a/E Roma/SP-LOC ./FS

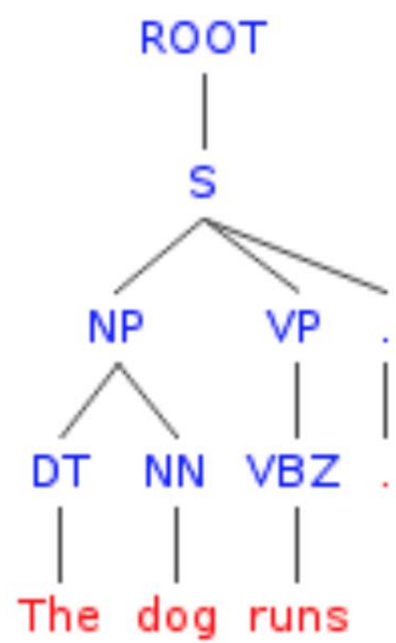
Parser

- Recognize the grammatical structure of sentences
- Highlight relations between words
- Two formalisms
  - Constituency based parser
  - Dependency based parser
- Details in next lectures, here just a quick look.



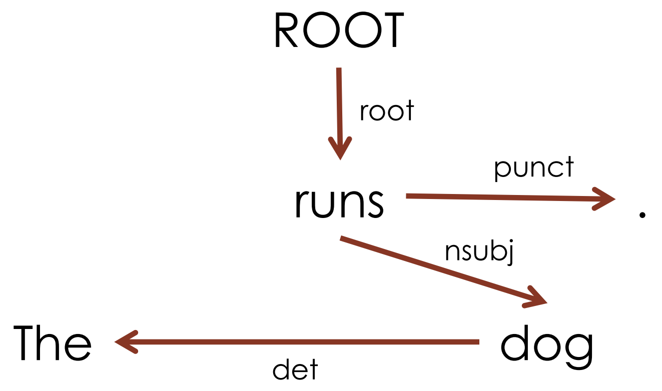
Constituent

- A constituency parse tree breaks a text into sub-phrases
  - Non-terminals in the tree are types of phrases
  - Terminals are the words in the sentence
  - Edges are unlabeled
- Context-free grammar formalisms
- Statistical methods
  - Probabilistic CFG



Dependency

- A dependency parse connects **words** according to their **relationships**
  - Each vertex in the tree represents a word
  - Child are words that are dependent on the parent
  - Edges are labeled by the relationship
- Again, Statistical methods are used



Example: Stanford Parser

- State-of-the-art parser
  - (http://nlp.stanford.edu/software/corenlp.shtml)
- Widely used in research
- Simple and intuitive java interface
  - Bindings also for other programming languages
- It includes a classical chain
  - With the idea of annotator
  - Include constituent and dependency based parsers
- Contains semantic level annotators
  - E.g. sentiment annotator

Stanford Parser

- Can be integrated as a maven dependency in your projects
  - Package *stanford-corenlp*

```
Properties props = new Properties();
props.put("annotators", "tokenize, ssplit, pos, lemma, parse");
StanfordCoreNLP pipeline = new StanfordCoreNLP();
Annotation document = new Annotation("The dog runs over the grass.");
pipeline.annotate(document);
List<CoreMap> sentences = document.get(SentencesAnnotation.class);
CoreMap sentence = sentences.get(0);
  for (CoreLabel token: sentence.get(TokensAnnotation.class)) {
     String surface = token.originalText();
     int start = token.beginPosition();
     int end = token.endPosition();
     String postag = token.tag();
     String lemma = token.lemma();
  }
// HERE ARE THE DEPENDENCIES
SemanticGraph dependencies = sentence.get(BasicDependenciesAnnotation.class);
```

Example: Open NLP

- Machine learning based toolkit for the processing of natural language text
  - Apache foundation software
  - https://opennlp.apache.org
- It supports
  - Tokenization
  - Sentence splitting
  - Part of speech tagging
  - Named entity recognition
  - Chunking
  - Parsing
  - Co-reference resolution
- Java interface

Example: RevNLT

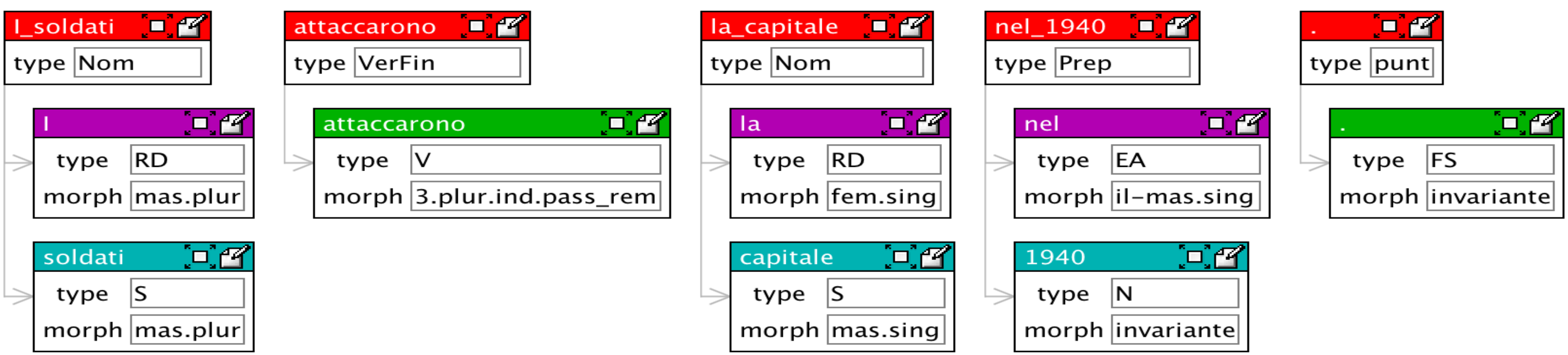
- A natural language toolkit for Italian and English
- Developed initially at the University of Roma Tor Vergata by the ART group
- It is based on the eXtended Dependency Graph formalism
- An incubator of research ideas
- That can be potentially used in production environments

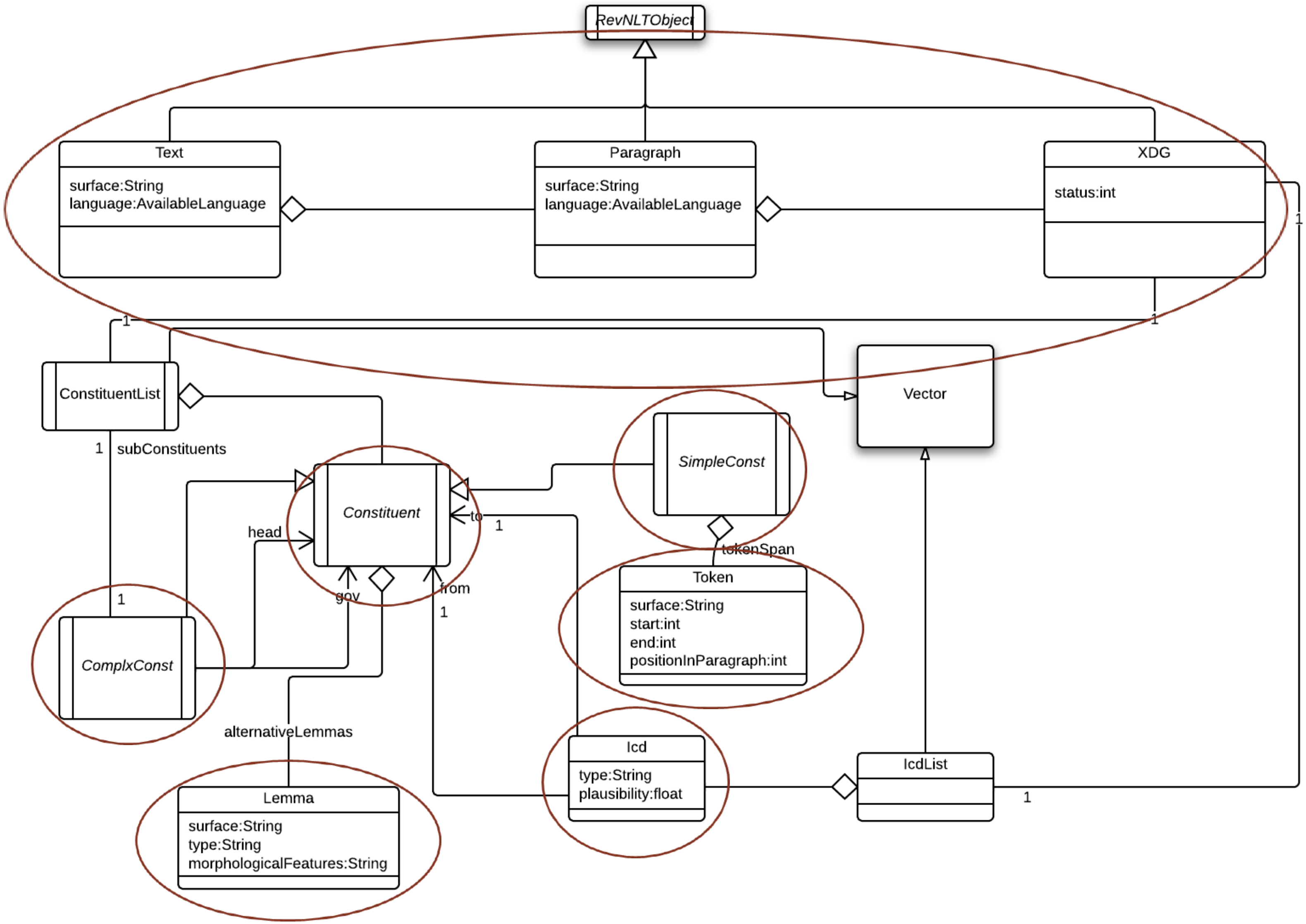
RevNLT

- It includes
  - Tokenizer
  - Morphology analyzer
  - POS Tagger
  - Named Entity Recognizer and Classifier
  - Chunker
  - Dependency Parser
- XDG is the common data structure
- Client/Server interface
- Graphical User Interface

Chunker

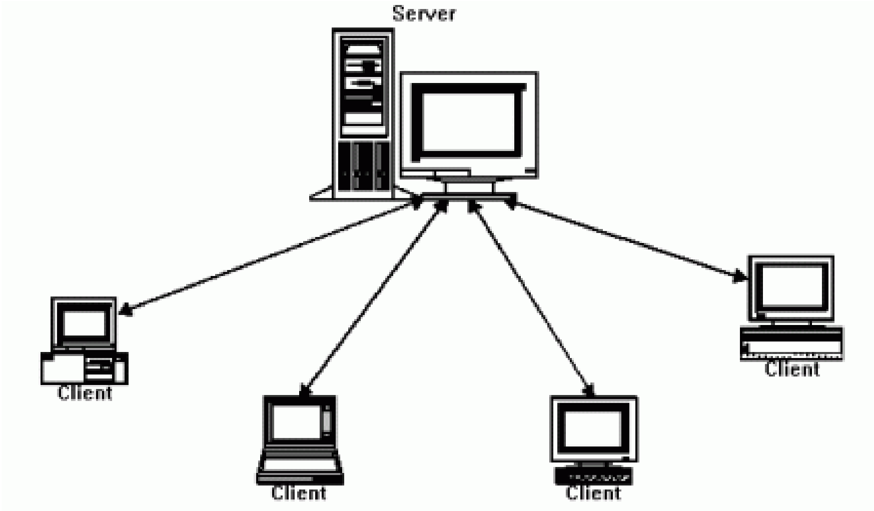
- Chunking can be seen as partial parsing
- Assigns a partial syntactic structure to a sentence
  - flatter structures
  - only deals with “chunks”
  - chunks are typically subsequences of constituents
- more efficient, robust and often deterministic




RevNLTObject
Text
surface:String
language:AvailableLanguage
Paragraph
surface:String
language:AvailableLanguage
XDG
status:int
1
1
1
ConstituentList
Vector
1
subConstituents
SimpleConst
Constituent
to
1
head
tokenSpan
Token
surface:String
start:int
end:int
positionInParagraph:int
1
gov
from
1
ComplxConst
alternativeLemmas
lcd
type:String
plausibility:float
IcdList
1
Lemma
surface:String
type:String
morphologicalFeatures:String

Client/Server architecture

- Support for a client/server architecture in Java
- Natural language processing is computational expensive
- NLP as a service



API

- Simple Java API to process texts
- Both local and client/server
- Today we'll use the client/server API to process texts

Data Structure API

- A *Text* is the main data structure
- Initialize a Text without paragraph information
- The system will split paragraphs, and then sentences.

```
Text t = new Text("The service was good.”, AvailableLanguage.en);
```

Data Structure API

- A *Text* is the main data structure
- Initialize a Text with a paragraph
- The system will not split paragraphs, but it will split sentences

```
Text t = new Text();
t.setLanguage(AvailableLanguage.en);
t.setParagraphs(new Vector<Paragraph>());
t.getParagraphs().add(new Paragraph("The
service was good."));
```

Data Structure API

- Retrieve useful information

```
for (Paragraph p : t.getParagraphs()) {
   for (XDG x : p.getXdgs()) {
      ConstituentList l = x.getConstituents();
      for (Constituent c : l) {
         SimpleConst s = (SimpleConst) c;
         String lemma = s.getFirstLemma().getSurfaceWithoutQuotes();
         String postag = s.getType();
         String surface = s.getSurfaceWithoutQuotes();
         System.out.println(surface + " "+ postag + " "+ lemma);
      }
   }
}
```

Simple constituents of a sentence

Lemma surface

Part Of Speech

Original Surface

Data Structure API

- Save a processed Text in XML for further use

```
t.save("myText.xml");
```

- Load a previously processed Text

```
Text loadedText = Text.load("myText.xml");
```

Client API

- How can we process a text?

```
Client c = new Client(”address", port);
System.out.println(c.askLanguage().toString());
// Text t = c.parseText(“The service is good.”, “TOK,MOA,POS,PMF”);
Text t= c.parseText(t, "TOK,MOA,POS,PMF");
```

Simple Exercise (10 min)

- Produce a text file in which each line is composed by
  - Original surface
  - POS tag
  - Lemma surface
- Use: *address* = 160.80.24.16, *port* = 4005
- All separated by a ***tab*** char, all sentences are separated by a blank line
- Use the file *inClassSentences.txt* as input file
  - Iterate over each line and process the file through the client interface
  - `classes(space separated)TABsentence`
- Produced file should be **identical** to *triColumnInClass.txt*

Example code

address = 160.80.24.16, *port* = 4005

```
Client c = new Client(”address", port);
System.out.println(c.askLanguage().toString());
// Text t = c.parseText(“The service is good.”, “TOK,MOA,POS,PMF”);
Text t= c.parseText(t, "TOK,MOA,POS,PMF");

for (Paragraph p : t.getParagraphs()) {
   for (XDG x : p.getXdgs()) {
      ConstituentList l = x.getConstituents();
      for (Constituent c : l) {
         SimpleConst s = (SimpleConst) c;
         String lemma = s.getFirstLemma().getSurfaceWithoutQuotes();
         String postag = s.getType();
         String surface = s.getSurfaceWithoutQuotes();
         System.out.println(surface + “ ”+ postag + “ ”+ lemma);
      }
   }
}
```

Evaluation

- How to evaluate NLP systems?
- It depends on the linguistic level we are interested in
- Different evaluations for
  - POS Tagging
  - Named entities
  - Parsing
- See two examples

Evaluating POS Tagging

- In general: accuracy
- Count the correct assigned Part Of Speech to each token
- Different evaluations for known words and unknown words
  - Verify algorithms generalization capability
  - Generate a list of known words given the training set
  - At test time a word not in this list is an unknown word

Evaluating Named Entities

- An entity based evaluation
  - Evaluate entities, not tokens!
- [Giuseppe Rossi]$_{PER}$ non è stato convocato da [Cesare Prandelli]$_{PER}$.
- Here, two entities
- The evaluation must consider a true positive as a correctly recognized **entity**
  - Giuseppe Rossi non è stato convocato da [Cesare Prandelli]$_{PER}$.
  - Here,
    - Precision = 1
    - Recall = 0.5
    - F1 = 0.66

Exercise

- Construct a dataset for svmlight
  - +1 1:1 50:1 55:1
- Topic classification task of sentences in the restaurant domain
  - A sub task of 2014 Aspect Based Sentiment Analysis task
  - Given a sentence classify it with respect to topic classes
  - Classes are not mutually exclusive

food service ambience Excellent atmosphere, delicious dishes good and friendly service.

How to model the task

- Let's think about the model
  - Can we model topics with bag of words?
  - And distributional vectors?
  - And Tree Kernels?
- In class, keep it simple,
  - Bag-of-words or bag-of-lemmapos
- Multiple classes -> multiple classifiers
- How to manage multiple classifiers and multiple (not mutually exclusive) classes?
  - Ideas?
  - Use the margin (score > 0) of the classifier to decide if an example belongs to a class

General algorithm for generating training files

- Analyze sentences with RevNLT
- For each class D
  - Produce a file training{D}.txt containing examples in which positive (+1) are the one of class {D} and others are negative (-1)
  - For each example produce a feature representation (bag-of-word or bag-of-lemmapos)
  - Produce also files for development and test

General algorithm for generating training files

- How to generate vectors?
  - Use a dictionary!
  - Associate to each words or lemma and pos pair a number
  - It will be its feature number
  - Use a boolean feature value
  - Write your dictionary on a file!

Tuning

- For each parameter c *(1,2,5,10)*
  - For each class *D*
    - Train a model model_c_D.model
    - Classify development with model_c_D.model
  - Choose classes $D_1,\ldots,D_k$ with margin > 0
  - Evaluate performances with respect to the oracle
- Performance is the F1 measure of correctly recognized classes
- Choose the parameter c that optimize the performance

Training

- Train |D| models with the best parameter found
- Classify test sentences
- Evaluate your system on test
- This is your final performance

Exercise

- In WMR_1314_nlpexercise.zip
  - trainingSentences.txt
  - developmentSentences.txt
  - testSentences.txt
  - TopicClassification.java: starter code in an Eclipse project
- In class,
  - Generate vector files using the starter code
- At home,
  - Perform a proper tuning phase of the C parameter
  - Perform testing and report via mail performances with a bag-of-words model and a bag-of-lemmapos model (also your tuning runs)

Hint

- Call the C svmlight executable via Java
- It can speed up (a lot) your coding time!

```
// parameters
float c = 1.0f;
// input file
String train_file="train.txt;
// output file
String modelFile = "model.tmp";
// your executable
String learnExecutable = "svm_learn"
//prepare your process
ProcessBuilder ps = new ProcessBuilder(learnExecutable, "-c", c, train_file, modelFile);
// start the process
Process p = ps.start();
// this is if you want to capture the log of the training phase
BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
String line;
PrintWriter logWriter = new PrintWriter(logLearn, "UTF-8");
while ((line = in.readLine()) != null) {
             logWriter.println(line);
}
p.waitFor();
logWriter.flush();
logWriter.close();
```