Natural Language Processing

Open NLP (http://opennlp.apache.org/)

- Java library for processing natural language text
  - Based on Machine Learning tools
    - maximum entropy, perceptron
  - Includes pre-built models for some languages and annotated text resources
  - Is work in progress....
- Supported NLP tasks
  - tokenization
  - sentence segmentation
  - part-of-speech tagging
  - named entity extraction
  - chunking
  - parsing
  - coreference resolution (experimental)

Library structure

- The library provides components to approach specific NLP tasks
  - The components can be combined to build a NLP processing pipeline
  - Each component interface in general has methods for
    - execute the NLP processing task on a given input text stream
    - train a model for the NLP task from examples
    - evaluate a model on test data
  - The component functionalities can be accessed through a Java API or a command line interface (CLI)
    - read the model from file
    - instantiate the model
    - execute the processing task

```
SomeModel model = new SomeModel(
   new FileInputStream("lang-model-name.bin"));
```

```
ToolName toolName = new ToolName(model);
```

```
String output[] = toolName.executeTask(
          "This is a sample text.");
```

CLI command

- The `opennlp` script allows to exploit the available modules

```
OpenNLP 1.5.3. Usage: opennlp TOOL
where TOOL is one of:
  Doccat                          learnable document categorizer
  DoccatTrainer                   trainer for the learnable document categorizer
  DoccatConverter                 converts leipzig data format to native OpenNLP
   format
  DictionaryBuilder               builds a new dictionary
  SimpleTokenizer                 character class tokenizer
  TokenizerME                     learnable tokenizer
  TokenizerTrainer                trainer for the learnable tokenizer
  TokenizerMEEvaluator            evaluator for the learnable tokenizer
  TokenizerCrossValidator         K-fold cross validator for the learnable tokenizer
  ............
  ChunkerME                       learnable chunker
  ChunkerTrainerME                trainer for the learnable chunker
  ......
  Coreferencer                    learnable noun phrase coreferencer
  CoreferencerTrainer
....
```

Sentence detector

- Punctuation is usually used to define sentence boundaries
  - The sentence detector detects if a punctuation character actually marks the end of a sentence

Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.

1 - Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
2 - Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.

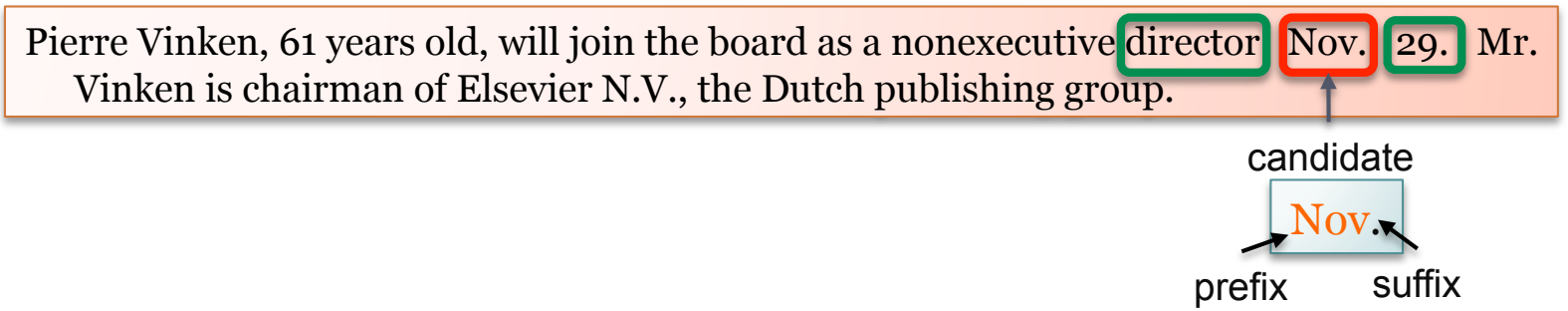
    - Not all the punctuation characters end a sentence. The full stop "." can be used in acronyms, abbreviations or be part of a token (IP addresses, Web URLs)
- Sentence detection is in general performed before tokenization
- Sentences are defined by the use of punctuation characters and not by their contents or graphical layout
  - titles and section headers may not be correctly separated from body

Boundary detection

- Sentence boundary detection is approached as a classification task
  - An occurrence of ".", "?", and "!" is a valid sentence boundary?
    - A list of punctuation marks is not sufficient
    - Full-stop "." has many different uses (decimal point, ellipsis, abbreviations, email and Internet addresses,..)
    - Punctuation can be present in an embedded quotation
    - "!" and "?" are less ambiguous but may appear in proper names (*Yahoo!*) or may be repeated to stress their meaning (*go out!!!*)
  - The classification rules can be learnt from examples
    - Handcrafting the classification rules can be a costly and error prone task
      - Lexically-based rules & exception lists for disambiguation
      - Debugging rules may be difficult due to the presence of interactions among them (e.g. absorption properties: "*The president lives in Washington D.C.*" – not "D.C..")
    - a model can be trained on a corpus annotated with sentence boundaries
      - J.C. Reynar, A. Ratnaparkhi "*A maximum entropy approach to identifying sentence boundaries*" in 5th Conference on Applied NLP, 1997

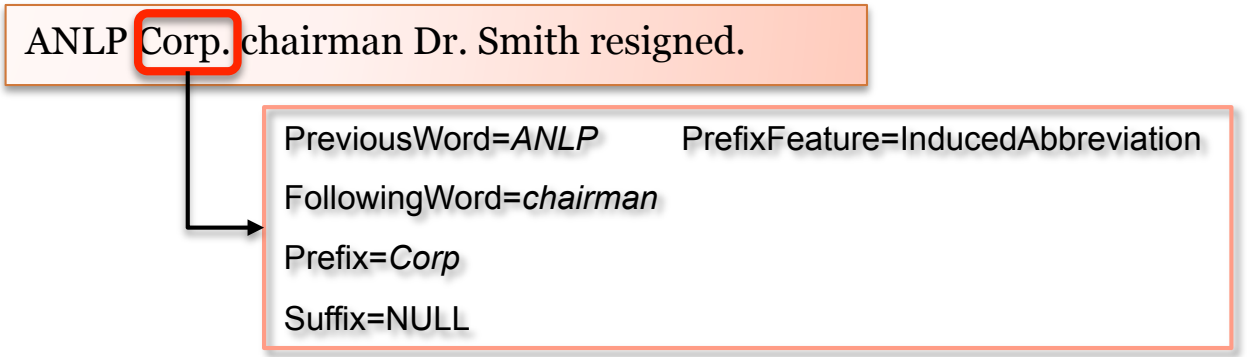
Classifier input features 1

- Potential sentence boundaries are tokens containing "!", "?", "."
  - Classifier inputs (for English financial texts)
    - information on the token containing the potential boundary (Candidate)
      - Prefix and Suffix (parts preceding and following the punctuation mark)
      - Presence of particular characters in the Prefix and Suffix
      - Whether the candidate is an honorific (Mr., Prof., Dr.,..)
      - Whether the candidate is a corporate designator (Corp., Inc., S.p.A,,..)
    - features of the word on the left (right) of the candidate (left and rights contexts)
      - wider contexts do not increase significantly the classifier accuracy



Classifier input features 2

- The list of abbreviations can be built from the training data
  - An abbreviation is a token that contains a "." that is not a sentence boundary in the training corpus
- Classifier inputs
  - The Prefix and Suffix
  - Whether the Prefix or Suffix is in the list of extracted abbreviations
  - The words on the left and right of the Candidate
  - Whether the words in the context are extracted abbreviations



Maximum Entropy classifier

- The use of a boundary token is modeled by the joint probability p(b,c) of its actual role and its context
  - *b* in {true,false} represents if the token is a boundary
  - *c* is the context of the token
- Estimation of p(b,c) from data
  - Text corpora contain information on the cooccurrence of b and c, but not enough to completely specify p(b,c) for all the (b,c) pairs
  - We need a method to exploit a sparse evidence about (b,c) to reliably estimate the probability model
- Maximum Entropy Principle
  - The “correct” distribution p(b,c) maximizes the entropy (uncertainty) subject to the constraints that represent the evidence (known facts)

Maximum Entropy Principle

- If $\mathcal{B}$ is the set of classes (boundary, not boundary) and $\mathcal{C}$ is the set of the contexts, the estimated *p(b,c)* should
  - maximize the entropy

$$H(p) = -\sum_{(b,c)\in\mathcal{B}\times\mathcal{C}} p(b,c)\log p(b,c)$$

  - remain consistent with the evidence

- The representation of the evidence determines the form of p(b,c)
  - The evidence is encoded as k features

Features

- A feature is a binary-valued function on events

$$f_j : \mathcal{B} \times \mathcal{C} \rightarrow \{0, 1\} \quad j = 1, \ldots, k$$

- Features typically represent a co-occurrence relation between the predicted class b and the context
  - for sentence boundary detection an useful feature is

$$f_j(b, c) = \begin{cases} 1 & if\ \text{Prefix}(c) = Mr\ \&\ b = \texttt{false} \\ 0 & otherwise \end{cases}$$

    - a period at the end of the word "*Mr.*" is unlikely to be a sentence boundary
    - the probability $p(\texttt{false}, c)$ will tend to be high if the Prefix is "*Mr.*"
  - the model require to define what features are to be used and not how to use them

Feature constraints

- The constraints on the features have the form

$$E_p[f_j] = E_{\tilde{p}}[f_j]$$

  - $E_p[f_j]$ is the model expectation of f_j
  - $E_{\tilde{p}}[f_j]$ is the observed expectation of f_j

$$E_{\tilde{p}}[f_j] = \sum_{(b,c)\in\mathcal{B}\times\mathcal{C}} \tilde{p}(b,c) f_j(b,c)$$

  - $\tilde{p}(b,c)$ is the probability of (b,c) estimated on the training corpus
  - The assumption that the features are binary-valued reduces the constraints to sum of sets of probabilities (i.e. the probabilities of those events for which the feature is present)

p(b,c) for MaxEnt

- Optimal expression for the joint probability distribution

$$p^*(b,c) = \pi \prod_{j=1}^{k} \alpha_j^{f_j(b,c)}$$

  - π is a normalization constant
  - $\alpha_j,\ j = 1,\ldots,k$ are the model parameters, each weighing the corresponding feature, that are estimated from the examples
- Given the model, the classifier yields the class maximizing the conditional probability $p(b|c)$
  - For the boundary detection task, a punctuation mark is a boundary if

$$p(\texttt{true}|c) = \frac{p(\texttt{true},c)}{p(\texttt{true},c) + p(\texttt{false},c)} > 0.5$$

MaxEnt solution

- It can be proven that the expression for p(b,c) is the unique solution of the MaxEnt problem given the feature constraints
  - Given the two probability function sets

$$P = \{p | E_p[f_j] = E_{\tilde{p}}[f_j],\ j = 1, \ldots, k\}$$

$$Q = \{p | p(b,c) = \pi \prod_{j=1}^{k} \alpha_j^{f_j(b,c)},\ \alpha_j > 0\}$$

  - the MaxEnt theorem states that if $p^* \in P \cap Q$ then $p^* = \text{argmax}_{p \in P} H(p)$. Furthermore, p^* is unique.
  - The solution as a maximum entropy model does not assume facts beyond those in the constraints
  - The same expression can be proved to be the solution to the maximum likelihood formulation, i.e. p^* is the distribution that fits the data as closely as possible

Parameter estimation 1

- The parameters $\alpha_j,\ j = 1, \ldots, k$ of the solution are determined by the Generalized Iterative Scaling procedure
  - Set
$$C = \max_{(b,c) \in \mathcal{B} \times \mathcal{C}} \sum_{j=1}^{k} f_j(b, c)$$
  - Add a correction feature
$$\forall_{(b,c) \in \mathcal{B} \times \mathcal{C}}\ f_{k+1} = C - \sum_{j=1}^{k} f_j(b, c)$$
  - such that
$$\forall_{(b,c) \in \mathcal{B} \times \mathcal{C}} \sum_{j=1}^{k+1} f_j(b, c) = C$$

Parameter estimation 2

- All events have at least one active feature

$$\forall_{(b,c)\in\mathcal{B}\times\mathcal{C}} \; \exists_{f_j} \; f_j(b,c) = 1$$

- The following procedure converges to p^*

$$\alpha_j^{(0)} = 1$$

$$\alpha_j^{(n+1)} = \alpha_j^{(n)} \left[\frac{\tilde{E}[f_j]}{E^{(n)}[f_j]} \right]^{\frac{1}{C}}$$

- where

$$E^{(n)}[f_j] = \sum_{(b,c)\in\mathcal{B}\times\mathcal{C}} p^{(n)}(b,c) f_j(b,c)$$

$$p^{(n)}(b,c) = \pi \prod_{j=1}^{k+1} [\alpha_j^{(n)}]^{f_j(b,c)}$$

Parameter estimation 3

- The observed mean for each feature is computed by a normalized count on the sample corpus $\mathcal{S} = \{(b_1, c_1), \ldots, (b_N, c_N)\}$

$$E_{\tilde{p}}[f_j] = \sum_{i=1}^{N} \tilde{p}(b_i, c_i) f_j(b_i, c_i) = \frac{1}{N} \sum_{i=1}^{N} f_j(b_i, c_i)$$

- The computation of the estimated mean is intractable in a model with k (overlapping) features

$$E^{(n)}[f_j] = \sum_{(b,c) \in \mathcal{B} \times \mathcal{C}} p^{(n)}(b, c) f_j(b, c)$$

  - $\mathcal{B} \times \mathcal{C}$ consists of 2^k distinguishable events
  - The computation can be approximated by summing over the context in $\mathcal{S}$ rather than in $\mathcal{B} \times \mathcal{C}$

$$E^{(n)}[f_j] = \sum_{i=1}^{N} \tilde{p}(c_i) \sum_{b \in \mathcal{B}} p^{(n)}(b|c_i) f_j(b, c_i)$$

SentenceDetector

- CLI invocation

```
> opennlp SentenceDetector models/en-sent.bin
Loading Sentence Detector model ... done (0.207s)
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29. Mr. Vinken is
    chairman of Elsevier N.V., the Dutch publishing group.
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
```

- Java interface

```
SentenceModel model = new SentenceModel(modelInputStream);
SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);
String sentences[] = sentenceDetector.sentDetect(textString);
Span sentences[] = sentenceDetector.sentPosDetect(textString);
```

  - The SentenceDetectorTrainer can be exploited to train a new model
    - The training file contains a sentence per line

Tokenizer

- The Tokenizer segments an input character sequence into tokens
  - words, punctuation, numbers,...
- OpenNLP has multiple Tokenizer implementations
  - Whitespace Tokenizer
    non whitespace sequences are idetified as Tokens
  - Simple Tokenizer
    Sequences of the same character class are Tokens
  - Learnable Tokenizer
    A maximum entropy tokenizer; uses a probability model to detect token boudaries

```
> openNLP/bin/opennlp TokenizerME models/en-token.bin
Loading Tokenizer model ... done (0.577s)
Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.
Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov. 29 .
```

Tokenizer API

- The WhitespaceTokenizer and the SimpleTokenizer can be retrieved from the static field INSTANCE of the corresponding class
- To instatiate the TokenizerME a Token Model must be created

```
TokenizerModel model = new TokenizerModel(modelInputStream);
Tokenizer tokenizer = new TokenizerME(model);
String tokens[] = tokenizer.tokenize(textString);
Span tokenSpans[] = tokenizer.tokenizePos(textString);
```

  - The probabilities associated to each token can be retrieved with the method `getTokenProbabilities()`
  - The training file contains the tokens separated either by a space or by the <SPLIT> tag (one sentence per line)

Name Finder

- Detection of Named Entities and numbers in text
  - A trainable model is exploited to detect the entities
    - The model depends on the language and on the entity type
    - A set of pre-trained models is available in the OpenNLP library
      - en-ner-date, en-ner-location, en-ner-money, en-ner-organization, en-ner-percentage, en-ner-person, en-ner-time
    - The processing needs is performed on the tokenized text

Person finder

```
> openNLP/bin/opennlp TokenNameFinder models/en-ner-person.bin
Loading Token Name Finder model ... done (1.585s)
Pierre Vinken , 61 years old , will join the board as a nonexecutive director Nov . 29 .
<START:person> Pierre Vinken <END> , 61 years old , will join the board as a nonexecutive director Nov . 29 .
```

Name types

- The model depends on a specific type

Date finder

```
> openNLP/bin/opennlp TokenNameFinder models/en-ner-date.bin
Loading Token Name Finder model ... done (1.516s)
Pierre Vinken , 61 years old , will join the board as a nonexecutive director November 29 .
Pierre Vinken , 61 years old , will join the board as a nonexecutive director <START:date> November 29 <END> .
```

  - The available model does not work with the abbreviation Nov.

Organization finder

```
➢openNLP/bin/opennlp TokenNameFinder models/en-ner-organization.bin
Loading Token Name Finder model ... done (1.606s)
The UN was founded in 1945 after World War II to replace the League of Nations , to stop wars between countries , and to provide a platform for dialogue .
The <START:organization> UN <END> was founded in 1945 after World War II to replace the <START:organization> League of Nations <END> , to stop wars between countries , and to provide a platform for dialogue .
```

NameFinderME

- The NameFinderME is not thread safe
  - A different instance must be created for each thread
  - The input text must be segmented into documents, sentences, tokens
  - The find method is called for each sentence in a document
  - The clearAdaptiveData must be called after the processing of each document to clear the adaptive data in the feature generators
  - The Span array contains the positions of the tokens of each entity

```
NameFinderME nameFinder = new NameFinderMe(model);

for (String document[][] : documents) {
   for (String[] sentence : document) {
     Span nameSpans[] = nameFinder.find(sentence);
     // do something with the names
   }
   nameFinder.clearAdaptiveData()
}
```

Name Finders

- Other implemented Name Finders
  - DictionaryNameFinder(Dictionary dictionary)
    scans for names inside a dictionary
  - RegexNameFinder(Pattern[] patterns)
    exploits a series of regular expressions (java.util.regex.Pattern)
- Training the NameFinderME
  - The model can be trained on a custom corpus
  - The OpenNLP training file format requires one tokenized sentence per line
    - The entities are marked with spans (<START:person>Marco Maggini<END>)
    - Empty lines separate the documents to restart the adaptive feature generators
    - A training file can contain more than one entity type (the generated model will be able to recognize multiple types)
    - The training data should contain enough sentences (suggested 150.000)

Feature generation

- The NameFinderME has a default feature generator but a custom one can be user defined
  - The same feature generator must be used in training and testing (!!)
  - A feature generator can be defined by the Java API
    - implement the `AdaptiveFeatureGenerator` interface
    - extend the `FeatureGeneratorAdapter` (non adaptive)
  - The feature generation can also be configured by an xml descriptor file

```
AdaptiveFeatureGenerator featureGenerator = new CachedFeatureGenerator(
   new AdaptiveFeatureGenerator[]{
     new WindowFeatureGenerator(new TokenFeatureGenerator(), 2, 2),
     new WindowFeatureGenerator(new TokenClassFeatureGenerator(true), 2, 2),
     new OutcomePriorFeatureGenerator(),
     new PreviousMapFeatureGenerator(),
     new BigramNameFeatureGenerator(),
     new SentenceFeatureGenerator(true, false) });
```

Default implementation

Feature generators 1

- **`CachedFeatureGenerator`**
  - Caches the features of a set of `AdaptiveFeatureGenerator`
- **`WindowFeatureGenerator`**
  - Generates features for a window around the current token using the specified `AdaptiveFeatureGenerator` and the given window size (previous, next)
- **`TokenFeatureGenerator`**
  - Generates a feature which contains the lowercase token
- **`TokenClassFeatureGenerator`**
  - Generates features for the class of the token (e.g. capitalized initial, all numeric, all capitalized,)

Feature generators 2

- **`OutcomePriorFeatureGenerator`**
  - Generates features for the prior distribution of the outcomes
- **`PreviousMapFeatureGenerator`**
  - generates features indicating the outcome associated with a previous occurrence of the word in the document
- **`BigramNameFeatureGenerator`**
  - Generates the token bigram features (with previous and next word)
  - Generates the token class bigram features (with previous and next word)
- **`SentenceFeatureGenerator`**
  - Creates sentence begin and end features (as specified by the constructor parameters)

Document categorizer

- Automatic text classification into a set of predefined classes
  - Uses a MaxEnt classifier
  - A specific model must be trained for a given classification task using an annotated corpus
    - The default format is one document per line, starting with a string representing the category name
  - The categorizer expects an input segmented into sentences
  - Training and testing can be performed with CLI commands or by calling the API in a Java program

➢ opennlp DoccatTrainer -model en-doccat.bin -lang en -data en-doccat.train -encoding UTF-8

➢ opennlp Doccat model < input

PoS tagger

- Assigns the word type given the word and its context
  - A token may have multiple PoS tags
  - The implemented tagger uses a MaxEnt classifier to predict a tag from a give tagset
  - A dictionary can be used to limit the feasible tags for a given word
  - The available English tagger uses the Penn Treebank tag set

➢ opennlp POSTagger models/en-pos-maxent.bin < example-TK.txt

Loading POS Tagger model ... done (2.142s)

Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB the_DT board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NNP 29_CD ._.

Mr._NNP Vinken_NNP is_VBZ chairman_NN of_IN Elsevier_NNP N.V._NNP ,_, the_DT Dutch_JJ publishing_NN group_NN ._.

Rudolph_NNP Agnew_NNP ,_, 55_CD years_NNS old_JJ and_CC former_JJ chairman_NN of_IN Consolidated_NNP Gold_NNP Fields_NNP PLC_NNP ,_, was_VBD named_VBN a_DT director_NN of_IN this_DT British_JJ industrial_JJ conglomerate_NN ._.

Chunker

- Splits the text into syntactically correlated groups of words
  - noun groups, verb groups,...
  - the internal structure of a group is not explained
  - the group role in the sentence is not determined
  - the input is a PoS tagged text

➢opennlp ChunkerME models/en-chunker.bin < example-POS.txt

Loading Chunker model ... done (1.058s)

[NP Pierre_NNP Vinken_NNP] ,_, [NP 61_CD years_NNS] [ADJP old_JJ] ,_, [VP will_MD join_VB] [NP the_DT board_NN] [PP as_IN] [NP a_DT nonexecutive_JJ director_NN] [NP Nov._NNP 29_CD] ._.

[NP Mr._NNP Vinken_NNP] [VP is_VBZ] [NP chairman_NN] [PP of_IN] [NP Elsevier_NNP N.V._NNP] ,_, [NP the_DT Dutch_JJ publishing_NN group_NN] ._.

[NP Rudolph_NNP Agnew_NNP] ,_, [NP 55_CD years_NNS] [ADJP old_JJ] and_CC [ADVP former_JJ] [NP chairman_NN] [PP of_IN] [NP Consolidated_NNP Gold_NNP Fields_NNP PLC_NNP] ,_, [VP was_VBD named_VBN] [NP a_DT director_NN] [PP of_IN] [NP this_DT British_JJ industrial_JJ conglomerate_NN] ._.

Chunker training

- The chunker can be trained to deal with a new language, a different context, or to improve its performance by providing more examples
  - training data consists in three columns (word, PoS tag, chunk tag)
  - the chunk tag contains the name of the type and a letter to indicate if the current word is the first in the chunk (B) or if its inside the chunk (I)
    - B-NP I-NP ; B-VP I-VP ; ....
    - Sentences are separated by an empty line

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

Parsing

- Chunking parser & treeinsert parser (experimental)
  - Uses the Penn Treebank format for parse trees with one sentence per line
  - Contains also a PoS tagger
  - Documentation still work in progress….

➢opennlp Parser models/en-parser-chunking.bin < toparse.txt

Loading Parser model … done (8.111s)

(TOP (S (NP (DT The) (NN policeman)) (VP (VBD shot) (NP (DT the) (NN thief)) (PP (IN with) (NP (DT the) (NN gun)))) (. .)))

(TOP (S (NP (DT The) (NN horse)) (VP (VBD raced) (SBAR (IN past) (S (NP (DT the) (NN barn)) (VP (VBD fell))))) (. .)))