From shallow to deep natural language processing
A hands-on tutorial

Johan Bos and Malvina Nissim

November 7, 2011

The aim of this tutorial is to provide hands-on experience in open-domain text processing, covering the following topics: tokenisation, part-of-speech tagging, named entity recognition, parsing, semantic processing, and logical inference. The tutorial will comprise an overview of statistical modelling for natural language processing tasks and brief introductions to the above topics, followed by practical exercises. It will be centered around the state-of-the-art C&C tools and Boxer. No prerequisite knowledge is required; nevertheless basic knowledge of shell commands in linux/unix environments is a plus for getting the most out of this course.

Contents

Getting Started

The exercises of the tutorial require a running version of the C&C tools and Boxer. These can be downloaded following the instructions at the following URL:

```
http://svn.ask.it.usyd.edu.au/trac/candc/wiki
```

There are distributions for Linux, Mac OS or Windows CYGWIN operation systems. The Linux or Mac OS environment is preferred. Ensure you got the development version of the tools, which you get via the subversion repository. Boxer requires the installation of SWI Prolog.

```
http://www.swi-prolog.org
```

Note that all shell commands in this tutorial assume that your directory is the `candc` directory (but it could have a different name). Within this directory there should be the directories `candc/bin` containing all binaries, and a directory `candc/working`, for storing temporary results. It's handy to create a new directory for the purpose of this tutorial:

```
mkdir -p working/tutorial
```

Abbreviations

CCG Combinatory Categorial Grammar

DRS Discourse Representation Structure

DRT Discourse Representation Theory

FOL First-Order Logic

NER Named entity recognition

POS Part of speech

1 Tokenisation

Armchair material

Tokenisation is typically the first thing you do in a pipeline of natural language processing tools. It involves usually two tasks, which are often performed at the same time:

1. separating punctuation symbols from words; and
2. detecting sentence boundaries

sentence boundary detection

Often white space is used as separation marks between tokens. Sentences are usually separated by new lines. (But there are other ways of doing this — for instance by using an XML markup language.)

At first it might sound like a tedious and rather trivial thing to do — how difficult can it really be to identify punctuation symbols such as hyphens, commas, and full stops? The problem is that many punctuation symbols are ambiguous in their use. To give an example: a hyphen can be used in a football score, in a range of numbers, in a compound word, or to divide a word at the end of line.

punctuation

For an accurate detection of sentence boundaries it is important to distinguish full stops in the use of abbreviations, and when they mark the end of a sentence (an abbreviation could be the end of the sentence, in which case there is only one full stop!).

Hands-on stuff!

First we need to install the tokeniser. This is done by invoking the shell command

```
make bin/t
```

in the `candc` directory. Now we're ready to play around with the tokeniser.

Exercise: 1.1 Tokenising
The following text is taken from Wikipedia (`http://en.wikipedia.org/wiki/Kate_Bush`).

> Kate Bush (born 30 July 1958) is an English singer, songwriter, musician, and record producer. Her eclectic musical style and idiosyncratic lyrics have made her one of England's most successful solo female performers of the past 30 years. Bush was signed by EMI at the age of 16 after being recommended by Pink Floyd's David Gilmour. In 1978, aged 19, she topped the UK charts for four weeks with her debut song "Wuthering Heights", becoming the first woman to have a UK number one with a self-written song.

Copy it in a file (called it for instance `katebush.txt` and store it in the `working/tutorial` directory. Now tokenise it with the following command:

```
bin/t --input working/tutorial/katebush.txt
```

This will send the output of the tokeniser to the screen. Compare the tokenised text with the raw version.

Exercise: 1.2 Tokenisation options
Explore the various options (you can see them with `bin/t --help` command). Use the `--output` option to redirect the output of the tokeniser to a file. And `--mode rich` to generate character offsets.

2 Part-of-speech tagging

Armchair material

As we know, words can be used as *nouns*, *adjectives*, *prepositions*, *verbs*, and so on. These categories are called parts of speech. Part-of-speech tagging, which you usually find written simply as POS tagging, is the task of automatically labelling each *token* in the sentence with its part of speech .

part of speech

That's not so difficult, is it? Then let's consider the following sentence:

> The function of sleep, according to one school of thought, is to consolidate memory.

Here are the outputs of the same sentence labelled by two different taggers:

> The-DET function-NOUN of-PREP sleep-NOUN ,-PUN according-VERB to-PREP one-DET school-NOUN of-IN thought-NOUN ,-PUN is-VERB to-PREP consolidate-VERB memory-NOUN .-PUN

> The-DET function-VERB of-PREP sleep-VERB ,-PUN according-VERB to-PREP one-DET school-NOUN of-IN thought-VERB ,-PUN is-VERB to-PREP consolidate-VERB memory-NOUN .-PUN

As you can see, there are considerable differences between the output of these taggers. They are caused by two main issues:

- ambiguities in natural language
- choice of tagset

Ambiguity Many words have more than just one syntactic category. Can you think of some?

> The *back* door = adjective
> On my *back* = noun
> Win the voters *back* = adverb
> Promised to *back* the bill = verb

The POS tagging problem is to determine the POS tag for *a particular instance* of a token. When we perform the task of POS tagging, we try to determine which of these syntactic categories (i.e., POS tags) is the most likely for a particular use of a token in a sentence.

Tagset Which and how many tags should we use? This is an issue researchers have been troubled with for a very very long time. Already in ancient Greek times, Aristotle distinguished between *three* categories: nouns, predicates, and conjunctions. Slightly later, a set of eight categories proposed by Dionysius Thrax, in the 2nd century BC, was one that was maintained (more or less unchanged) for a period of about two thousand years! And here is what Mark Twain says about the matter in his book "The Awful German Language:"

> *There are ten parts of speech, and they are all troublesome.*

In the past decades, also due to the development of NLP techniques for more sophisticated processing, the tagsets used have been extended to comprise even up to 100 tags. Of course, tagsets are not always independent of the language, and often also application dependent.

In this tutorial we use the Penn Treebank tagset (see Appendix), which comprises 36 different parts-of-speech plus punctuation. So, the sentence we have seen above would be assigned the following more specific tags.

> The-DT function-NN of-IN sleep-NN ,-, according-VBG to-TO one-CD school-NN of-IN thought-NN ,-, is-VBZ to-TO consolidate-VB memory-NN .-.

Well, how good is it? The performance of current state-of-the-art POS taggers shows an accuracy per token of around 97–98%. This is impressive! But what about accuracy *per sentence*? How many fully correctly tagged sentences do we get with a figure like this for token accuracy?

Hands-on stuff!

The POS-tagger is activated by the command `bin/pos`. Here is an example:

```
bin/pos --input working/tutorial/katebush.tok --model models/pos
```

As you can see from the result, the output contains of the tokens, a vertical bar, and the assigned part of speech.

```
bin/pos --input working/tutorial/katebush.tok --model models/pos --output
working/tutorial/katebush.pos
```

Exercise: 2.1 Finding tagging mistakes
Try to find mistakes in the tagged text, and try to give an explanation as to what caused the mistake.

Exercise: 2.2 Tagging homographs
Homographs are words that spelled the same, but have different meanings (and usually different pronunciations). Let's see what the POS-tagger makes of them! Try the following examples (thanks to A. Terry Bahill):

A cat with nine lives lives around the corner.
The dove dove into the deep grass.
I did not object to that object.
The landfill was so full it had to refuse refuse.

Exercise: 2.3 Part of speech tags
In the appendix of this document you will find a list of all tags used by the pos-tagger. Complete the table by filling in the blanks.

3 Named entity recognition

Armchair material

Named entities are phrases that contain **names** of any type which could be of interest for natural language processing tasks. These could be names of, as in most standard approaches, *persons*, *organisations*, *locations*, *times* and *quantities*. But they could also be *genes*, *proteins*, *cell types* and any other sort of interesting entity for those who want to mine biological data. Or for someone building an application concerning published works, the names entities of interest could comprise *films*, *books*, *paintings* and the like.

So, first one has to find out *which entities* are of interest for our field and task, and then *assign a label* to them. In other words, there are two steps to named entity recognition (NER):

1. *detecting* named entities
2. *classifiying* named entities

However, these two step are often performed in tandem and difficult to distinguish as two separate phases.

So, how are we going to find these entities? Well, one way would be to have very long precompiled lists of names (usually called *gazetteers*), and in some cases they do not work that bad. For example, we could compile lists of countries, cities, rivers, and so on, and the list might be quite exhaustive. But whereas this holds for more "closed" classes, this is not true for more open ones, such as organisations. New companies are born (and die) every day, and it would be not only ridiculously time-consuming but perhaps also quite pointless and foolish to try list them all.

But the problem is not only completeness. *Ambiguity* (yet again) is the other reason why gazetteers cannot fully work. Is "Washington" a person or a place? Does "Ericson" refer to a company or a person? Is "1984" a time expression or a measure phrase? Well, it depends on the context, of course. What the NER module is supposed to do is exactly dealing with this problem: tagging entities in context (a sentence).

The NE tagger that we use in our tutorial takes care of seven different classes, as shown in the table below.

NE	Description	Example
PER	named person	Kate Bush
DAT	date expressions	January 16
TIM	time expressions	5 p.m.
LOC	named locations	England
MON	monetary expressions	50 million dollar
ORG	organisation	Pink Floyd
PCT	percentage	10%

Hands-on stuff!

Exercise: 3.1 Using the NE tagger
Try this to start the tagger:

```
bin/ner --input working/tutorial/katebush.pos --model models/ner
```

Exercise: 3.2 Finding tagging mistakes
Try to find mistakes in the tagged text, and try to give an explanation as to what caused the mistake.

Exercise: 3.3 Training a tagger
Add a new NE class to the training data. Use `bin/train_ner` to create a new model. Then evaluate the new model on a different portion of data.

4 Syntax

Armchair material

A parser assigns syntactic structure to a string, based on a grammar and lexicon. We follow Combinatory Categorial Grammar in defining the lexicon and grammar for English. CCG is a lexicalised theory of grammar: it has many lexical categories, but few grammar rules (combinatory rules as they are called in CCG).

Combinatory Categorial Grammar

In a CCG grammar, categories are assigned to words and expressions. Categories can be simple (S for sentence, NP for noun phrase, N for noun, PP for prepositional phrase) or combined. The combined categories are composed by the use of the slashes "\" and "/". For instance, S\NP for an intransitive verb, and N/N is the category for an adjective. The slashes indicate directionality: the \ tells to look at the immediate left, the / at the immediate right. So the N/N category encodes the information that it is looking for an N on its right. The S\NP category encodes the information that it is looking for an NP on its left.

CCG uses combinatory rules to combine smaller phrases (categories) into larger ones. There is only a dozen number of such rules. The most common ones are application and composition. The rules can be defined using the following templates:

```
X/Y   Y          Y   X\Y         X/Y   Y/Z         Y\Z  X\Y
-------[fa]      -------[ba]     ---------[fc]     --------[bc]
   X                X              X/Z              X\Z
```

Here the abbreviations fa and ba stand for forward and backward applicaton respectively, and fc and bc for forward and backward composition. There are also other rules that deal with type shifting, coordination, and punctuation, depending on the version of CCG theory you're using.

Hands-on stuff!

To produce syntactic structure (in the form of CCG derivations) we use the C&C parser, a statistical parser for CCG, trained on a large database of CCG derivations (CCGbank). For convenience we will use a combined program that can be started via `bin/candc` that also performs part-of-speech tagging and named entity recognition.

The parser expects one sentence per line, and the sentences need to be tokenised. Here is how you run the parser:

```
bin/candc --input <FILE> --models models/boxer --candc-printer boxer
```

What do these options mean? The `--models models/boxer` option selects a pre-trained statistical model of the grammar (it's a large model, that's why it takes quite a bit of time to load it — if you run the parser in server mode you don't face this problem). The `--candc-printer boxer` option ensures that the output is displayed in CCG derivations. It is possible to redirect the output to a file via the `--output <FILE>` option.

Exercise: 4.1 Parsing
Try the parser using the command above on a (tokenised) text. You can also try to the parser on a raw text, and find out why tokenisation is important!

Exercise: 4.2 Inspecting the output

Create a file `working/tutorial/syn1.txt` with the following sentence:

```
Bill Gates stepped down as chief executive officer of Microsoft
on January 12, 2000.
```

Tokenise it, then parse it. What are the lexical categories? And the combinatorial rules?

Exercise: 4.3 Parsing homographs
Try the parser on the sentences with homographs given in Exercise 2.2 on Page 7.

5 Semantics

In this part of the tutorial we will see how we can construct semantic representations on the basis of the syntactic derivations produced by the parser. The tool we're going to use for this purpose is **Boxer**. (If you haven't compiled Boxer yet, type: `make bin/boxer` on the command line.)

Boxer

Armchair material

The semantic representations that we will produce are known as Discourse Representation Structures (DRSs), which are proposed in Discourse Representation Theory , a formal theory of natural language meaning dealing with a wide variety of linguistic phenomena.

Discourse Representation Theory

Simply put, a DRS consists of a pair of discourse referents (also known as the domain of the DRS) and a set of conditions. A discourse referens denotes an entity, a condition constrains the interpretation of this entity. DRSs are recursive structures — a DRS might contain other DRSs constructed with the logical operators negation, disjunction, or implication.

DRSs can be converted to formulas of first-order logic. Discourse referents are interpreted as existential or universal quantifiers, depending on the position of the DRS in which it occurs is embedded in other DRSs or not. The structure of DRSs is used in pronoun resolution: discourse referents in embedded DRSs are not accessible for pronouns.

Hands-on stuff!

Exercise: 5.1 Boxing
Create a file `working/tutorial/sem1.txt` with the following three sentences:

```
Bill saw a busy manager.
Bill saw every busy manager.
Bill saw no busy manager.
```

Use the tokeniser (Section 1) to tokenise these sentence in a file `working/tutorial/sem1.tok`. Then produce a CCG derivation with the parser (Section 4), and redirect the output in the file `working/tutorial/sem1.ccg`. Then run Boxer with the following command:

```
bin/boxer --input working/tutorial/sem1.ccg --box --resolve
```

This should produce DRSs both in Prolog format and pretty-printed boxes.

Exercise: 5.2 DRS structure
Compare the DRS structures of the three DRSs. The `+` indicates a conjunction of DRSs, the `>` an implication, and `--` a negation. Try to paraphrase the content of the DRSs in English.

Exercise: 5.3 Output formats
Run the same command as before but now add the option `--format no` and then again with `--format xml`.

Exercise: 5.4 Boxing
Create a file `working/tutorial/sem2.txt` with the following three lines:

```
<META>text1
Mr. Jones bought an expensive car.
He is a busy manager.
```

Tokenise this file, then parse it and store the output in a file `working/tutorial/sem2.ccg`. Now run Boxer on this file, once without the `--resolve` option, and once with, then compare the results:

```
bin/boxer --input working/tutorial/sem2.ccg --format no --box
bin/boxer --input working/tutorial/sem2.ccg --format no --box --resolve
```

The `<META>` tag causes the sentences following it to be analysed in one DRS, rather than in a separate DRS for each DRS. You can have more than one `<META>` tags in an input file.

Exercise: 5.5 Thematic Roles
Boxer employs a neo-Davidsonian approach to event semantics. This means that verbs introduce discourse referents denoting events, which are linked to participants of the event by thematic roles. By default, Boxer uses a simple set of roles: agent, patient, and theme. Run Boxer on the example with the option `--roles verbnet`. Can you spot the differences?

Exercise: 5.6 Discourse Relations
If you run Boxer with the option `--semantics sdrs` you will see that the output is produced in the form of Segmented Discourse Representation Structures, following SDRT. Boxer has currently limited ways to automatically assign discourse structure and rhetorical relations to texts (it assigns almost always the same discourse relation between segments), but this will hopefully change in the near future.

Exercise: 5.7 First-order Logic
In the next section we will see how we can perform various inference tasks on DRSs. The way this works is by translating DRSs into formalus of first-order logic, that are then given to automated theorem provers and model builders. If you run Boxer with the option `--semantics fol` you will see the translation into first-order logic.

6 Inference

Here we will see how we can use techniques from automated deduction to draw logical inferences from texts. We will do this with the help of **Nutcracker**, a system for recognising textual entailment.

Armchair material

The two tools we are going to use are a theorem prover and a model builder for first-order logic (FOL). We will translate the DRSs generated by Boxer to FOL and then give it to the theorem prover and model builder. As the name suggests, a theorem prover attempts to find out whether a given input formula is a theorem. In other words, it checks whether the input is a validity, that is, true in all possible models. On the other hand, a model builder attempts to construct a model for a given input.

theorem prover

model builder

How can we make use of these tools? Imagine a text is inconsistent, such as the following sentence:

> Mr Jones is a woman.

Say we produced a semantic representation for this sentence, and generated background knowledge that Mr entails a male human entity, and woman a female human entity, and that male and female are disjoint properties. We all put this in one big formula – let's call it ϕ. Now when given ϕ to a model builder, the model builder will fail to find a model, because there isn't one satisfying the input (ϕ doesn't have a model, or ϕ is not satisfiable). It might be that in such cases the model builder is going on forever (or for a very long time) trying to find a model.

Therefore we also use a theorem prover. But we don't give it ϕ. We give it the negation, $\neg\phi$. Why? Well think about it: if ϕ isn't true in any model, then the negation of ϕ must be true in all models. And that's precisely what theorem provers are good in finding out.

In sum: the theorem prover and model builder are complementary tools. We always use them together. When we give ϕ to the model builder, we give $\neg\phi$ to the theorem prover. If the theorem prover finds a result, we know that the input text is inconsistent, and if the model builder finds a result, we know that the text is consistent. So basically, we're now able to perform consistency checking for texts.

consistency checking

Hands-on stuff!

Exercise: 6.1 Consistency checking
Make a directory `working/tutorial/rte`. Create two text files in this directory: a file named `t`, and a file named `h`. Make the contents of the first file the sentence "Bill Gates bought a car.", and the that of the second "Bill Gates bought no car.". Then run Nutcracker:

```
bin/nc --dir working/tutorial/rte --info
```

Exercise: 6.2 Background knowledge
Nutcracker calculates background knowledge using the WordNet lexical database. It uses this background knowledge while it draws inferences. Let's look at some examples:

```
t:John has a dog.
h:John has an animal.
```

```
t:John has an animal.
h:John has a dog.
```

```
t:John likes no animal.
h:John likes a dog.
```

What does Nutcracker predict for these three textual entailment pairs? Find out what the background knowledge ontology looks like by viewing the contents of the file `mwn.pl` in the directory of each example.

Exercise: 6.3 Looking under the hood
After using Nutcracker, have a look at files generated for drawing inferences. They are stored in the directory of each example, and bear names with the extensions `*.in` and `*.out`. These files contain the last generated input for and output from a theorem prover or model builder.

Appendix: Part of Speech (POS) tagset

Tag	Desciption (Penn Treebank tagset)	Example
CC	Coordinating conjunction	
CD	Cardinal number	
DT	Determiner	
EX	Existential “there”	
FW	Foreign word	
IN	Preposition or subordinating conjunction	
JJ	Adjective	
JJR	Adjective, comparative	
JJS	Adjective, superlative	
LS	List item marker	
MD	Modal	
NN	Noun, singular or mass	
NNS	Noun, plural	
NNP	Proper noun, singular	
NNPS	Proper noun, plural	
PDT	Predeterminer	
POS	Possessive ending	
PRP	Personal pronoun	
PRP$	Possessive pronoun	
RB	Adverb	
RBR	Adverb, comparative	
RBS	Adverb, superlative	
RP	Particle	
SYM	Symbol	
TO	“to”	
UH	Interjection	
VB	Verb, base form	
VBD	Verb, past tense	
VBG	Verb, gerund or present participle	
VBN	Verb, past participle	
VBP	Verb, non-3rd person singular present	
VBZ	Verb, 3rd person singular present	
WDT	Wh-determiner	
WP	Wh-pronoun	
WP$	Possessive wh-pronoun	
WRB	Wh-adverb	