TreeCloud & Unitex: an increased synergy

Claude Martineau

TreeCloud & Unitex: an increased synergy

Claude Martineau

UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

LABORATOIRE D'INFORMATIQUE GASPARD-MONGE

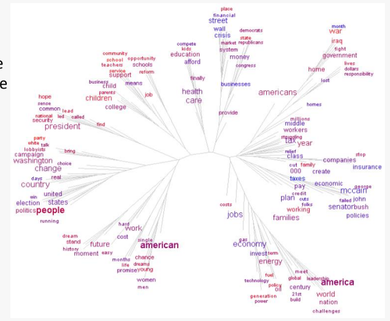
Sous la co-tutelle de :
CNRS
ÉCOLE DES PONTS PARISTECH
ESIEE PARIS
UPEM • UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

Projet PEPS CNRS/UPE Eclavit

TreeCloud is a *tree cloud visualization* of a text

TreeCloud builds a tree cloud visualization of a text, which looks like a tag cloud where the tags are displayed around a tree to reflect the co-occurrence distance between the words in the text.

The adjacent treecloud gives an overview of Barack Obama's 2008 presidential campaign speeches.



Tree construction process

Input text	
Remove empty words	***Stopword lists of French, English, Spanish, etc.***
Word selection	***The n most frequent words***
Co-occurrence search	***Co-occurrence window (sliding window, online version) or separator character***
Get distances between words	***12 co-occurrence distance formulas***
Build the tree	***Call to SplitsTree (downloadable version), integrated (on-line version)***
Size of the words	***Using Frequencies or personalized values***
Color of the words	***Using Frequencies, chronology, dispersion or personalized values***
Draw the tree	***Call to SplitsTree (downloadable version), integrated (online version)***

- The **stopword list** contains ***grammatical words*** and ***auxiliary*** or ***modal verbs***.
  For example, in English: *the*, *of*, *is*, *was*, *have*, *may*, *can*, etc.

  The stopwords are usually the most common words in a language. To build a **meaningful tree,** all these words must be firstly **removed** from the input text.
- Whenever a sliding window of analysis is used, its size (i.e. a number of words) must be given as parameter.

Co-occurrence and distance calculation

Given two words *A* and *B* and:

- O_{11}, observed number of sliding windows containing **both** *A* **and** *B*
- O_{12}, observed number of sliding windows containing *A* **but not** *B*
- O_{21}, observed number of sliding windows **not** containing *A* **but** *B*
- O_{22}, observed number of sliding windows containing **neither** *A* **nor** *B*

the following variables are defined:

- $R_1 = O_{11} + O_{12}$, number of sliding windows containing *A*
- $R_2 = O_{21} + O_{22}$, number of sliding windows **not** containing *A*
- $C_1 = O_{11} + O_{21}$, number of sliding windows containing *B*
- $C_2 = O_{12} + O_{22}$, number of sliding windows **not** containing *B*
- $N = R_1 + R_2 = C_1 + C_2$, number of sliding windows
- $E_{11} = (R_1C_1/N)$, expected number of sliding windows containing **both** *A* and *B*
- $E_{12} = (R_1C_2/N)$, expected number of sliding windows containing *A* **but not** *B*
- $E_{21} = (R_2C_1/N)$, expected number of sliding windows **not** containing *A* **but** *B*
- $E_{22} = (R_2C_2/N)$, expected number of sliding windows containing **neither** *A* **nor** *B*

The definitions of **co-occurrence formulas** are the following:

- **jaccard**: $1 - O_{11} / (O_{11} + O_{12} + O_{21})$
- **liddell**: $1 - (O_{11}O_{22} - O_{12}O_{21}) / (C_1C_2)$
- **dice**: $1 - 2O_{11} / (R_1 + C_1)$
- **hyperlex**: $1 - \max(O_{11} / R_1, O_{11} / C_1)$
- **poissonstirling**: $O_{11}(\log O_{11} - \log E_{11} - 1)$
- **chisquared**: $1000 - N(O_{11} - E_{11})^2 / (E_{11}E_{22})$
- **zscore**: $1 - (O_{11} - E_{11}) / \mathrm{sqr}(E_{11})$
- **ms**: $1 - \min(O_{11} / R_1, O_{11} / C_1)$
- **oddsratio**: $1 - \log((O_{11}O_{22}) / (O_{12}O_{21}))$
- **loglikelihood**: $1 - 2(O_{11}\log(O_{11}/E_{11}) + O_{12}\log(O_{12}/E_{12}) + O_{21}\log(O_{21}/E_{21}) + O_{22}\log(O_{22}/E_{22}))$
- **gmean**: $1 - O_{11}/\mathrm{sqr}(R_1C_1) = 1 - O_{11}/\mathrm{sqr}(NE_{11})$
- **mi** (mutual information): $1 - \log(O11/E_{11})$
- **ngd** (normalized Google distance): $(\max(\log R_1, \log C_1) - \log O_{11}) / (N - \min(\log R_1, \log C_1))$

Several versions of TreeCloud

Downloadable version in Python

- 2009: TreeCloud 1.3 for Windows, Linux, Mac developed by Philippe Gambette
  Use of SplitsTree 4.10 to draw the tree

On-line version in C

- 2009: 1st C version developed by Jean-Charles Bontemps
- 2012: Transition to Unicode developed by Claude Martineau
- 2014: 1st implementation of Unitex developed by Claude Martineau

Unitex/GramLab is a *corpus analyser* and *annotation* tool

- **Based** on **Automata** and **RTNs** with **outputs**
- **Multilingual**: Up to 22 languages (French, English,..., Greek, ... , Korean, Thaï)
- **Unicode 3.0** (UTF8, UTF16LE, UTF16BE)
- **Cross-platform**: Linux, macOS, Windows
- **Open source**: https://github.com/UnitexGramLab
- **Website and binary installers**: http://unitexgramlab.org
- **Under development since 2001 by a group of passionate volunteers**

Unitex/GramLab uses *linguistic resources*:

- **DELA (LADL electronic dictionaries)**

  A typical DELA entry is composed by a simple or compound inflected form, followed by a lemma and grammatical information. Each entry can be associated with syntactic and semantic attributes and inflection rules:

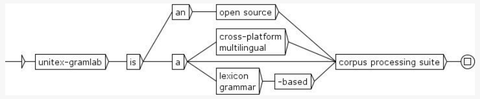
  inflected_form,lemma.grammatical_information+attributes:inflection_rule

  Example: Given the French simple word "avocat" (*lawyer*) and the compound word "avocat d'affaires" (*business lawyer*), a DELA representation would be:

avocat,avocat.N+Hum+Prof:ms	avocat d'affaires,avocat d'affaires.N+Hum+Prof:ms
avocate,avocat.N+Hum+Prof:fs	avocate d'affaires,avocat d'affaires.N+Hum+Prof:fs
avocats,avocat. N+Hum+Prof:mp	avocats d'affaires,avocat d'affaires. N+Hum+Prof:mp
avocates,avocat.N+Hum+Prof:fp	avocates d'affaires,avocat d'affaires.N+Hum+Prof:fp
lawyer	*business lawyer*

  1 inflected form
  2 ,canonical form
  3 .grammatical category
  4 +semantic attributes
  5 :inflectional information (m: masculine, f: feminine, s: singular, p: plural)

- **Syntactic or semantic rules called «local grammars» represented by graphs**
  - Graphical representations of local grammars are composed by a set of linked boxes.
  - A **successful path** is a path between initial and final states.



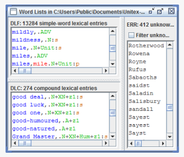
Some examples of **matched** and **unmatched** sequences by the above grammar:

Unitex-GramLab is a corpus processing suite [**MATCH**]
Unitex-GramLab is an open source corpus processing suite [**MATCH**]
Unitex-GramLab is a hard to learn corpus processing suite [**FAIL**]
Unitex-GramLab is [**FAIL**]

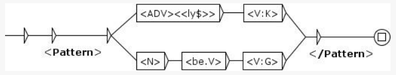
An example of analysis

Application of a dictionary; the result is the **text dictionary,** then **application of a local grammar**
The grammar below contains two search paths:
- an **adverb** (<***ADV***>) ending in ***-ly*** followed by a **past participle** (<***V:K***>)
- a **noun** (<***N***>) followed by a **verb** in **progressive form** (<***be.V***> <***V:G***>)

A lexical mask like ***<V:K>*** refers to the text dictionary.
The recognized sequences are surrounded by the tag <**pattern**>. The results are represented in the form of concordances.



Text dictionary



A very simple local grammar

by one who has <Pattern>unwittingly fallen</Pattern> under your disple
.nce John, "his <Pattern>heart is sinking</Pattern>; I am jealous lest
rchers, having <Pattern>previously determined</Pattern> by lot their o

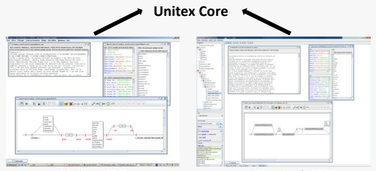
Sample of concordances

Notice that Unitex/GramLab can also produce an annotated text that includes all the tagged patterns.

Several ways to use Unitex/GramLab

Two interfaces written in **JAVA**:
- **Unitex IDE** (classic)
- **GramLab IDE (**project-oriented)

They refer to
Unitex Core written in **C/C++**



Unitex IDE **GramLab IDE**

Use the API C and JAVA (JNI) that provides access to
- a **virtual file system**
- **a persistence layer for resources (alphabets, dictionaries and corpora)**

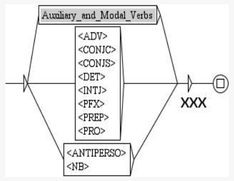
Command lines or system calls with Perl, Python, etc.

How and Why to plug Unitex into TreeCloud?

Construction of the tree with Unitex

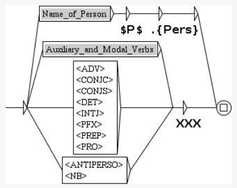
1) **Unitex transforms** the ***input text*** into **a *new text*** with all the **forbidden/stopwords replaced** by the ***XXX*** « word »
2) The ***new text*** is sent to TreeCloud with ***XXX*** as the unique forbidden word (the unique word in the stop list)

Get a larger and more accurate coverage of forbidden words



The adjacent local grammar contains three paths/boxes:
- The **greyed box** represents a call to a **subgraph** that recognizes ***auxiliary*** and ***modal verbs.***
- The second box uses some **lexical masks** (<ADV>, <CONJ>, ...) to match ***grammatical words.***
- The <ANTIPERSO> lexical mask in the last box matches a list of stopwords given by the user.
  Each word of this list is labelled/tagged with ANTIPERSO.
  <***NB***> matches numbers.

Insert multiwords into the tree

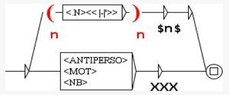


If a dictionary contains compound words, these words can be kept in the tree but all multiwords cannot be listed in a dictionary.

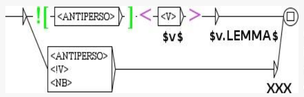
The adjacent grammar contains a path that recognizes person names. The variable P contains the name of a person captured by the subgraph Name_of_Person.

The .{Pers} label is added to the output (2017 online version).

Make a strong selection of the words kept in the tree



Since the number of words kept in the tree is very low (50 up to 150) and depends on their frequency, if a special category should be displayed, their selection should be made carefully. The adjacent grammar keeps only compound nouns.



In order to display verbs into the tree, it could be useful to get their **LEMMA** (since they have many inflected forms). The adjacent grammar is designed for this goal. A similar process is used for languages with nouns or adjectives with cases (e.g. Serbian or Greek).

2017 on-line version of TreeCloud: an improved implementation

Introduction of the concept of file processing

In the online version of 2014, there is only a single Unitex processing for each language. The necessary resources were hard-coded into the program.

In the new 2017 version, there can be several processing procedures for each language. Furthermore, the Serbian language (Latin and Cyrillic) has been added. In order to manage these pairs (language, processing) a concept of processing file has been set up.

For example, in the processing file below, the first line indicates the path for French resources, then three dictionaries (French general dictionary, first name dictionary, toponym dictionary) have to be applied to the text. In the end, the local grammar is applied in the replace mode.

```
REP:TreeCloud_WS/French/src
NB_DICOS=3
        FICHIER=Dela/dela-fr-public.bin
        FICHIER=Dela/prenom-s.bin
        FICHIER=Dela/Prolex-Unitex_1_2_TOPONYMES.bin
GRAMMAIRE:
        FICHIER=Graphs/Treecloud_N_Pers_Top_v1_FR.fst2
        MODE=REPLACE
```

Take advantage of the work already done by Unitex

Unitex/GramLab analysis steps

program called	created files
Normalize	*text.snt*
Tokenize	*tokens.txt, text.cod*
Dico	*dlf, dlc, err*
Locate	*concord.ind*
Concord	
Concordances	**Annotated text**

At the end of the Unitex analysis process, *text.snt* contains a cleaned text (normalization of separator characters), ***text.cod*** contains the list of indexes of the tokens into the ***tokens.txt*** file list.

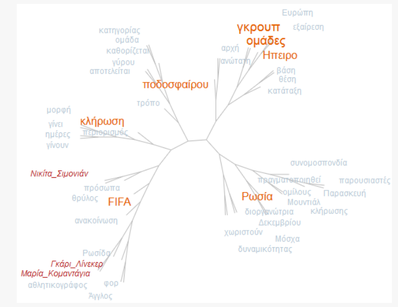
dlf, dlc, err, respectively contain simple words, compound words, unknown words

concord.ind contains the matched sequences with their position into the text (XXX, and multiword units)

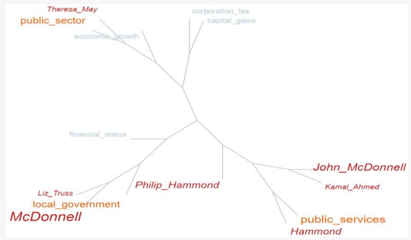
To get the *«new text»*, we **retokenize** the text with **matched sequences** of the ***concord.ind*** file as the new tokens of the text. New ***token.txt*** and ***text.cod*** files are created. This process prevents double reading of the text and double division into words.

Thanks to the **Unitex API** and **virtual file system,** all this work is done **in memory.**

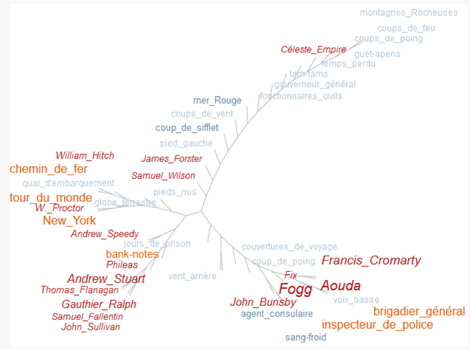
Some examples of trees in different languages



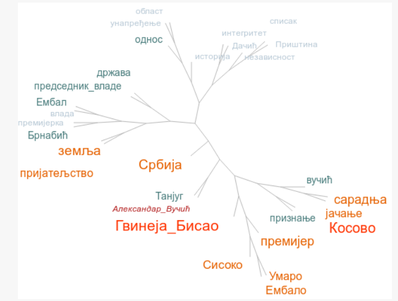
Tree of the 50 most frequent words in a Greek article



Tree of the 20 most frequent compound and person nouns in an English article



Tree of the 45 most frequent compound and person nouns in a Jules Verne's novel



Tree of the 30 most frequent words in a Serbian article

Conclusion

Plug-in of **Unitex** into **TreeCloud** provides:

- **A more accurate representation of forbidden words**
- **All kinds of multiwords to be recognized in the text and presented in the tree**
- **A visual representation of some grammatical or semantic categories of the words.**
- **A faster construction of the tree (via a careful use of the Unitex API)**

http://treecloud.univ-mlv.fr