Description of TextPanorama software, practical implementation

1. Description

The software presented here is designed to display a collection of texts (poems, songs, interviews). This collection—referred to here as the "corpus"—is further partitioned into text categories. For an author's songs, these categories might be "albums," which generally group songs from a specific period (e.g., Leonard Cohen, Georges Brassens, Charles Aznavour, Jacques Brel). In the case of William Shakespeare's 154 sonnets — which serves as our running example—the classification involves grouping the sonnets into eight main themes (or "topics"), a categorization accepted by several scholars of his work; see, for instance, www.dtmvic.com/doc/Lebart_JADT_2018.pdf. This text also appears in www.dtmvic.com/doc/Fr_Brassens_Shakespeare_2022.pdf.

For poetic bodies of work such as those of Alfred de Vigny or Victor Hugo in French literature, the categories might correspond to the collections or publications that mark the milestones of the authors' lives. If the corpus includes multiple authors, the *a priori* classification into categories may simply be based on the authors themselves.

1.1 What does the proposed statistical procedure do? What does it demonstrate?

(Each analysis creates a results folder (containing tables and figures) timestamped with the date, hour, and minute of the analysis. In the general case, this "Results" folder contains approximately 20 graphs and 12 text files.)

Essentially, it provides visualizations—using Correspondence Analysis (CA)—of the "words × texts" lexical table (based on a minimum word frequency). A simultaneous visualization is also generated by selecting only the *nvt* most characteristic words for each text (the parameter *nvt* = 3 in the example). Categories (topics, albums, collections) are then projected as supplementary elements onto the principal planes.

For these supplementary points, Bootstrap confidence regions (ellipses or convex hulls of replications) are plotted, with or without the Bootstrap replications displayed on the graphs for verification purposes. This constitutes a “partial Bootstrap approach”, testing the statistical validity of the partition without calling the CA itself into question. (See: "Which Bootstrap for principal axes methods?" In: *Selected Contributions in Data Analysis and Classification*, P. Brito et al. (eds.), Springer (2007), pp. 581–588.)

(www.dtmvic.com/doc/Lebart_Brito_2007.pdf)

To complement the CA—and in an attempt to detect structural features not visible on the initial principal planes—an additive tree of the texts is plotted using the algorithm that has experimentally demonstrated the best consistency with the CA. The supplementary categories are then positioned within the plane of the tree (see, for example, Lebart_JADT2024.pdf). Then, confidence ellipses and convex hulls make it possible to validate the positions of these additional categories.

1 2. CA and AT: How do they complement each other?

CA and AT methods serve different purposes because they are based on different models of reality. It is worth recalling the quote by Sattath and Tversky (1977) from a seminal paper on additive trees: “It is interesting to note that tree and spatial models are opposing in the sense that very simple configurations of one model are incompatible with the other model. For example, a square grid in the plane cannot be adequately described by an additive tree.” For instance, we know that the CA of a graph representing a checkerboard (a regular grid) perfectly reconstructs that checkerboard within the principal plane formed by axes 1 and 2—something no other classification technique can achieve. CA should therefore be viewed as an essential preliminary step (it allows for the complete reconstruction of the original data if all axes are included). Conversely, in an article titled “Trees or Axes for Text Analysis?” (http://www.dtmvic.com/doc/Lebart_jadt2016.pdf), we demonstrated that ATs yield far better results than CA when describing semantic similarity graphs. It therefore appears essential to employ both approaches simultaneously.

1. 3. Two basic options

1) **The first basic option** (using the parameter "wordbag") transforms texts into "bags of words" (if wordbag = 1) . It is often applied to poetic texts due to their sometimes erratic lexical frequencies. It essentially focuses on the vocabulary itself (the presence or absence of words). See, for example, the article: "Poems and Songs: What can Textual Data Analysis do?" (www.dtmvic/doc/Lebart_JADT_2022.pdf).

This option is not mandatory, but we list it first because, if selected, it serves as a preliminary step for all subsequent processing.

2) **The second basic option** offered (the "anagreg" parameter) aggregates texts based on a categorical variable (e.g., topics, albums).

The case anagreg = 1:

A Correspondence Analysis of the aggregated table (categories × words) is followed by the construction of an additive tree where the nodes represent the categories, allowing for the simultaneous visualization of both words and categories. This purely descriptive option does not involve validation, as the categories are treated as active (rather than supplementary) variables.

The case anagreg = 0.

This is the most general case, leading to much more detailed analyses and, above all, fundamental validation phases. In this case, the relevance of the categories (with respect to the vocabulary used) is questioned and potentially validated by the positions of the Bootstrap ellipses (or convex hulls).

3) The following parameters are described in the interface (Figure 1) with their default values (these default values can be modified since the Python source code for the interface is available).

The key article to consult for the entire approach is "www.dtmvic.com/doc/Lebart_JADT_2026.pdf (Validation of Simultaneous Additive Trees in Textual Data Visualization (2026))."

2. Practical implementation of TextPanorama using the "Shakespeare/Sonnets" example.

It is assumed that the Python software (version 3.11 or later recommended) and its IDLE editor have been downloaded. We also recommend downloading the free text editor Notepad++, which is very useful for handling text and data files—particularly for tasks such as ANSI-to-UTF-8 conversion—as well as for editing Python code, as it offers syntax highlighting. The version presented here runs on Windows 10 or later.

2.1 Python modules

All Python functions are located in a base folder named "BASIC_TextPanorama". This folder is obtained by downloading and then unzipping the compressed file "BASIC_TextPanorama.zip".

This base folder contains the interface module "Interface_TextPanorama.py". This module (interface shown in Figure 1) triggers the execution of the main module, "TextPanorama.py", via the "Execute" button.

The main module, "TextPanorama.py", relies on the library of modules contained in the "LIB_TextPanorama" folder. This library comprises approximately 31 Python modules that are imported mainly by the main module. It is essential that the two modules ("Interface_TextPanorama.py" and "TextPanorama.py") and the library folder ("LIB_TextPanorama") all reside in the same folder—the name of which does not matter.

To get started, open "**Interface_TextPanorama.py**" (from within the "**BASIC_TextPanorama**" folder) in the **IDLE** Python editor and select "**RUN**" followed by "**Run Module**". The interface will be displayed (see Figure 1).

Next, select in the interface the "**DATA_Shakespeare**" folder and, then, the three files (*Text file, Category file, identifier of categories file : see section 2.2 below*) it contains ; the filenames will be displayed once the folder has been selected.

For this initial test run, you can keep all default parameter values and proceed directly to execution.

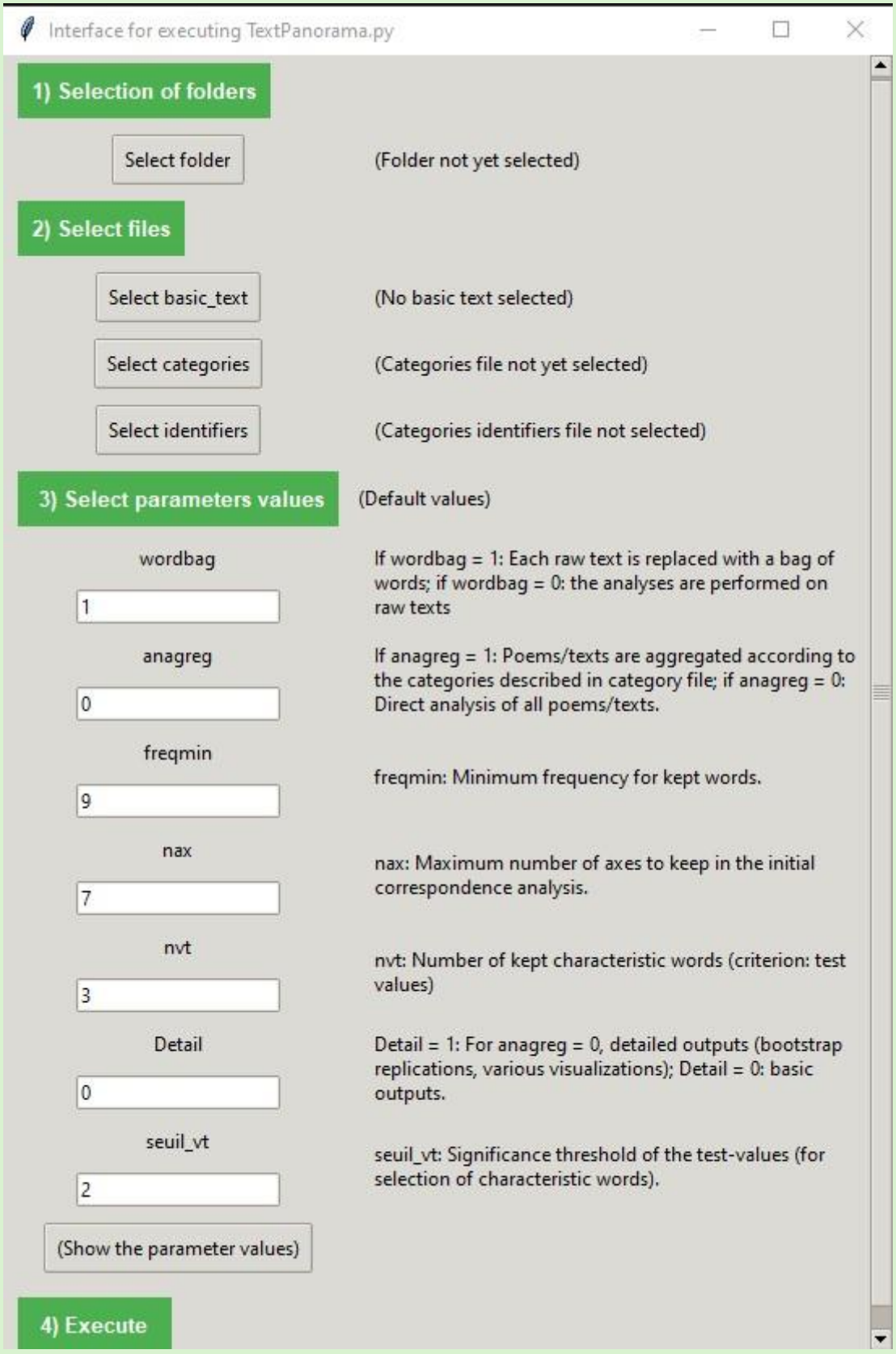

Figure 1. Image produced by the "Interface_TextPanorama" interface.

2.2 The three files from the Shakespeare example:

The data folder: The provided DATA_Shakespeare.zip folder should be downloaded and then unzipped.

The "**DATA_Shakespeare**" folder contains the 3 essential files : The source text file, the categories file, and the category identifiers file. For other applications, the name of the DATA folder and the names of the 3 files within it are arbitrary, as they are selected by the user via the interface.

2.2.1) The source text file. Format: text separators ("****") aligned in column 1. "====": end of file. After 4 spaces, the text identifier: (fewer than 12 characters, no spaces)

Outline of the file "**Sonnet_LowerCase_1_154_2.txt**"

```
****    S01
from fairest creatures we desire increase,
that thereby beauty's rose might never die,
but as the riper should by time decease,
his tender heir might bear his memory:
but thou, contracted to thine own bright eyes,
feed'st thy light'st flame with self-substantial fuel,
. . . . . .

****    S02
when forty winters shall beseige thy brow,
and dig deep trenches in thy beauty's field,
thy youth's proud livery, so gazed on now,
will be a tatter'd weed, of small worth held:
then being ask'd where all thy beauty lies,
where all the treasure of thy lusty days,
to say, within thine own deep-sunken eyes,
were an all-eating shame and thriftless praise.
. . . . . .
====
```

(Moreover, this format is the native text format for the DtmVic software)

2.2.2) The **categories** file in ".csv" format. (Note: this is an Excel-style file using the comma "," as a separator. In countries where Excel defaults to using semicolons, the file must be converted. This is easily done using Notepad++, the free software program highly recommended for this purpose.)

Preview of the file "Categ.csv" file (here, the first 10 sonnets belong to category "1"). 2 columns, 155 rows (154 sonnets + header)

```
Title, T8
S1,1
S2,1
S3,1
S4,1
S5,1
S6,1
S7,1
S8,1
S9,1
S10,1
....
```

2.2.3) The small ".csv" file containing **category identifiers**. The same note applies regarding the "," separators.

The file "Ident_Categ.csv"

```
Topic8, ident
T1, 1_Procreation
T2, 2_YoungMan
T3, 3_DarkLady
T4, 4_Absence
T5, 5_Storm
T6, 6_Rivalry
T7, 7_Death
T8, 8_Eternal_poetry
```

For this type of file, it is advisable to start the identifiers (second column) with a "category number," as these numbers will be used by the category's bootstrap replicates in certain figures.

2.3 Execution.

Once the three data files have been selected, you can—as a first step—simply accept all the interface's default values and click "Execute" at the bottom of the interface menu. The results and graphs (which must be systematically closed to allow the program to continue running) appear in the IDLE interface. Note that all graphs are saved in the "Results_date_hour_minutes" folder created during each execution.

3. TextPanorama.py module

This module serves as the "backbone" of the processing workflow: its main function, "**CA_AT_synthesis**," is triggered directly from the interface and calls upon—either directly or indirectly—the 31 modules within the "**LIB_TextPanorama**" library.

It calculates the base lexical table (words × texts) and proceeds to analyze either the aggregated table (based on categories)—when the `anagreg` parameter is set to 1—or the texts directly when the `anagreg` parameter is set to 0, performing then the following steps in sequence (briefly):

Correspondence analysis of the Texts × Words lexical table.

Projection of categories (supplementary or illustrative variables) onto the principal planes.

Bootstrap confidence ellipses for category positions.

Plotting of an additive tree with texts as vertices (using the Kawada-Kavai algorithm).

Positioning of categories (supplementary or illustrative variables) within the tree plot's plane.

Bootstrap confidence ellipses for category positions. All these operations are controlled by a set of parameters described in the interface shown in Figure 1.

The `Interface_TextPanorama.py` interface calls (via the "Execute" button) a main function located in the `TextPanorama.py` Python module (described below):

That main function is `CA_AT_synthesis()`. This function transforms the initial texts into a "bag of words" (or not, depending on the "wordbag" parameter)—meaning each text is converted into its unique "vocabulary," where each word appears only once per text. This option is often useful for certain poetic texts (containing refrains, choruses or repetitions).

Next, it aggregates the texts by category (or not, depending on the "anagreg" parameter).

If the texts are aggregated (`anagreg = 1`), it calls the function:

`text2tree_Topic()` [brief descriptions, using simultaneous CA and AT, of the aggregated Words x Categories table].

If the texts are not aggregated (`anagreg = 0`), it calls the function:

`text2panorama()` [detailed descriptions, using simultaneous CA and AT, of the Words x Texts table, with categories included as supplementary variables and Bootstrap validation of the locations of categories].

TextPanorama.py module: Python code

```
#
# Description of the file "TextPanorama.py" downloaded as part of the
folder "BASIC_TextPanorama". Editing directly the file "TextPanorama.py"
with an editor such as NotePad++ should provide the same syntactic
coloration.
#-------------------------------------------------------------------
# Synthesis:  Correspondence Analysis + Simultaneous Additive Tree +
bootstrap.
#-------------------------------------------------------------------
# (here : basic texts with categories (albums or topics) with
visualizations and bootstrap validation of categories)
# The main module (TextPanorama.py) must be in the same folder as both
the library folder: 'LIB_TextPanorama', and the interface program
'Visu_CA_SAT_Valid_E.py'
#-------------------------------------------------------------------
import os
import sys; sys.path.append('LIB_TextPanorama')
    # Making available the library of functions included in the provided
     folder: LIB_TextPanorama
import numpy as np
#-------------------------------------------------------------------
# All the following importations are modules included in the folder :
# 'LIB_TextPanorama'
#-------------------------------------------------------------------
import Words_bag as Wb          # Transforming texts into "presence or
                                 # absence of words"
import table_lex as tex         # Building a lexical table
import agreg_tablex as agreg    # Aggregating columns of tablex
#
#---- CA ----
import ac_dtm_AT as act           # CA of "tablexfile.csv"
import f_testval12 as vt          # Computation of test-values
import CA_graf_complet5 as gr5    # CA display + characteristic words
import CA_cat_alb as CA_alb       # CA display + Categories (axes 1, 2)
import Sup_cat_CA as sup_CA       # Coordinates of supplementary
                                 # categories (CA)
#
#---- Bootstrap for CA
import CA_cat_alb_boot as catboot              # CA with replicates
import CA_cat_alb_boot_ellipses as catbootell # CA with confidence
                                  # ellipses for categ./albums/topics
import CA_cat_alb_hulls as CA_hulls            # CA with convex hulls of
                                               #category clouds
import CA_cat_alb_boot_hulls as CA_boot_hulls # CA with convex hulls of
                                               #  replicates
#
#---- SAT (Simultaneous Additive Trees)
import Tree_cat_alb_hulls as TreeHulls  # Convex hulls of categories
                                        # or albums or topics
import AT_NJ_calc as njt    # NJ tree basic computation
```

```
import Tree_base as tbase    # basic tree (Tree_base.jpg), creating
                             # coordinates and identifiers coord_v and
                             #id_v
import Tree_cat as cat       # Drawing the same tree with categories
                             # numbers as identifiers.

import Forest as fo    # Series of cat-ident trees ( when FOREST = 1,
                       # FOREST being an internal parameter)
import axis1_sign as ax1         # Possible change of axes directions
import w_coord_plot as wcp       # Coordinates of words in NJ tree space
import tree_simult_n11 as ct     # Tree of aggregated texts + words
                                 # ("ct" => "Christmas Tree")
import Tree_cat_alb as talb      # Tree with categories (alb = album)
import coord_alb as calb         # Coordinates of categories (1, 2)
import ident_alb as new_id       # Building extended identifiers
#
#---------  Bootstrap for Simultaneous Additive Trees (SAT)
import Tree_cat_alb_boot as talboot  # Tree with Bootstrap replicates of
                                     # categories
import coord_alb_boot as calboot     # Coordinates of categories and
                                     # their replicates
import Dupli_csv_boot as Dup         # replicates of categories
import Tree_cat_alb_boot_ellipses as talboot_ellipses
                                  # Tree with Bootstrap confidence ellipses
import Sup_cat_AT as sup_AT          # Coordinates of supplementary
                                     # categories (AT)
import Tree_boot_hulls as TBH        # Tree with Convex Hulls of
                                     # replicates of categories
#
#*************************************************************
#
# CA_AT_synthesis (dossier_data, chemin, sentier, piste, wordbag,
# anagreg, freqmin, nax, nvt, Detail, seuil_vt):
#
#---------------------------------------
# dossier_data : path to the DATA folder
# chemin     : path to the basic text file (within DATA folder)
# sentier    : path to the categories file (within DATA folder)
# piste      : path to the identifiers of categ file (within DATA folder)
#---------------------------------------
# Parameters
#  "wordbag": If wordbag = 1: Each raw text is replaced with a bag of
#             words (frequency = 1 for each word of the text)
#             If wordbag = 0: the analyses are performed on raw texts.
#  "anagreg": If anagreg = 1: Poems/texts are a priori aggregated
#              according to the categories described in category file.
#             If anagreg = 0: Direct analysis of all poems/texts.
#  "freqmin": Minimum frequency for kept words.If wordbag = 1, the
#             frequency of word W is the number of texts containing W.
#  "nax":     Maximum number of axes to keep in the initial CA
#            (correspondence analysis.
#  "nvt":     Number of kept characteristic words (criterion: test-
#             values).
```

```
#  "Detail":  If Detail = 1: For anagreg = 0, detailed outputs
#             (bootstrap replications, various visualizations).
#             If Detail = 0: basic outputs (default value)
#  "seuil_vt": Significance threshold for the test-values (for selection
#              of characteristic words).
#-----------------------------------------------------------------
#
def CA_AT_synthesis (dossier_data, chemin, sentier, piste, wordbag,
    anagreg, freqmin, nax, nvt, Detail, seuil_vt):
    from pathlib import Path
    from datetime import datetime
#-------------
# A folder "Results" (with date and hours/min) is created within the
folder DATA
#-------------
    stamp = datetime.now().strftime('%Y-%m-%d_%H-%M')
    input_dir = Path(dossier_data).resolve()
    output_dir = input_dir/("Results_" + stamp)
    Path(output_dir).mkdir(exist_ok = True)
#----------------------
    output_results = output_dir
#----------------------
# creating the file "Report.txt" in the folder "Results" created within
# the folder DATA
#----------------------
    input_dir_new = Path(output_results).resolve()
    ch = input_dir_new/"Report.txt"
#----------------------
    g = open(ch, "w+")
    print  ("\n "   + "chemin =  " + chemin + "   \n" )
    print  ("\n "   + "sentier = " + sentier + "  \n" )
    print  ("\n "   + "piste =   " + piste + "  \n" )
    print   ("\n "   + "wordbag  =" + str(wordbag) + ",   anagreg = " +
     str(anagreg) + ",   freqmin = " +  str(freqmin) + "\n" )
    print   ("\n "   + "nax  =" + str(nax) + ",   nvt = " + str(nvt) +
     ",   Detail = " +  str(Detail) + "\n" )
    print   ("\n "   + "seuil_vt  =" + str(seuil_vt) + "\n" )
#------------------------------------ (report)
    g.write("\n\n " + "chemin = " + chemin + "  \n" )
    g.write("\n\n " + "sentier =" + sentier + "  \n" )
    g.write("\n\n " + "piste =  " + piste + "  \n" )
    g.write  ("\n\n "   + "wordbag  =" + str(wordbag) + ",   anagreg = "
     + str(anagreg) + ",   freqmin = " +  str(freqmin) + "\n" )
    g.write  ("\n\n "   + "nax  =" + str(nax) + ",   nvt = " + str(nvt)
     + ",   Detail = " +  str(Detail) + "\n" )
    g.write  ("\n\n "   + "seuil_vt  =" + str(seuil_vt) + "\n" )
#-------------------------------------
#
#---**** First decision: Wordbag or not wordbag? (wordbag is
#       a parameter)
#-------------------------------------
    if(wordbag == 1):
        input_dir_new2 = Path(output_results).resolve()
```

```
        nomfich = Path(chemin).stem
        cheminwb = input_dir_new2/(nomfich + "_wb" + ".txt")
    #   The new wordbag textfile will be in the folder "Results..."
        Wb.wb(chemin, cheminwb)  # Function wb
#-----------------------------------

        print ("\n\n " + "A wordbag file is created: Name = " +
           str(cheminwb)  + "\n" )
#------------------------ (report)
        g.write("\n\n " + "A wordbag file is created: Name = " +
           str(cheminwb)  + "\n" )
#-------------------------
        chemin = cheminwb
#---------------------------------- end if.
#------------------------ (report)
    ligne = "Step: tablex \n------------\n"
    print (ligne)
    g.write("\n" + ligne)
#-------- Computation of the Global Lexical Table (Words x Texts)
#         (function tablex)
    ch_csv = tex.tablex(output_results, chemin, freqmin)
#-----------------------------------
#--------------**** Second decision aggregation or not of the
# texts according to categories (anagreg is a parameter)
#
    if(anagreg == 1):  # will  lead to function "text2tree_Topic()"
#
#----------------- In this option: anagreg = 1, the categories are
# active. They are used to aggregate a priori the texts.
        print ( " Aggregated lexical table   \n ")
#
#-----------------------------------
        text2tree_Topic(ch_csv, output_results, g, freqmin, chemin,
                        sentier, piste, nax, nvt, seuil_vt)
#-----------------------------------
#
    if(anagreg == 0):  # will  lead to function "text2panorama ()"
#
#----------------- In this option: anagreg = 0, the categories are
# supplementary. Their location in the display of the tree will be
# validated through bootstrap replications.
        print ( " Direct analysis of texts (poems, sonnets, etc.  \n ")
#-----------------------------------
        text2panorama (ch_csv, output_results,g, chemin, sentier, piste,
                       freqmin, Detail, nax, nvt, seuil_vt)
#-----------------------------------
        return
#---------------------------------------- [when anagreg = 1]  ---------
#-----------------------------------------------------------------------
#              Function "text2tree_Topic()"
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
def text2tree_Topic(ch_csv, output_results, g, freqmin, chemin, sentier,
```

```
                    piste, nax, nvt, seuil_vt):
#
#  The categories are active. They are used a priori to aggregate the
texts.

#------------------------ (report)
    ligne = "Step: numpy_agreg_new_ok \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#-------------------------------------------
#****  Computation of the aggregated lexical table
#-------------------------------------------
    ch_classic = agreg.agregtablex(ch_csv, output_results, sentier,
     piste)
    lane = ch_classic
#------------------------ (report)
    ligne = "Step: CA_base \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#--------------
#  Classical CA of the aggregated table
#--------------
    X, F, fi, fj, psi, psi_u, phi, phi_u, vp = act.AC_base_AT(
      output_results,lane)
    ligne = "Step: CA_plot: rows + columns \n-----------\n"
    print (ligne); g.write("\n" + ligne)
#
#  Simultaneous graph of aggregated texts and words
#--------------
    ax_h, ax_v = 1, 2
    act.graf_simult (output_results, X, psi, phi, ax_h, ax_v)
    ligne = "Step: CA_plot: rows + columns + char. words\n-----------
\n" ; print (ligne); g.write("\n" + ligne)
#
#  Computation of "test-values" (Selection of characteristic words
# of the categories and texts)
#--------------
    num_vt, nvs = vt.Test_Val12(output_results, nvt, seuil_vt, lane)
#------------------------ (report)
    ligne = "Test-values computed \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#--------------
#  Again: Simultaneous CA graph of aggregated texts and words, with
"nvt" most characteristic words linked to their texts
#--------------
    gr5.grafACcomplet5 (output_results, X, psi, phi, ax_h, ax_v, nvt,
                        num_vt, nvs)
#----------
#  Computation of the AT (Additive Tree) - Nearest Joining Neighbors
#  method. Vertices : categories
#---------- computation of distances
    disMatrix = act.dist(phi, nax)
```

```
#------------------------ (report)
    ligne = "Step: NJ_tree \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#----------
    ident = list(X.columns)
    p = X.shape[1]
 #----------
    ch_edge, ch_gml = njt.AT_NJ_calc(output_results, p, disMatrix,
                                     ident)
#------------------------ (report)
    ligne = "NJ_tree computed \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#----------
# First drawing of the AT
#----------
    id_v, coord_v = tbase.Treebase(ch_gml, output_results, p)
#------------------------ (report)
    ligne = "NJ_tree first plot \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#--------------
# (possible change of first axis orientation)
#--------------
    aa = ax1.change(coord_v, phi)
    coord_v = aa
#------------------------ (report)
    ligne = "Horiz. orientation of NJ_Tree modified \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#--------------
#   Coordinates of words around the AT
#--------------
    coorm = wcp.coor_mots(F, fi, coord_v)
#------------------------ (report)
    ligne = "Words coordinates computed \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#--------------
#  Again: Simultaneous AT graph of aggregated texts and words, with
"nvt" most characteristic words linked to their texts
#--------------
    ct.Tree_simult11 (ch_edge, output_results, F, fi, id_v, coord_v,
                      coorm, nvt, num_vt, nvs)
#------------------------ (report)
    ligne = "Words decorated tree computed and drawn \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#-------
    ligne = "End of the process \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
```

```
#-------------------------
    g.close()
#--------------
    return
#
#--------------
# End of the function "text2tree_Topic"
#--------------
#
#
#[when anagreg = 0]:
#-----------------------------------------------------------------
#-----------------------------------------------------------------
#          Function "text2panorama()"
#-----------------------------------------------------------------
#-----------------------------------------------------------------
#
def text2panorama (ch_csv, output_results, g, chemin, sentier, piste,
                   freqmin, Detail, nax, nvt, seuil_vt):
#
# Reminder: In this option (anagreg = 0) the categories are
# supplementary. Their location in the display of the tree are validated
# with bootstrap replications leading to confidence ellipses.
#-------------------------
# Internal control parameter nboot (should be changed within the code)
#-------------------------
    nboot = 30    # The value of 30 (which may seem low) provides good
# results in this specific multidimensional context. The problem is not
# to estimate the value of a numerical variable. Of course, larger
# values could be tested.
#*****************************
# Internal parameter FOREST (should be changed here, within the code)
# -------------------------
    FOREST = 0   # Producing a "forest" of additive trees
# -------------
# FOREST = 0 :  Without tree visualizations for each category (or topic
# or album)
# FOREST = 1 : (advisable only if nc, number of categories [or albums,
# topics] is, e.g., < 10)
# If FOREST = 1: Sequence of *nc* ATs displays identified by each
# of the *nc* categories (or *nc* albums). Goal: better visualize both
the location and the dispersion of each category in the additive tree.
#-------------------------
    print  ("\n "   + "chemin =" + chemin.name + "  freqmin = " +
     str(freqmin) + "   nvt = " +  str(nvt) +  " nax = " + str(nax) +
     "\n" )
#------------------------ (report)
    g.write("\n\n " + "chemin =" + chemin.name + "  freqmin = " +
     str(freqmin) + "   nvt = " +  str(nvt) +  " nax = " + str(nax) +
     "\n" )
#-----
    ligne = "Step: tablex \n-----------\n"
    print (ligne)
```

```
    g.write("\n" + ligne)
#--------------
    lane = ch_csv  # address of global lexical table Basic_texts x Words
# computed in "CA_AT_synthesis"
#-------------------------------------------------------------
# for some displays : 16 different colors [ first 16] out of 48.
# The maximum number of categories is then < 49, but the following list
# can be extended if necessary
#-------------------------------------------------------------
    colors =
['black','red','green','blue','orange','cyan','steelblue','pink','limegr
een', 'khaki','yellow', 'crimson', 'coral', 'silver', 'olive', 'navy',
'black','red','green','blue','orange','cyan','steelblue','pink','limegre
en', 'khaki','yellow', 'crimson', 'coral', 'silver', 'olive', 'navy',
'black','red','green','blue','orange','cyan','steelblue','pink','limegre
en', 'khaki','yellow', 'crimson', 'coral', 'silver', 'olive', 'navy']
#-------------------------------------------------------------
#tablex has already been saved on files: tablexfile.txt and
tablexfile.csv. Note that tablexfile.csv uses commas as separators (US
convention; for some European countries, commas could be changed into
semicolons, for an XL visualization of the table.
#-------------------------------------------------------------
#------------------------ (report)
    ligne = "Step: CA_base \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
#
# Basic CA : table words x Texts
#------------------------------
# (Reminder: import ac_dtm_AT as act)
    X, F, fi, fj, psi, psi_u, phi, phi_u, vp = act.AC_base_AT(
      output_results, lane)
#--> (Fig CA_1)
#xxxxxxxxxxxxxx
# The coordinates are in the created file: AC_file_tablexfile.csv
#------------------------ (report)
    ligne = "Step: CA_plot: rows + columns \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
# Simultaneous Visualization words x Texts with CA plane(1,2)"
#-------------------------------------------------
    ax_h, ax_v = 1, 2
    act.graf_simult (output_results, X, psi, phi, ax_h, ax_v)
#-------------------------------------------------
#--> (Fig CA_2.1)
#xxxxxxxxxxxxxx
#------------------------ (report)
    ligne = "Step: CA_plot: rows + columns \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
```

```
#
# Simultaneous Visualization words x Texts with CA plane (3,4)"
#-------------------------------------------------
    ax_h, ax_v = 3, 4
    act.graf_simult (output_results, X, psi, phi, ax_h, ax_v)
#--> (Fig CA_2.2)
#xxxxxxxxxxxxxxx
#------------------------ (report)
    ligne = "Step: CA_plot: rows + columns + char. words\n----------\n"
    print (ligne)
    g.write("\n" + ligne)
    ligne = "Computation of Test-values \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
#
# fsize : size of font in some displays
    fsize = 1
#************ below: nvt = number of test-values (characteristic words)
    if(nvt > 0):
        num_vt, nvs = vt.Test_Val12(output_results, nvt, seuil_vt, lane)
#------------------------ (report)
        ligne = "Test-values computed (file: Test_values.txt) \n----\n"
        print (ligne)
        g.write("\n" + ligne)
#------------------------------
#
# Visualization in which characteristic words are linked to their Texts.
#-------------------------------------------------------------------------
#------------------------ (report)
        ligne = "Visualization: texts x Characteristic words \n-----\n"
        print (ligne)
        g.write("\n" + ligne)
#------------------------------
#  (Reminder: import graf_AC_complet5 as gr5)
        ax_h, ax_v = 1, 2
        gr5.grafACcomplet5 (output_results, X, psi, phi, ax_h, ax_v,
                            nvt, num_vt, nvs)
#--> (Fig CA_3.a)
#xxxxxxxxxxxxxxx
        ax_h, ax_v = 3, 4
        gr5.grafACcomplet5 (output_results, X, psi, phi, ax_h, ax_v,
                            nvt, num_vt, nvs)
#--> (Fig CA_3.b)
#xxxxxxxxxxxxxxx
#       Reminder:    n,p = X.shape : n words , p Texts
#*****************
    ident = list(X.columns)
    p = X.shape[1]
    noident = [""]*len(ident)   # liste vide
#---------------------------------
# New identifiers ("albums")for texts (datafile csv: "sentier")
#---------------------------------
```

```
    id_CA_topic, catv = sup_CA.SupCat_CA(sentier)
#------------------------ (report)
    ligne = "Table id_CA_topic \n-----------------\n"
    print (ligne)
    g.write("\n" + ligne)
    g.write( str(id_CA_topic))
#-----------
    ligne = "catv = vector of albums/topics numbers \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
# Coordinates of categories (albums/topics) on axes up to 4
    naxtot = 4
    coord_album_CA, nalbum = calb.coordalbums(catv, phi, naxtot)
#------------------------------
    import ident_alb as new_id
#------------------------------
    idalb = new_id.identif_alb(piste)
#------------------------------
    categ = "_basic ident."
#  (Reminder: import CA_cat_alb as CA_alb)
#
    ax_h, ax_v = 1, 2
    CA_alb.CA_catalb (output_results, categ, catv, ident, phi, fsize,
coord_album_CA, nalbum, idalb, ax_h, ax_v))
#--> (Fig CA_4.a) (axes (1,2)
#xxxxxxxxxxxxxxxxxxxxxxx
#
    ax_h, ax_v = 3, 4
    CA_alb1.CA_catalb1 (output_results, categ, catv, ident, phi, fsize,
coord_album_CA1, nalbum, idalb, ax_h, ax_v))
#--> (Fig CA_4.b) (axes (3,4)
#xxxxxxxxxxxxxxxxxxxxxxx
#
#---------- Optional change within the code: ------------
# The convex hulls of supplementary categories (here: albums/topics)
# emphasize the fact that supplementary categories are DILATED centroids
# of the concerned texts (poems, sonnets, topics...). They could be
# outside the convex hulls of the concerned texts.
#---- (if ok with option, remove the 4 next "#" below to execute)
#    ligne = "Convex Hulls of albums (main CA plane) \n-----------\n"
#    print (ligne)
#    g.write("\n" + ligne)
#    CA_hulls.CA_catalb_hulls (output_results, catv, id_CA_topic, phi,
fsize, coord_album_CA, nalbum, idalb, categ, colors)
#-------------------------------------------------------------
#
# *** BOOTSTRAP ********************** Replicates  ****
#*************************************************
#
#-------------------------------------------------------------
#------------------------ (report)
    ligne = "Replicates of albums \n-----------------\n"
```

```
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
    pboot, pboot_n, pboot_list = Dup.Dupli_csv2 (output_results, nboot,
sentier)
#**************************************
#------------------------ (report)
    ligne = "Coordinates of albums + replicates \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
    coord_album_replic, nalbum,  ntot = calboot.coordalbums_boot(
           output_results, catv, phi, nboot)
#------------------------ (report)
    ligne = "Identifiers including replicates \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
    idalboot, idalboot_num = talboot.create_idalboot2(nalbum,
                                nboot, piste)
#
#------------------------------
# Reminder: "Categories of texts", "albums", "topics", designates the
# same categorical variables in the present code.
#------------------------ (report)
    ligne = "Graph with replicates of albums, and albums" \n-----
    ------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
#  (Reminder: import CA_cat_alb_boot as catboot)
#------------------------------
#  Graph with replicates of albums, and albums
#------------------------------
    xh_album, xv_album = catboot.CAcatalboot (output_results, catv,
ident, phi, fsize, coord_album_replic, nalbum, ntot, idalboot)
#--> (Fig CA_5)
#xxxxxxxxxxxxx
    catboot.Print_coord_albS_CA(output_results, xh_album, xv_album,
idalboot )
#--------------------
    nreplic = 1  # presence of replicates in the graphical displays
    nid = 0
    categ = "(nreplic=1_nid =0_basic_ident)"
#  (Reminder: import CA_cat_alb_boot_ellipses as catbootell)
#
    list_all = [i for i in range(nalbum)] # Ellipses for all albums
#
    selec_list = list_all
#--------------------
# Confidence ellipses with replicates and basic text identifiers
#--------------------
```

```
    xh_album, xv_album = catbootell.CAcatalbootellipses (output_results,
selec_list, catv, nid, ident, phi, fsize, coord_album_replic, nalbum,
nboot, idalboot, categ, nreplic, colors)
#
#-->(Fig CA_6_1 - basic identifiers)
#xxxxxxxxxxxxxxx
    nreplic = 1 ; nid = 0
#-----------------
#
    categ = "(nreplic = 1, nid = 0, ident_topic)"
#--------------------
# Confidence ellipses with replicates and category identifiers
#--------------------
    xh_album, xv_album = catbootell.CAcatalbootellipses (output_results,
selec_list, catv, nid, id_CA_topic, phi, fsize, coord_album_replic,
nalbum, nboot, idalboot, categ, nreplic, colors)
#
#--> (Fig CA_6_2 - identifiers = category numbers- ident_topic)
#xxxxxxxxxxxxxxx
#----------- Optional change within the code: -------------
#   xxxxxxx Call selection of 2 ellipses xxxxxx *** Internal option.
#  Example of categories (topics, albums) 6 and 7.
#  Selection of 2 ellipses from a list: here: [5,6] = topics 6 and 7
#  (remember, origin = 0, instead of 1).
#  This particular selection concerns only the Shakespeare's Sonnets
#  example. To be modified according to possible external information.
#-------------------------------------
    selec_list = [5,6]  #  (Albums/topics 6 and 7)
#-------------------------------------
    categ = "(_selection_of_2_ellipses_replicates)"
    xh_album, xv_album = catbootell.CAcatalbootellipses (output_results,
selec_list, catv, nid, id_CA_topic, phi, fsize, coord_album_replic,
nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig CA_6_2_opt1 - identifiers = category numbers- ident_topic)
#xxxxxxxxxxxxxxx
    nreplic = 0
    categ = "(_selection_of_2_ellipses_no_replicates)"
    xh_album, xv_album = catbootell.CAcatalbootellipses (output_results,
selec_list, catv, nid, id_CA_topic, phi, fsize, coord_album_replic,
nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig CA_6_2_opt2 - identifiers = category numbers- ident_topic)
#xxxxxxxxxxxxxxx
#-----------------------------------------------------------
#
    nreplic = 1 ; nid = 1 ; fsize = 1
    categ = "(nreplic = 1, nid =0, basic_ident)"
#
#   ********* Convex hulls of replicates *************
#  (Reminder: import CA_cat_alb_boot_hulls as CA_boot_hulls)
#-------------------------------------
    xh_album, xv_album = CA_boot_hulls.CAcatalboothulls (output_results,
catv, nid, ident, phi, fsize, coord_album_replic, nalbum, nboot,
idalboot, idalboot_num, categ, nreplic, colors)
```

```
#--> (Fig CA_7)
#xxxxxxxxxxxxx
#-----------------------------------

#------------------------ (report)
    ligne = "End of CA computations and visualizations \n----------\n"
    print (ligne)
    g.write("\n" + ligne)
#**********************************
#------------------------ (report)
    ligne = "Simultaneous Additive Trees (AT), computations and
visualizations \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#
#-------------------------------------------------------------
# Section  Additive Trees    AT  AT  AT  AT  AT  AT  AT ---------
#-------------------------------------------------------------
#
# Computation of the distances matrice (distances between texts)
#------------------------ (report)
    ligne = "Computation of distances matrix\n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#-----------------------------
#
    disMatrix = act.dist(phi, nax)
    ident = list(X.columns)
    p = X.shape[1]
#------------------------ (report)
    ligne = "Step: Neighbours Joining_Tree \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#-----------------------------
# Computation of the (Additive Tree) - Nearest Joining Neighbors method.
#----------
    ch_edge, ch_gml = njt.AT_NJ_calc(output_results, p, disMatrix,
                                     ident)
#------------------------ (report)
    ligne = "NJ_tree computed \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
    ligne = "NJ_tree first plot (+ coord_v + id_v) \n--------------\n"
    print (ligne)
    g.write("\n" + ligne)
#-----------------------------
#
# First drawing of the AT.  ( id_v = ident of vertices,
#  coord_v = coordinates of vertices)
#----------
    id_v, coord_v = tbase.Treebase(ch_gml, output_results, p)
#----------
#--> (Fig Tree_1_Treebase)
```

```
#xxxxxxxxxxxxxxxxxxxxxx
#


#------------------------ (report)
    ligne = "New ident for vertices\n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
# New identifiers of categories ("albums") for texts (file csv:
"sentier")
    id_AT_topic, catv = sup_AT.SupCat_AT (sentier)
#
#------------------------ (report)
    ligne = "Table / id_AT_topic \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
# Copy of identifiers
    g.write( str(id_AT_topic))
#**********************************
#------------------------ (report)
    ligne = "catv = vector of albums/topics \n----------\n"
    print (ligne); g.write("\n" + ligne)
    g.write(str(catv))
    ligne = "Same tree with new idents \n----------------\n"
    print (ligne); g.write("\n" + ligne)
#------------------------------
#*****************************
# Same AT with ident (topics/albums/categories) instead of basic texts
idents [sonnets/songs/poetries])
    cat.Treecat (output_results, p, id_AT_topic, coord_v)
#--> (Fig Tree_2_Treebase_ident_cat)
#xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#-----------------------------------
#------------------------ (report)
    ligne = "Coordinates on AT of albums (texts) \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#-----------------------------------
    naxtot = 2  # only 2 dimensions (AT is planar)
    coord_album_AT, nalbum = calb.coordalbums(catv, coord_v, naxtot)
#-----------------------------------
    import ident_alb as new_id
#-----------------------------------
    idalb = new_id.identif_alb(piste)
# ***************
#------------------------ (report)
    ligne = "Basic identifiers for Additive Trees (ident complemented
with blank spaces) \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
```

```
    ident_AT = ident + [""]*(len(id_AT_topic) - len(ident))
    g.write(str(ident_AT))
#-----------------------------------
# Same AT with idents "albums" (file csv : piste_boot) + replicates
#------------------------ (report)
    ligne = "Coordinates of albums + replicates \n-----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#-------------------------------
# (reminder: calboot ...for module: coord_alb_boot.py in the library)
    coord_album_replic_AT, nalbum,  ntot =
calboot.coordalbums_boot(output_results, catv, coord_v, nboot)
#
    categ = "(basic_ident)" ; fsize = 1
#-----------------------------------
    nreplic = 0
#------------------------ (report)
    ligne = "Tree with replicates of albums, and albums (texts) \n------
----------\n"
    print (ligne)
    g.write("\n" + ligne)
#-------------------------------
# (reminder: talboot, for module : Tree_cat_alb_boot.py in the library)
#-------------------------------
    xh_album, xv_album = talboot.Treecatalboot (output_results, categ,
catv, ident_AT, coord_v, fsize, coord_album_replic_AT, nalbum, ntot,
idalboot, nreplic)
#--> (Fig Tree_3_1)_Treebase + categories_sup)
#xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    nreplic = 1
#-------------------------------
    xh_album, xv_album = talboot.Treecatalboot (output_results, categ,
catv, ident_AT, coord_v, fsize, coord_album_replic_AT, nalbum, ntot,
idalboot, nreplic)
#--> (Fig Tree_3_2)_Treebase + categ_sup + replicates)
#xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    talboot.Print_coord_albST(output_results, xh_album, xv_album,
idalboot )
#-----------------------------------
    nreplic = 1
    nid = 0
#-------------------------------
    if(Detail > 0):
        categ = "1_Boot_Conf_Ellipses_(nreplic_=_1,_ident_topic)"
#  Conf. ellipses, identifiers: categories
        list_all = [i for i in range(nalbum)]
        selec_list = list_all #(all categories are selected)
#-------------------------------
        xh_album, xv_album = talboot_ellipses.Treecatalbootellipses
(output_results, selec_list, catv, nid, id_AT_topic, coord_v, fsize,
coord_album_replic_AT, nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig Tree_4_1)
#xxxxxxxxxxxxxxxxx
```

```
#-----------------------------------
        nid = 0
        nreplic = 0
        categ = "2_Boot_Conf_Ellipses_(nreplic_=_0,_basic_ident)"
        fsize = 3
#  Conf. ellipses, identifiers: basic texts
#-----------------------------------
        xh_album, xv_album = talboot_ellipses.Treecatalbootellipses
(output_results, selec_list, catv, nid, ident_AT, coord_v, fsize,
coord_album_replic_AT, nalbum, nboot, idalboot, categ, nreplic, colors)
#-----------------------------------
    nreplic = 0; nid = 0
#--> (Fig Tree_4_2)
#xxxxxxxxxxxxxxxxx
#-----------------------------------
    categ = "3_Boot_Conf_Ellipses_(nreplic_=_0,_nid_=_0,_ident_topic)"
#  Conf. ellipses, identifiers: categories, no replicates
#-----------------------------------
    xh_album, xv_album = talboot_ellipses.Treecatalbootellipses
(output_results, selec_list, catv, nid, id_AT_topic, coord_v, fsize,
coord_album_replic_AT, nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig Tree_4_3)
#xxxxxxxxxxxxxxxxx
#-----------------------------------
#     Convex hulls of replicates.
#-----------------------------------
    nreplic = 1
    categ = "_(basic_ident)"
#-----------------------------------
    TBH.Treecatalboot_hulls (output_results, catv, nid, ident_AT,
coord_v, fsize, coord_album_replic_AT, nalbum, nboot, idalboot,
idalboot_num, categ, nreplic, colors)
#--> (Fig Tree_5)
#xxxxxxxxxxxxxxx
#-------------------------------------------------------------
#---------- Optional change within the code: ------------
#*********************************
#  call with select: Internal option (intervention within the code).
#  Selection of 2 ellipses from a list: As an example: [5,6] =
confidence ellipses for albums (or topics) 6 and 7 (remember, origin =
0, instead of 1). This is just an example of selection of a pair of
albums (to compare CA and AT ellipses, case of Shakespeare sonnets)
#-----------------------------------
    selec_list = [5,6]  #  (Albums 6 and 7)
    categ = "4_(ident_basic_Selection_of_2_ellipses)"
#-----------------------------------
    xh_album, xv_album = talboot_ellipses.Treecatalbootellipses
(output_results, selec_list, catv, nid, ident_AT, coord_v, fsize,
coord_album_replic_AT, nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig Tree_5.a)
#xxxxxxxxxxxxxxxxx
#----------------------
    categ = "5_(ident_basic_Select_2_ellipses_no_replicates)"
```

```
    nreplic = 0
    xh_album, xv_album = talboot_ellipses.Treecatalbootellipses
(output_results, selec_list, catv, nid, ident_AT, coord_v, fsize,
coord_album_replic_AT, nalbum, nboot, idalboot, categ, nreplic, colors)
#--> (Fig Tree_5.b)
#xxxxxxxxxxxxxxxxx
#
#****************************** FOREST: internal parameter
# Reminder: If FOREST = 1: Sequence of *nc* ATs successively identified
by each of the *nc* categories (*nc* albums). Goal: visualizing both the
position and the dispersion of each category within the tree
#******************************
#
    if(FOREST < 0.5):
#------------------------ (report)
        ligne = "End of the process \n-----------\n"
        print (ligne)
        g.write("\n" + ligne)
        g.close()
#-------------------------------------------
        return
#------------------------------
#------------------------ (report)
    ligne = "Series of tree with new idents \n----------------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
    fo.forest(output_results, catv, id_AT_topic, p, coord_v)
#------------------------ (report)
    ligne = "End of the process \n-----------\n"
    print (ligne)
    g.write("\n" + ligne)
#------------------------------
    g.close()
#-------------------------------------------
    return
#-------------------------------------------
#----- END OF:  TexPanorama.py
#-------------------------------------------
```