
bilbo2

Release 1.0.0

Oct 11, 2021

1 Purpose	3
2 Demonstration	5
3 Requirements	7
4 Installation	9
5 First steps	11
6 Input - Output	13
7 Pipelines	15
8 Chaining algorithms	17
9 CLI toolkit usage	19
10 Bilbo in a shell	21
11 Annotator bilbo usage	27
12 Configuration File options	29
13 Knowledge Base	35
14 Evaluation	37
15 Resultat	39
16 bilbo	41
17 Indices and tables	55
Python Module Index	57
Index	59

BILBO2 is an open source software for automatic annotation of bibliographic reference. It provides the segmentation and tagging of input string. Its main purpose is to provide both a complete development and research space for the improvement of bibliographic reference detections and to be a solid tool capable of being used in production like **OpenEdition** for example. What you will find here is the user documentation, the technical documentation and the developer documentation for the Bilbo software.

This documentation is organized into a few main sections :

- *Purpose*
- *Getting started*
- *Essential Things To Know*
- *Usage*
- *Configuration*
- *Developer*

1.1 Context

In academic papers, bibliographies are an essential aspect of research. In many cases, only few bibliographic references are identified by an information system.

We can consider that there are three levels of detections of bibliographic references:

- Standard bibliographic references: These are usually located at the end of the scientific article. They are mentioned in a TEI-XML document by the tag `<bibl>`
- Footnote: Footnotes do not appear every time a bibliography is present. The classification of notes that contain bibliographies or not is the main difficulty. Implemented by the tag `<note>` (TEI-XML Document).
- Implicit reference. In the full text, authors sometimes mention a bibliographic reference. This reference is partial and explicit. Implemented by the tag `<p>` (TEI-XML Document).

1.2 Philosophy

Algorithm complexity to extract bibliography is increasing at each level (`<bibl>`, `<note>`, `<p>`). Currently, the first level could be considered as efficient. The others are not. Bilbo2 is considered to be dedicated to research.

It has been thought to be a tool for easily implementing new machine learning algorithms at any level of bibliography. Everything has been done so that we can easily add new algorithms to existing codes without affecting what can be deployed in production.

Data structure of a document is constructed to manipulate a Document at any level. A level (that we call a section) is a tag. The scope of the processing algorithm will be the section chosen. Then all sections corresponding to this tag will be processed.

Each instance of Bilbo is specialized in a specific tag. To handle different types of tags, you can pass and process each time the XML document.

2.1 SIMPLE WEB API

There is a online demonstration of Bilbo2

- <http://openedition.huma-num.fr/bilbo-api/>

In this demonstrator, only bibliographic reference and footnote is processing. The corpus is trained on french and english dataset. Just paste each bibliographic references surrounded by tag `<bibl>` or `<note>` and click on annotate. You already have pre-loaded examples.

This instance is for test purposes only and should therefore not be used as production tools. For a production usage you should contact mathieu.orban@openedition.org. The data processing is not kept at all. In the future, a full REST web-API is programming. This REST-API should integrate the research of Digital Object Identifier (DOI) for each bibliographics references processed.

2.2 Command Line Interface DEMO

For an overview of different features and CLI command just launch in a shell :

```
cd bilbo2
bash tools/demo_cli.sh -v
```


Bilbo2 can be installed on many Linux distributions but has been tested only on debian and ubuntu distributions with the following prerequisites:

- **python3.5**
- **git** $\geq 1.7.10$ (needed by github)
- **pip** and **setuptools** , necessary for launch python installation

3.1 Libraries

Bilbo2 was tested on Linux/Debian distribution (Debian stretch release). It is running on python versions equal to or greater than 3.5.

3.1.1 Debian

Starting from a [debian image](#) loaded in a virtual machine, with root privileges or via `sudo`.

```
apt-get install git make curl python3 build-essential libxml2-dev libxslt-dev python3-  
↳dev python3-setuptools zlib1g-dev
```


Make sure you full filled the [requirements](#) before going any further.

You can now running the installation of python module with setup.py. It will :

- Install **LIBSVM**. A python binding to libsvm.
- Install **python-crfsuite**. A smart python binding to crfsuite.
- Install **lxml**, python libraries to process xml document.

4.1 Stable version

To install the last stable version on a Unix system, open a console and enter:

```
git clone https://github.com/openedition/bilbo2.git
cd bilbo2
git checkout `git describe --tags --abbrev=0`
python3 setup.py install --user
```

4.2 Development version

If you wish to install the development version, open a console and enter:

```
git clone https://github.com/openedition/bilbo2.git
cd bilbo2
python3 setup.py install --user
```

4.3 Uninstall bilbo2

For uninstall bilbo2:

```
pip3 uninstall bilbo2
```

For remove and clean your local bilbo2 repositories:

```
cd bilbo2
rm -rvf build/
rm -rvf bilbo2.egg-info/
rm -rvf dist/
```

Or you can use directly bash script clean.sh.

```
./clean.sh
```

Note that you have to replace the **right path** to your repository in the clean.sh script.

4.4 Optional libraries

If you have `jenkins` installed and you wish to make a continuous integration testing, you need to install a optional libraries to generate xml report. It will fit python unittest report to jenkins format specification :

```
pip3 install unittest-xml-reporting
```

Keep in mind that bilbo has already been trained on a french and english annotated xml corpus. It is trained on `<note>` and `<bibl>` section. By default, bilbo2 is running (for annotation) on a specified pipeline with a default pre-trained model (french and english languages on `<bibl>` tag).

5.1 Command Line Interface API

5.1.1 Overview Command Line Interface API

For an overview of different features and CLI command just launch in a shell :

```
cd bilbo2
bash tools/demo_cli.sh -v
```

5.1.2 Common use

You will see that a quick use, to annotate your bibliographics references (indicate as `<bibl>` in TEI) just launch :

```
cd bilbo2
python3 -m bilbo.bilbo --action tag -i PATH_TO_XMLFILE -o XML_OUTPUT_TAGGED
```

For annotate your footnote you need first to mention the tag to process (note) and specify explicitly the config file. Currently the config file `pipeline_note.cfg` is available.

```
cd bilbo2
python3 -m bilbo.bilbo --action tag -t note -c bilbo/config/pipeline_note.cfg -i PATH_
↪TO_XMLFILE -o XML_OUTPUT_TAGGED
```

For train, you just have to change `--action=tag` to `--action=train` and given an annotated input xml corpus. Note that output option is not necessary in this case. Saved trained model will be saved at the path indicated in the config file.

5.2 Interactive Python Interface

Open a terminal:

```
python3
```

In a interactive python shell:

```
from bilbo.importer import Importer
from bilbo.bilbo import Bilbo

importer = Importer("path_to_your_xml_file")
doc = importer.parse_xml('tag(bibl or /note)')
bilb = Bilbo(doc, "path_config_file")
bilb.annotate("path_to_output.xml", format_=None)
```

5.2.1 Autoloaded Models

Two models and prexisted external list already exists and can be loaded as a data package. You do not need to give the config file path. You just have to load 'bibl' or 'note' with class method load(bibl/note):

```
from bilbo.importer import Importer
from bilbo.bilbo import Bilbo

importer = Importer("path_to_your_xml_file")
doc = importer.parse_xml('note')
Bilbo.load('note')
bilbo = Bilbo(doc)
bilb.annotate("path_to_output.xml", format_=None)
```

6.1 Input

Input must be a valid XML document. Bilbo has been trained on **TEI-XML** format (Text Exchange Initiative) format. But you could give a **JATS** input format. . . Actually, any valid XML document can be used for bilbo, the annotation tagging will be done according to TEI schemas.

6.1.1 Scope of annotation

Bilbo is only handling a scope inside a xml. This scope is bounded by a tag. Element outside the scope are only kept in memory to rebuild the xml file as a result.

6.2 Output

Output is XML. Bilbo has personal output for research purpose: this output schema is the default xml output. Any output schema can be specified from this default schema. For this, an xsl sheet must contain the xml conversion. This xsl file should be placed in `bilbo/stylesheets/` directory. You can specify TEI (Text Exchange Initiative), JATS format and personal research.

6.2.1 XML/TEI OpenEdition Schema

The TEI schema version is used by the OpenEdition Books and OpenEdition Journals platforms. It is associated to the journals editorial model shipped with the Lodel software <https://github.com/OpenEdition/lodel>

Among the different XML encoding standards for machine-readable texts, the TEI (**Text Encoding Initiative**) is probably the most comprehensive and mature. The TEI Guidelines define some 500 different textual components and concepts (word, sentence, character, glyph, person, etc.). Any particular usage of the TEI supposes a customization of the TEI to their specificities, so as to adapt and constraint the richness of the TEI to a well scoped and tuned schema. The TEI community has created a specification language called ODD ("**One Document Does it all**") to modify the

general TEI schema. Having ODD descriptions, it is possible with a tool called Roma to generate automatically customised TEI schemas (xsd, relaxNG, etc.) and some documentation.

6.2.2 JATS

6.2.3 Personnal research output

This is NOT A VALID XML/TEI schemas. It is only set for research purpose. Each added tag is mentionned by an attribute `bilbo="true"`. It could be used to analyse in a same xml document difference between manual annotated and automatic annotation.

Bilbo is founded on processing a specific structure (data struct) on series. The Input and ouptput of each data is stable. In some case the output is enhanced from an information on each section or at the level of a token.

7.1 Importer

Document is imported: lxml library parse the whole document. Each document is segmented according to one section, the data structure is constructed at the level of section.

7.2 Module available

All this module can be found in components directory. Each class will inherit from the Component Class.

7.2.1 Shape Data: extract XML value and tokenizer

Shaper section is dedicated to handle xml data and tokenize. Tokenizer is written for french and english. structure is constructed at the level of token. This module is certainly and should be the first in the pipelines series. For see CLI API functionalities (CLI):

```
python3 -m bilbo.components.shape_data -h
```

For specify shape component options

7.2.2 Features

Features could be extract from list or dictionaries files (external features). Features could be extract from the local specificity of a word. Features could be extract from the specificity of section or position of a token (global features). List of word could be simple or multiple. For see CLI API functionalities (CLI):

```
python3 -m bilbo.components.features -h
```

For specify feature component options

7.2.3 Conditional random field

This is based on on python-crfsuite Pythons crf-suite is a pyhon binding of CRFSuite. CRFSuite is an implementation of Conditional Random Fields (CRFs) for labeling sequential data. This labelling is generated by an extraction of feature and to get easier an wrapping with crf++ is avalaible. For see CLI API functionalities (CLI):

```
python3 -m bilbo.components.crf -h
```

For specify crf component options

7.2.4 Support Vector Machine

This is based on libsvm. LIBSVM is an integrated library for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification. For now, it is used to classify foot note which are contains bibliography. For see CLI API functionalities (CLI):

```
python3 -m bilbo.components.svm -h
```

For specify svm component options

7.3 Optionnal output step

Each document is segmented according to one section: the data xml structure is reconstructed.

Chaining algorithms

For chained Algorithms with action train, tag or evaluate you need to specified parameters in a configuration file. Two default configuration files are defined in *./bilbo/config/* directory. The order of data processing must be clarified. For each pipe you need to fulfilled the algorithm specification.

For specify order and used algorithms see [pipeline options](#)

Each pipe is launching a component. At each passage, document object is enhanced from differents attribute. Attribute can be extract (extractor component class) or predict (estimator component class)

CLI toolkit usage

If you want to start and understand how bilbo is handling each pipeline, you can launch independently some test on bilbo component. As above you can do in CLI or in Interactive Python. Beware input of each component. Examples: it does not make sense to launch CRF modules if you have not extract features previously (in a file or in bilbo data structure).

9.1 Overview

For an overview and a test of cli usage, from a terminal, run:

```
cd bilbo2
/bin/bash tools/demo_cli.sh -v
```

You can add `-v` argument to see output.

9.2 Command Line Interface API

To see an exhaustive list of modules which could be used by bilbo you can launch

```
python3 -m bilbo.bilbo -L
```

For instance, you should see which features are extracted for each token according to the default configuration file. Your features (crf++ format) will be extracted in the output file mentioned in "bilbo/config/pipeline_bibl.cfg" If you want to see explicit output just add `-vvvv` for a logger output.

```
python3 -m bilbo.components.features -cf bilbo/config/pipeline_bibl.cfg -s "Amblard F.
↪, Bommel P., Rouchier J., 2007, « Assessment and validation of multi-agent models ».
↪..."
```

In order to improve your research you want to analyse directly from a crf++ input format and crf++ pattern your prediction.

```
python3 -m bilbo.components.crf -cf bilbo/config/pipeline_bibl.cfg -i bilbo/testFiles/  
↳features.output.txt --tag -v
```

```
import configparser  
from bilbo.importer import Importer  
from bilbo.components.shape_data.shape_data import ShapeSection  
from bilbo.components.features.features import FeatureHandler  
from bilbo.components.crf.crf import Crf  
from bilbo.bilbo import Bilbo
```

10.1 Construct Data Structure

First import your xml document. You can import string or a file. For any action (machine learning prediction, features extraction, set a new xml properties), you will handle this document object.

```
#xml_str = '<xml>Oustide<bibl><pubPlace>Marseille</pubPlace>, <sponsor>OpenEdition is
↳"! inside </sponsor>>a bibl</bibl></xml>'
xml_str = """<TEI xmlns="http://www.tei-c.org/ns/1.0"> Outside
<bibl>Hillier B., 1996, <hi>Space is the Machine</hi>, Cambridge University Press,
↳<pubPlace>Cambridge.</pubPlace>
</bibl></TEI>"""
imp = Importer(xml_str)
doc = imp.parse_xml('bibl', is_file = False)
```

10.2 Tokenize, extract and wrap xml informations

First, load parameters.

```
dic = """
[shaper]
tagsOptions = {
↳
    "pubPlace": "place",
    "sponsor": "publisher"
}
verbose = True
"""
#Load the dic.
#There are differnt ways to set parameters (ini file...)see: https://docs.python.org/3/library/configparser.html#quick-start
↳
```

(continues on next page)

(continued from previous page)

```
config = configparser.ConfigParser(allow_no_value=True)
config.read_string(dic)
```

Use ShapeSection class. Note at any moment you can call help for parameters function:

```
help(ShapeSection.__init__)
```

Help on function `__init__` in module `bilbo.components.shape_data.shape_data`:

```
__init__(self, cfg_file, type_config='ini', lang='fr')
    Initialize self. See help(type(self)) for accurate signature.
```

```
sh = ShapeSection(config, type_config='Dict')
sh.transform(doc)
```

```
<bilbo.storage.document.Document at 0x7fc3740d7390>
```

To see an overview of your document:

```
for section in doc.sections:
    for token in section.tokens:
        print('Token:{0}\t\t Label:{1}'.format(token.str_value, token.label))
```

```
Token:Hillier           Label:biBl
Token:B.                Label:biBl
Token:,                 Label:c
Token:1996              Label:biBl
Token:,                 Label:c
Token:Space             Label:hi
Token:is                Label:hi
Token:the                Label:hi
Token:Machine           Label:hi
Token:,                 Label:c
Token:Cambridge         Label:biBl
Token:University        Label:biBl
Token:Press             Label:biBl
Token:,                 Label:c
Token:Cambridge         Label:place
Token:.                 Label:c
```

10.3 Features

Set features that you are needed. For external features, you need to give the right path to externals lists...

```
dic = """
[features]
listFeatures = numbersMixed, cap, dash, biblPosition, initial
listFeaturesRegex = ('UNIVERSITY', '^Uni.*ty$')
listFeaturesExternes = ('surname', 'surname_list.txt', 'simple'),
listFeaturesXML = italic
output = output.txt
verbose = False
```

(continues on next page)

(continued from previous page)

```
"""
config = configparser.ConfigParser(allow_no_value=True)
config.read_string(dic)
```

Features are given for convenience in Crf++ format.

```
feat = FeatureHandler(config, type_config='Dict')
feat.loadFonctionsFeatures()
doc = feat.transform(doc)
feat.print_features(doc)
```

```
Hillier NONUMBERS FIRSTCAP NODASH BIBL_START NOINITIAL NOUNIVERSITY SURNAME NOITALIC_
↳bibl

B. NONUMBERS ALLCAP NODASH BIBL_START INITIAL NOUNIVERSITY NOSURNAME NOITALIC bibl
, NONUMBERS NONIMPCAP NODASH BIBL_START NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC c
1996 NUMBERS NONIMPCAP NODASH BIBL_START NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC_
↳bibl
, NONUMBERS NONIMPCAP NODASH BIBL_START NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC c

Space NONUMBERS FIRSTCAP NODASH BIBL_START NOINITIAL NOUNIVERSITY NOSURNAME ITALIC hi
is NONUMBERS ALLSMALL NODASH BIBL_IN NOINITIAL NOUNIVERSITY NOSURNAME ITALIC hi
the NONUMBERS ALLSMALL NODASH BIBL_IN NOINITIAL NOUNIVERSITY NOSURNAME ITALIC hi
Machine NONUMBERS FIRSTCAP NODASH BIBL_IN NOINITIAL NOUNIVERSITY NOSURNAME ITALIC hi
, NONUMBERS NONIMPCAP NODASH BIBL_IN NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC c

Cambridge NONUMBERS FIRSTCAP NODASH BIBL_IN NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC_
↳bibl

University NONUMBERS FIRSTCAP NODASH BIBL_END NOINITIAL UNIVERSITY NOSURNAME NOITALIC_
↳bibl

Press NONUMBERS FIRSTCAP NODASH BIBL_END NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC_
↳bibl
, NONUMBERS NONIMPCAP NODASH BIBL_END NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC c

Cambridge NONUMBERS FIRSTCAP NODASH BIBL_END NOINITIAL NOUNIVERSITY NOSURNAME_
↳NOITALIC place

. NONUMBERS NONIMPCAP NODASH BIBL_END NOINITIAL NOUNIVERSITY NOSURNAME NOITALIC c
```

10.4 Make predictions

First, to get an Document storage object which make sense (not as above, just for demonstration usage). We load right parameters with path_pipeline_bibl:

```
# This part is a fast resume of TOKENIZER AND FEATURE explain above.
# There are runned again with the appropriate parameter (path to pipeline_bibl.cfg).
imp = Importer(xml_str)
doc = imp.parse_xml('bibl', is_file = False)
bbo = Bilbo(doc, 'pipeline_bibl.cfg')
bbo.shape_data(doc)
bbo.features(doc)
```

```
<bilbo.storage.document.Document at 0x7fc3740ac828>
```

We have now a Document storage object which contains all needed information

```
# Start to make predictions
tagger = Crf(bbo.config, type_config='Dict')
labels = tagger.predict(doc)

for label in labels:
    for l in label:
        print(l[0], l[1])
```

```
Hillier surname
B. forename
, c
1996 date
, c
Space title
is title
the title
Machine title
, c
Cambridge publisher
University publisher
Press publisher
, c
Cambridge pubPlace
. c
```

10.5 Add prediction at the data structure

Always use transform() function for added prediction to Document storage object. Note for estimator component, three option are availables : 'tag', 'train', 'evaluate'

```
tagger.transform(doc, 'tag')
```

```
for section in doc.sections:
    for token in section.tokens:
        print('Token:{0}\t\t Label:{1}'.format(token.str_value, token.predict_label))
```

```
Token:Hillier           Label:surname
Token:B.                Label:forename
Token:,                 Label:c
Token:1996              Label:date
Token:,                 Label:c
```

(continues on next page)

(continued from previous page)

Token:Space	Label:title
Token: is	Label:title
Token:the	Label:title
Token:Machine	Label:title
Token:,	Label:c
Token:Cambridge	Label:publisher
Token:University	Label:publisher
Token:Press	Label:publisher
Token:,	Label:c
Token:Cambridge	Label:pubPlace
Token:.	Label:c

11.1 For bibliography (Standard tagging)

```
imp = Importer('resources/corpus/bibl/test_bibl.xml')
doc = imp.parse_xml('bibl')

Bilbo.load('bibl')

bilbo = Bilbo(doc)
bilbo.run_pipeline('tag', '/tmp/output.xml', format_= None)
```

11.2 For bibliography (With Lang Detection tagging)

```
imp = Importer('resources/corpus/bibl/test_bibl.xml')
doc = imp.parse_xml('bibl')

Bilbo.load('bibl_lang')

bilbo = Bilbo(doc)
bilbo.run_pipeline('tag', '/tmp/output.xml', format_= None)
```

11.3 For note

```
imp = Importer('resources/corpus/note/test_note.xml')
doc = imp.parse_xml('note')
bilbo = Bilbo(doc, 'pipeline_note.cfg')
bilbo.run_pipeline('tag', '/tmp/output.xml', format_= None)
```

11.4 Train

Just modify tag parameter to train parameter!! Note: output could be some binaries constructed model (They must be specified in pipeline_bibl.cfg not as parameters in run_pipeline() function.

11.5 Evaluation (end to end)

For evaluate the models just launch bilbo on your datatest annotated as:

```
imp = Importer('resources/corpus/bibl/data_test.xml')
doc = imp.parse_xml('bibl')
bilbo = Bilbo(doc, 'pipeline_bibl.cfg')
bilbo.run_pipeline('evaluate', None, None)
```

label	precision	rappel	f-measure	occurences
abbr	0.874	0.765	0.816	452
biblScope	0.887	0.571	0.695	594
booktitle	0.903	0.629	0.742	89
date	0.716	0.915	0.803	614
edition	0.690	0.460	0.552	126
emph	1.000	1.000	1.000	2
extent	1.000	0.979	0.989	48
forename	0.929	0.956	0.942	942
genName	1.000	1.000	1.000	1
journal	0.823	0.732	0.774	514
nameLink	0.282	1.000	0.440	11
orgName	0.902	0.836	0.868	110
place	0.824	0.933	0.875	15
pubPlace	0.962	0.934	0.948	379
publisher	0.936	0.732	0.821	920
ref	1.000	0.071	0.133	14
surname	0.937	0.934	0.936	823
title	0.868	0.889	0.879	5740
mean	0.863	0.797	0.828	11394
weighted mean	0.877	0.852	0.864	11394

11.6 Evaluation by component

You can evaluate each component. In this case we use bilbo as toolkit usage. Load your annotated data : data format annotated is depended of component used. You have to always generate this data first. And just launch (for svm for instance)

```
svm.evaluate(input_svm_data_format)
```

Configuration File options

Bilbo comes with a `pipeline_config` file (located at the `bilbo/config` of the `bilbo2` directory). Actually, there is two `pipeline_config` available, one is for annotating bibliographies references (tag `<bibl>` in the TEI/XML format), one other is for annotating footnote (tag `note` in the TEI/XML format). You can modified each of the options presented in this file. Currently, the file is an INI configuration file. In future we expect to handle json or XML file configuration. As expected, each module of Bilbo can run on his own. A series (not all) of parsing options are available and can be set with `arg cli python` running.

12.1 PIPELINE

In this part, you have to specify the pipeline wanted. Note that for training it does not make sens to add generate pipeline. Pipeline is one on this [components](#) before going any further. This section is marked by:

- [PIPELINE]

12.1.1 verbose

Set at False by default

12.1.2 pipeline

You have to chained the desired chained algorithm as instance:

```
PIPELINE=shape_data, features, svm, crf, generate
```

Example:

```
[PIPELINE]
PIPELINE=shape_data, features, svm, crf, generate
outputFile=None
verbose=True
```

12.2 SHAPER

This section is marked by:

- [shaper]

12.2.1 tokenizerOption

This option is currently unnecessary. The next developments regarding tokenization will make it active soon.

12.2.2 tagOptions

This is a wrapper for reduce or rename tag to an other

```
tagsOptions = {
    "title_a": "title",
    "distributor": "publisher",
    "country": "place",
    "sponsor": "publisher"
}
```

Example:

```
[PIPELINE]
PIPELINE=shape_data, features, svm, crf, generate
outputFile=None
verbose=True
```

12.3 FEATURES

This section is marked by:

- [features]

12.3.1 listFeatures

Default Value is set to: numbersMixed, cap, dash, biblPosition, initial

You can removed some of them or add a new one (see ../developer/modules.html) This is a wrapper for reduce or rename tag to an other

12.3.2 listFeaturesRegex

You can add a list of regex as : (name_of_regex, python_regex), (name_of_regex1, python_regex1)

12.3.3 listFeaturesExternes

(unic_named_list, path_to_external_list, List_type), ...

Note type_list is simple (simple word list) or multi (multi word list as journals names for instance)

12.3.4 listFeaturesXML

This is set to italic by default.

12.3.5 output

Path output, it is handling when you use feature component. Output is fitted to CRF++ format data.

Example:

```
[features]
listFeatures = numbersMixed, cap, dash, biblPosition, initial
listFeaturesRegex = ('WEBLINK', '^((http:\\\\www\\.|https:\\\\www\\.|http:\\\\|https:\\\\|
↵)?[a-z0-9]+(\\-\\.){1}[a-z0-9]+)*\\. [a-z]{2,5}(:[0-9]{1,5})?(\\/\\.*)?$')
listFeaturesExternes = ('place', 'resources/external/place_list.txt', 'multi'),
                        ('possmoth', 'resources/external/month_list.txt', 'simple'),
                        ('posseditor', 'resources/external/editor_abbr_list.txt', 'simple
↵'),
                        ('posspage', 'resources/external/page_abbr_list.txt', 'simple'),
                        ('journal', 'resources/external/journals_list.txt', 'multi'),
                        ('surname', 'resources/external/surname_list.txt', 'simple'),
                        ('forename', 'resources/external/forename_list.txt', 'simple')
listFeaturesXML = italic
output = bilbo/testFiles/features.output.txt
verbose = False
```

12.4 CRF

This section is marked by:

- [crf]

12.4.1 name

Name of libraries used, in some cases you can change the crf libraries (for wapiti for instance)

12.4.2 algoCrf

Default value is set to [lbfgs](for https://en.wikipedia.org/wiki/Limited-memory_BFGS) algorithm : {'lbfgs', 'l2sgd', 'ap', 'pa', 'arow'}

12.4.3 optionCrf

Many option are available . see [crfsuite manual](#)

Most important are c1 for a L1 regularisation (in this case algoritm is switch to orthant method), c2 regression ridge and and max_iterations

epsilon : The epsilon parameter that determines the condition of convergence. value set by default at 1e-5

```
optionCrf = {  
    'c2': 0.00001,  
}
```

12.4.4 patternsFile

path to wapiti pattern. By default pattern used is located in resources/models/bibl/wapiti_pattern_ref

12.4.5 modelFile

Path to the model generated in train action or used in tag action.

12.4.6 seed

This is used to generate a pseudo-random number. This random number is used when you evaluate the crf algorithm only (not the full pipeline)

Example:

```
[crf]  
name = crfsuite  
algoCrf = lbfgs  
# lbfgs for Gradient descent using the L-BFGS method,  
# l2sgd for Stochastic Gradient Descent with L2 regularization term  
# ap for Averaged Perceptron  
# pa for Passive Aggressive  
# arow for Adaptive Regularization Of Weight Vector  
optionCrf = {  
    'c2': 0.00001,  
    'max_iterations': 2000,  
}  
seed = 3  
patternsFile = resources/models/note/wapiti_pattern_ref  
modelFile = resources/models/note/crf_OE_fr.txt
```

12.5 SVM

This section is marked by:

- [svm]

12.5.1 name

Name of libraries used, in some cases you can change the svm libraries for an other.

12.5.2 modelFile

Path to the vocab model generated in train action or used in tag action.

12.5.3 vocab

Path to the vocab model generated by svm train. Vocab attribute at each word a integer.

12.5.4 output

Not already implemented

Example:

```
bsvm
modelFile = resources/models/note/svm_OE_fr.txt
vocab = resources/models/note/inputID.txt
output = /tmp/data_SVM.txt
```


Knowledge Base are located in `resources/` path at the root at `bilbo2` . There are splitted in three ways:

- Training Corpus *corpus*
- External List *external*
- Model and specific pattern *model*

13.1 Corpus

They are used to train bilbo automatic annotation. This is annotated data used in supervised machine learning algorithms. XML / TEI corpus are available in 4 languages (pt, fr, de, en) for bibliographies references. Only a mixed corpus of french and english is available for footnote.

13.2 Externals list

List can be simple or with multiwords, you must specify type of list in `options`:

- Authors (fullname, surname, forename).
- Abbreviation (month, page, editor).
- Journals
- Place

13.3 Models

Models are splitted in two ways (in `bibl` and `note` directory). There are contains feature templates pattern (CRF++ format), see [documentation](#). Note that we are used `crf++` templates with `crf-suite` for convenience. A script is handling conversion between the both input format.

In note we are used crf and svm model. To svm model see installation and data format [documentation](#) in libsvm README.

It is not easy in a pipelines to evaluate the relevance of each algorithm and the relevance of a series of algorithms. As bilbo is constructed as a toolkit for researcher, we can evaluate each pipeline and doing a end to end evaluation. In some cases, a component library can handle its own evaluation. In all cases we rebuilt a confusion matrix.

14.1 End to End Evaluation

For a end to end evaluation, split your dataset in train and test then you need to train first:

```
python3 -m bilbo.bilbo --action train -c MY_PIPELINE.cfg -i DATA_TRAIN.xml -t tag
```

Then, evaluate:

```
python3 -m bilbo.bilbo --action evaluate -c MY_PIPELINE.cfg -i DATA_TEST.xml -t tag
```

14.2 Evaluation by component

For evaluate one component, you need to create the input standart format of your library (the features matrice fitted to your library):

```
python3 -m bilbo.bilbo --action train -c MY_PIPELINE.cfg -i DATA_SET.xml -t tag
```

Check in MY_PIPELINE.cfg the path to your input standart standart format of your component.

Then evaluate:

```
python3 -m bilbo.components.MY_COMPONENT -cf MY_PIPELINE.cfg -i INPUT_STANDART_  
↪COMPONENT --evaluate
```

14.3 K-Fold Croos Validation

For easier use, a bash and python scripts are available for doing K-Fold Cross Validation.

Utilisation : `bash tools/eval.sh DIR_TO_CORPUS/ DIR_TO_OUTPUT/ K_FOLD_INTEGER PATH_CONFIG_FILE` Keep in mind to give directories with the slash `'/'`.

OUTPUT are `std_out`, train and test file splitting and a summary file (`evaluation.csv`) of the different evaluations.

```
cd bilbo2
bash tools/eval.sh ../bibl/ ../test/ 5 bilbo/config/pipeline_bibl.cfg
```

14.4 Examples

For resultat and evaluation of bilbo: [Resultat](#)

15.1 Bibliography tag

15.1.1 Crf component

For evaluate Conditional Random Field on the first step of the bibliography pipeline, you have to train crf component and moreover write a crf input format

```
python3 -m bilbo.bilbo --action train -i resources/corpus/bibl/oe_bibl_en_fr.xml -c ↵  
↵bilbo/config/pipeline_bibl1.cfg
```

(Ajouter un write features)

For evaluate, just get your input format and launch the module with evaluate parameter. It is possible because python-crf module offers an option with a random seed for split dataset in two dataset (train and test)

```
python3 -m bilbo.components.crf -cf bilbo/tests/pipeline.cfg -i bilbo/testFiles/  
↵features.output.txt --evaluate -vvvvv
```

15.1.2 End to End evaluation

In this case you need to split by yourself your dataset in two ways (train.xml and test.xml). Below we randomly assign data in two sets (one data train and one dataset), A simple holdout method for validation (80 % data train, 20 % datatest)

```
python3 -m bilbo.bilbo --action train -i resources/corpus/bibl/train.xml -c bilbo/  
↵config/pipeline_bibl1.cfg -vvvvv
```

```
python3 -m bilbo.bilbo --action evaluate -i resources/corpus/bibl/test.xml -c bilbo/  
↵config/pipeline_bibl1.cfg -vvvvv
```

15.2 Note tag

In this case, we have to evaluate the classifier algorithm (SVM) (dedicated to get note which contains bibliography) and the CRF components used to annotated bibliographies.

15.2.1 Crf component evaluation

15.2.2 Svm component evaluation

```
python3 -m bilbo.components.svm --evaluate -c bilbo/config/pipeline_notel.cfg -i_
↳resources/models/note/data_SVM.txt
```

Accuracy = 93.3993% (283/303) (classification) (93.3993399339934, 0.264026402640264, 0.6858941220502061)

15.2.3 End to End evaluation

In this case you need to split by yourself your dataset in two ways (train.xml and test.xml). Below we randomly assign data in two sets (one data train and one dataset), A simple holdout method for validation (70 % data train, 30 % data test)

Pour les notes:

```
python3 -m bilbo.bilbo --action train -c bilbo/config/pipeline_notel.cfg -i resources/
↳corpus/note/train.xml -t note -vvvv
python3 -m bilbo.bilbo --action evaluate -c bilbo/config/pipeline_notel.cfg -i_
↳resources/corpus/note/test.xml -t note -vvvv
```

16.1 bilbo package

16.1.1 Subpackages

bilbo.components package

Subpackages

bilbo.components.crf package

Submodules

bilbo.components.crf.crf module

Module contents

CRF module (train / tag / evaluate)

bilbo.components.features package

Submodules

bilbo.components.features.decorator_feature module

decorator class

```
class bilbo.components.features.decorator_feature.PositionDecorator (extractor)  
    Bases: object
```

PositionDecorator Class

```
class bilbo.components.features.decorator_feature.SectionDecorator (extractor)  
    Bases: object
```

SectionDecorator Class

```
class bilbo.components.features.decorator_feature.WordDecorator (extractor)  
    Bases: object
```

WordDecorator class

```
class bilbo.components.features.decorator_feature.XmlDecorator (extractor)  
    Bases: object
```

SectionDecorator Class

bilbo.components.features.externalfeatures module

External feature Class

```
class bilbo.components.features.externalfeatures.DictionaryFeature (name,  
                                                                file-  
                                                                name)
```

Bases: *bilbo.components.features.externalfeatures.ExternalFeature*

Get features from dictionnaires

```
create_list (sequence)
```

Create a liste of token from a sequence

Parameters **sequence** – list of token and label associated i.e: [[“token”, “label”], [“token”, “label”]]

```
class bilbo.components.features.externalfeatures.ExternalFeature
```

Bases: object

ExternalFeature CClass generate the feature from external ressources

```
classmethod factory (typeft, name, list_name)
```

Chose between single or multiple tokens feature

Parameters

- **typeft** – simple or multiple
- **name** – the name of the feature
- **list_name** – list file

Returns the right function to call

```
class bilbo.components.features.externalfeatures.ListFeature (name, list_name)
```

Bases: *bilbo.components.features.externalfeatures.ExternalFeature*

ListFeature Class

bilbo.components.features.features module

Features

```

class bilbo.components.features.features.FeatureHandler(cfg_file,
                                                       type_config='ini')
    Bases: bilbo.components.component.Extractor
    Feature handler
    format_to_list (doc)
    loadFonctionsFeatures ()
        Load function for the features
    print_features (doc)
    save_features (doc)
        Write the features for each token in the output file specify in the cli
        Parameters doc – document object

```

bilbo.components.features.localfeatures module

Local features

```

class bilbo.components.features.localfeatures.LocalFeature
    Bases: object
    Local features class
    biblPosition = <bilbo.components.features.decorator_feature.PositionDecorator object>
    cap = <bilbo.components.features.decorator_feature.WordDecorator object>
    dash = <bilbo.components.features.decorator_feature.WordDecorator object>
    initial = <bilbo.components.features.decorator_feature.WordDecorator object>
    numbersMixed = <bilbo.components.features.decorator_feature.WordDecorator object>

```

bilbo.components.features.regexfeatures module

regular expression features

```

class bilbo.components.features.regexfeatures.RegexFeature(name, pattern)
    Bases: object
    generate features based on regular expressions

```

bilbo.components.features.xmlfeatures module

XML features

```

class bilbo.components.features.xmlfeatures.XmlFeature
    Bases: object
    Generate feature based on XML datas
    static format_xpath (element, xpath)
    global_boolean = <bilbo.components.features.decorator_feature.SectionDecorator object>
    italic = <bilbo.components.features.decorator_feature.XmlDecorator object>
    punc_counter = <bilbo.components.features.decorator_feature.SectionDecorator object>

```

Module contents

Features generation module

bilbo.components.shape_data package

Submodules

bilbo.components.shape_data.shape_data module

Module contents

Turns document into a data structure manipulable

bilbo.components.svm package

Submodules

bilbo.components.svm.svm module

SVM

class `bilbo.components.svm.svm.Svm(cfg_file, type_config='ini')`

Bases: `bilbo.components.component.Estimator`

SVM class

evaluate (*document*)

Evaluate the model for the given data. All the data are split into 80/20% for the training / testing process

Parameters **document** – document object

extract_xy (*data*)

fit (*document*, *keep_on_doc=False*)

generate_vocab_dict (*document*)

get_svm_data (*document*, *keep_on_doc*)

shape the svm features data for the svm :param document: document object of the document :returns: svm shaped data

predict (*document*)

tag the new data basde on a given model

Parameters **document** – document object

Returns list of predictions

train (*document*)

Train the SVM model

Parameters **document** – document object

transform (*document*, *mode*)

word_count (*section*)

words_iterator (*section*)

write_data_svm (*data*)

Module contents

SVM module (train / tag)

Submodules

bilbo.components.component module

Component

class bilbo.components.component.**Component** (*cfg_file, type_config*)

Bases: object

Component abstract Class

fit (*document*)

classmethod **get_module_name** ()

classmethod **get_parser_name** ()

transform (*document*)

class bilbo.components.component.**Estimator** (*cfg_file, type_config*)

Bases: *bilbo.components.component.Component*

Estimator Extract Class

evaluate ()

predict (*document*)

train (*document*)

transform (*document, mode*)

class bilbo.components.component.**Extractor** (*cfg_file, type_config*)

Bases: *bilbo.components.component.Component*

Extractor Extract Class

extract_from_section (**args*)

fit (*document*)

transform (*document*)

Module contents

bilbo.components.**load_components** ()

bilbo.libs package

Submodules

bilbo.libs.opts module

@brief Gestion des options

class bilbo.libs.opts.**BilboParser**

Bases: object

classmethod **factory** (*type_args, section_args*)

classmethod **getArgs** (*args, opt, type_opt, pipe*)

class bilbo.libs.opts.**DictParser**

Bases: *bilbo.libs.opts.BilboParser*

classmethod **getArgs** (*args, opt, type_opt=None, pipe=None*)

class bilbo.libs.opts.**IniParser**

Bases: *bilbo.libs.opts.BilboParser*

classmethod **getArgs** (*cfg_file, opt, type_opt=None, pipe=None*)

Gets arguments from config file

Parameters

- **cfgFile** – config file
- **opt** – specify the option in the file

class bilbo.libs.opts.**Parser**

Bases: object

classmethod **getArgs** ()

classmethod **get_parser** (*name, help=None*)

classmethod **parse_arguments** ()

Module contents

bilbo.storage package

Submodules

bilbo.storage.document module

Document

class bilbo.storage.document.**Document** (*str_value, xml_tree, tag, sections=None*)

Bases: object

Class that describe a document.

Str_value string value with tags

Xml_tree lxml object

Section cf section

```

genereDocumentPivot ()
    print the document stored

```

bilbo.storage.section module

section

```

class bilbo.storage.section.Section (str_value, section_naked, section_xml, tokens=None, token_str_lst=None, bibl_status=True, keys=None)

```

Bases: object

describe the section stored

Str_value string value with tags

Section_naked string value without tag

Section_xml lxml object

Tokens list of token object (cf token)

Token_str_lst list of string token

Bibl_status True if the section contain a bibl tag Faslse otherwise

check_constraint (*constraint*)

```

print_tokens ()
    print section

```

bilbo.storage.token module

Token

```

class bilbo.storage.token.Token (str_value, label, features=None, predict_label=None, separator=True)

```

Bases: object

Describe the token object

Str_value string value

Label label associated

Feature list of feature

```

printToken ()
    Print token

```

tail

word

bilbo.storage.trie module

Trie

```

class bilbo.storage.trie.Trie (filename=None)

```

Bases: object

The Trie object.

```
add (sequence)  
    add element  
  
data
```

Module contents

init storage

bilbo.tests package

Submodules

bilbo.tests.test_feature module

```
class bilbo.tests.test_feature.TestDictionnayFeature (methodName='runTest')  
    Bases: unittest.case.TestCase
```

```
    setUp ()  
        Hook method for setting up the test fixture before exercising it.
```

```
    test_dict ()
```

```
class bilbo.tests.test_feature.TestListFeature (methodName='runTest')  
    Bases: unittest.case.TestCase
```

```
    setUp ()  
        Hook method for setting up the test fixture before exercising it.
```

```
    test_dict ()
```

```
class bilbo.tests.test_feature.TestLocalFeature (methodName='runTest')  
    Bases: unittest.case.TestCase
```

```
    test_biblposition ()
```

```
    test_cap ()
```

```
    test_dash ()
```

```
    test_initial ()
```

```
    test_numbersMixed ()
```

```
class bilbo.tests.test_feature.TestRegexFeature (methodName='runTest')  
    Bases: unittest.case.TestCase
```

```
    setUp ()  
        Hook method for setting up the test fixture before exercising it.
```

```
    test_match_regex ()
```

```
    test_nomatch_regex ()
```

```
bilbo.tests.test_feature.load_list_predict (section, function)
```

```
bilbo.tests.test_feature.load_section (data)
```

bilbo.tests.test_importer module

bilbo.tests.test_shapesection module

bilbo.tests.tests module

Module contents

bilbo.tokenizers package

Submodules

bilbo.tokenizers.en module

```
class bilbo.tokenizers.en.EnglishTokenizer
    Bases: bilbo.tokenizers.tokenizers.DefaultTokenizer

    tokenize (option)
```

bilbo.tokenizers.fr module

```
class bilbo.tokenizers.fr.FrenchTokenizer
    Bases: bilbo.tokenizers.tokenizers.DefaultTokenizer

    tokenize (text)
        Tokenize the sentence given in parameter and return a list of tokens. This is a two-steps process: 1.
        tokenize text using punctuation marks, 2. merge over-tokenized units using the lexicon or a regex (for
        compounds, ‘^[A-Z][a-z]+-[A-Z][a-z]+$’).
```

bilbo.tokenizers.tokenizers module

tokenizer module

```
class bilbo.tokenizers.tokenizers.DefaultTokenizer
    Bases: object

    lexicon = None
        The dictionary containing the lexicon.

    loadlist (path)
        Load a resource list and generate the corresponding regexp part.

    resources = None
        The path of the resources folder.

    tokenize (text)

class bilbo.tokenizers.tokenizers.Tokenizer
    Bases: object

    Tokenizer class tokenize a given string
```

Module contents

Tokenizers modules

bilbo.utils package

Submodules

bilbo.utils.crf_datas module

crf data

`bilbo.utils.crf_datas.apply_patterns` (*sections_xseq, patterns, empty_features=False*)

brief Transform a list of features given patterns

Parameters `sections_xseq` – iterable : a generator on a list of sections features list and labels

Returns a generator that yields a list new list of features given patterns

`bilbo.utils.crf_datas.extract_y` (*sections, nfeatures=None*)

Parameters

- **sections** – iterable : a sections generator (like returned by `fd2sections()`)
- **nfeatures** – Nonelint : if None the first line of the first section is expected to be with a label for last feature. Else `nfeatures` indicate the number of features, `sections[x][nfeatures]` is the line's label.

Returns a generator that yields one tuple(`xseq`, `yseq`) per section

`bilbo.utils.crf_datas.fd2patterns` (*patterns_fd*)

brief Read a Wapiti pattern file

Parameters `patterns_fd` – iterable : a line generator

Returns An array of tuple(`name`, `row`, `col`)

`bilbo.utils.crf_datas.fd2sections` (*datas_fd, sep=None*)

brief Generator that yield sections of features from a BIOS formatted content coming from a line generator

Parameters

- **datas_fd** – iterable: a line generator (as returned by `open()`)
- **sep** – Nonelstr : if None yield single string containing BIOS formatted features. Else splits lines and features given `sep`

Returns Depends on bios

`bilbo.utils.crf_datas.sections2evaluate` (*sections, prop=0.8, seed=None*)

brief Split sections into a training and an evaluation part

Parameters

- **sections** – iterable: items are sections
- **prop** – float : div proportions
- **seed** – int | None: random seed

Returns split section fro train / test purposes

`bilbo.utils.crf_datas.trainer_opts` (*name, options*)
 brief Return a dict of options for the trainer

Parameters

- **name** – str : can be wapiti | crfsuite
- **options** – str (dict) with the option of crfsuite

Returns a dict

bilbo.utils.dictionaries module

dictionaries

`bilbo.utils.dictionaries.compile_multiword` (*infile*)

Parameters *infile* – str

`bilbo.utils.dictionaries.generatePickle` (*dic, infile*)
 Generate de pickle file

Parameters

- **dic** – dictionnaire
- **infile** – str

Returns pickle file

bilbo.utils.svm_datas module

`bilbo.utils.svm_datas.fd2features` (*datas_fd, to_dict=False*)
 Process SVM data file

Parameters *to_dict* – bool : if true yield values are dict, else strings

Returns a generator

`bilbo.utils.svm_datas.fd2labeled_evaluation` (*datas_fd, to_dict=False, prop=0.8, seed=None*)

brief Return 2 iterator on training and on evaluation datas (same generator than `fd2labeled_features`

Parameters *to_dict* – bool : if true return a dict else a string

Returns tuple(train_datas, validation_datas)

`bilbo.utils.svm_datas.fd2labeled_features` (*datas_fd, to_dict=False*)

Generator comparable to `fd2features` but that yield a tuple with (label, features)

Parameters *to_dict* – bool: if true the features are returned as a dict else a string is yield

Returns a generator that yield tuples

`bilbo.utils.svm_datas.svmRepport` (*y_test, y_pred*)
 Print the evaluation repport given the test and prediction data

Parameters

- **y_test** – list of test label (oracle)

- **y_pred** – list of predicted label (same range as test)

`bilbo.utils.svm_datas.svm_opts()`

Return kwargs and args for model training given argparse parsed arguments

Parameters `args` – Namespace: as returned by `ArgumentParser.parse_argument()`

Returns a tuple(args, kwargs)

bilbo.utils.timer module

Timer class

class `bilbo.utils.timer.Timer` (*name=""*, *autostart=True*)

Bases: object

Simple timer class

last

mean ()

Returns the average of recorded timers

name

reset (*name=None*)

Reset the timer and store elapsed time

Parameters `name` – str: new timer name. If given stored data are erased

start ()

Starts the timer

t ()

Returns elapsed seconds since last start() call

Module contents

utils init

16.1.2 Submodules

16.1.3 bilbo.bilbo module

16.1.4 bilbo.eval module

class `bilbo.eval.Evaluation` (*gold*, *predicted*, *option='fine'*)

Bases: object

Evaluation class

evaluate ()

Compute all the precisions, recalls, f-measures and count for the confusion matrix :return: dict(label, precision), dict(label, recall), dict(label, f_measures), dict(label, count), dict(macro)

get_col_sum (*label*)

return the sum of a given column

get_confusion_matrix ()
 Generate the confusion matrix populate matrix with the confusion matrix populate imap

get_count_for_label (*label*)
 param label : a given label return the number of occurrences for a given label

get_count_for_labels ()
 return a dict with the number of occurrences for each label :return: dict (label, count)

get_f_measure_for_labels (*beta: float = 1*)
 Returns F1 score for all labels. See http://en.wikipedia.org/wiki/F1_score

Parameters **beta** – the beta parameter higher than 1 prefers recall, lower than 1 prefers precision

Returns dict (label, F1)

get_macro_f_measure ()
Returns the mean f-measure for the whole document

get_macro_f_measure_weighted ()
Returns the weighted mean f-measure for the whole document

get_macro_precision ()

get_macro_precision_weighted ()

get_macro_recall ()

get_macro_recall_weighted ()

get_precision_for_label (*label*)
 param label : a given label return the precision for a given label

get_precision_for_labels ()
 return a dict with the precision of each label :return: dict (label, precision)

get_recall_for_label (*label*)
 param label : a given label return the recall for a given label

get_recall_for_labels ()
 return a dict with the recall of each label :return: dict (label, recall)

get_row_sum (*label*)
 return the sum of a given row

get_true_positive (*label*)
 return the true positive from the matrix

get_unique_label ()
 return a list of unique label from the gold and predicted lists

print_csv (*precisions, recalls, f_measures, counts, macro, csvfile*)

print_std (*precisions, recalls, f_measures, counts, macro*)

16.1.5 bilbo.generateXml module

16.1.6 bilbo.importer module

16.1.7 Module contents

CHAPTER 17

Indices and tables

- `genindex`
- `modindex`
- `search`

b

`bilbo`, 53

`bilbo.components`, 45

`bilbo.components.component`, 45

`bilbo.components.crf`, 41

`bilbo.components.features`, 44

`bilbo.components.features.decorator_feature`, 41

`bilbo.components.features.externalfeatures`, 42

`bilbo.components.features.features`, 42

`bilbo.components.features.localfeatures`, 43

`bilbo.components.features.regexfeatures`, 43

`bilbo.components.features.xmlfeatures`, 43

`bilbo.components.shape_data`, 44

`bilbo.components.svm`, 45

`bilbo.components.svm.svm`, 44

`bilbo.eval`, 52

`bilbo.libs`, 46

`bilbo.libs.opts`, 46

`bilbo.storage`, 48

`bilbo.storage.document`, 46

`bilbo.storage.section`, 47

`bilbo.storage.token`, 47

`bilbo.storage.trie`, 47

`bilbo.tests`, 49

`bilbo.tests.test_feature`, 48

`bilbo.tests.tests`, 49

`bilbo.tokenizers`, 50

`bilbo.tokenizers.en`, 49

`bilbo.tokenizers.fr`, 49

`bilbo.tokenizers.tokenizers`, 49

`bilbo.utils`, 52

`bilbo.utils.crf_datas`, 50

`bilbo.utils.dictionaries`, 51

`bilbo.utils.svm_datas`, 51

`bilbo.utils.timer`, 52

A

add() (*bilbo.storage.trie.Trie* method), 47
 apply_patterns() (*in module bilbo.utils.crf_datas*),
 50

B

biBlPosition (*bilbo.components.features.localfeatures.LocalFeature*
attribute), 43
 bilbo (*module*), 53
 bilbo.components (*module*), 45
 bilbo.components.component (*module*), 45
 bilbo.components.crf (*module*), 41
 bilbo.components.features (*module*), 44
 bilbo.components.features.decorator_feature
 (*module*), 41
 bilbo.components.features.externalfeatures
 (*module*), 42
 bilbo.components.features.features (*mod-
 ule*), 42
 bilbo.components.features.localfeatures
 (*module*), 43
 bilbo.components.features.regexfeatures
 (*module*), 43
 bilbo.components.features.xmlfeatures
 (*module*), 43
 bilbo.components.shape_data (*module*), 44
 bilbo.components.svm (*module*), 45
 bilbo.components.svm.svm (*module*), 44
 bilbo.eval (*module*), 52
 bilbo.libs (*module*), 46
 bilbo.libs.opts (*module*), 46
 bilbo.storage (*module*), 48
 bilbo.storage.document (*module*), 46
 bilbo.storage.section (*module*), 47
 bilbo.storage.token (*module*), 47
 bilbo.storage.trie (*module*), 47
 bilbo.tests (*module*), 49
 bilbo.tests.test_feature (*module*), 48
 bilbo.tests.tests (*module*), 49

bilbo.tokenizers (*module*), 50
 bilbo.tokenizers.en (*module*), 49
 bilbo.tokenizers.fr (*module*), 49
 bilbo.tokenizers.tokenizers (*module*), 49
 bilbo.utils (*module*), 52
 bilbo.utils.crf_datas (*module*), 50
 bilbo.utils.dictionaries (*module*), 51
 bilbo.utils.svm_datas (*module*), 51
 bilbo.utils.timer (*module*), 52
 BilboParser (*class in bilbo.libs.opts*), 46

C

cap (*bilbo.components.features.localfeatures.LocalFeature*
attribute), 43
 check_constraint() (*bilbo.storage.section.Section*
method), 47
 compile_multiword() (*in module
 bilbo.utils.dictionaries*), 51
 Component (*class in bilbo.components.component*), 45
 create_list() (*bilbo.components.features.externalfeatures.Dictionary*
method), 42

D

dash (*bilbo.components.features.localfeatures.LocalFeature*
attribute), 43
 data (*bilbo.storage.trie.Trie* attribute), 48
 DefaultTokenizer (*class in
 bilbo.tokenizers.tokenizers*), 49
 DictionaryFeature (*class in
 bilbo.components.features.externalfeatures*),
 42
 DictParser (*class in bilbo.libs.opts*), 46
 Document (*class in bilbo.storage.document*), 46

E

EnglishTokenizer (*class in bilbo.tokenizers.en*), 49
 Estimator (*class in bilbo.components.component*), 45
 evaluate() (*bilbo.components.component.Estimator*
method), 45

evaluate() (*bilbo.components.svm.svm.Svm method*), 44
 evaluate() (*bilbo.eval.Evaluation method*), 52
 Evaluation (*class in bilbo.eval*), 52
 ExternalFeature (*class in bilbo.components.features.externalfeatures*), 42
 extract_from_section() (*bilbo.components.component.Extractor method*), 45
 extract_xy() (*bilbo.components.svm.svm.Svm method*), 44
 extract_y() (*in module bilbo.utils.crf_datas*), 50
 Extractor (*class in bilbo.components.component*), 45

F

factory() (*bilbo.components.features.externalfeatures.ExternalFeature class method*), 42
 factory() (*bilbo.libs.opts.BilboParser class method*), 46
 fd2features() (*in module bilbo.utils.svm_datas*), 51
 fd2labeled_evaluation() (*in module bilbo.utils.svm_datas*), 51
 fd2labeled_features() (*in module bilbo.utils.svm_datas*), 51
 fd2patterns() (*in module bilbo.utils.crf_datas*), 50
 fd2sections() (*in module bilbo.utils.crf_datas*), 50
 FeatureHandler (*class in bilbo.components.features.features*), 42
 fit() (*bilbo.components.component.Component method*), 45
 fit() (*bilbo.components.component.Extractor method*), 45
 fit() (*bilbo.components.svm.svm.Svm method*), 44
 format_to_list() (*bilbo.components.features.features.FeatureHandler method*), 43
 format_xpath() (*bilbo.components.features.xmlfeatures.XmlFeature static method*), 43
 FrenchTokenizer (*class in bilbo.tokenizers.fr*), 49

G

generate_vocab_dict() (*bilbo.components.svm.svm.Svm method*), 44
 generatePickle() (*in module bilbo.utils.dictionaries*), 51
 genereDocumentPivot() (*bilbo.storage.document.Document method*), 46
 get_col_sum() (*bilbo.eval.Evaluation method*), 52
 get_confusion_matrix() (*bilbo.eval.Evaluation method*), 52
 get_count_for_label() (*bilbo.eval.Evaluation method*), 53
 get_count_for_labels() (*bilbo.eval.Evaluation method*), 53
 get_f_measure_for_labels() (*bilbo.eval.Evaluation method*), 53
 get_macro_f_measure() (*bilbo.eval.Evaluation method*), 53
 get_macro_f_measure_weighted() (*bilbo.eval.Evaluation method*), 53
 get_macro_precision() (*bilbo.eval.Evaluation method*), 53
 get_macro_precision_weighted() (*bilbo.eval.Evaluation method*), 53
 get_macro_recall() (*bilbo.eval.Evaluation method*), 53
 get_macro_recall_weighted() (*bilbo.eval.Evaluation method*), 53
 get_parser_name() (*bilbo.components.component.Component class method*), 45
 get_parser() (*bilbo.libs.opts.Parser class method*), 46
 get_parser_name() (*bilbo.components.component.Component class method*), 45
 get_precision_for_label() (*bilbo.eval.Evaluation method*), 53
 get_precision_for_labels() (*bilbo.eval.Evaluation method*), 53
 get_recall_for_label() (*bilbo.eval.Evaluation method*), 53
 get_recall_for_labels() (*bilbo.eval.Evaluation method*), 53
 get_row_sum() (*bilbo.eval.Evaluation method*), 53
 get_svm_data() (*bilbo.components.svm.svm.Svm method*), 44
 get_true_positive() (*bilbo.eval.Evaluation method*), 53
 get_unique_label() (*bilbo.eval.Evaluation method*), 53
 getArgs() (*bilbo.libs.opts.BilboParser class method*), 46
 getArgs() (*bilbo.libs.opts.DictParser class method*), 46
 getArgs() (*bilbo.libs.opts.IniParser class method*), 46
 getArgs() (*bilbo.libs.opts.Parser class method*), 46
 global_boolean (*bilbo.components.features.xmlfeatures.XmlFeature attribute*), 43

I

IniParser (*class in bilbo.libs.opts*), 46
 initial (*bilbo.components.features.localfeatures.LocalFeature attribute*), 43
 italic (*bilbo.components.features.xmlfeatures.XmlFeature attribute*), 43

L

last (*bilbo.utils.timer.Timer* attribute), 52

lexicon (*bilbo.tokenizers.tokenizers.DefaultTokenizer* attribute), 49

ListFeature (class in *bilbo.components.features.externalfeatures*), 42

load_components() (in module *bilbo.components*), 45

load_list_predict() (in module *bilbo.tests.test_feature*), 48

load_section() (in module *bilbo.tests.test_feature*), 48

loadFonctionsFeatures() (*bilbo.components.features.features.FeatureHandler* method), 43

loadlist() (*bilbo.tokenizers.tokenizers.DefaultTokenizer* method), 49

LocalFeature (class in *bilbo.components.features.localfeatures*), 43

M

mean() (*bilbo.utils.timer.Timer* method), 52

N

name (*bilbo.utils.timer.Timer* attribute), 52

numbersMixed (*bilbo.components.features.localfeatures.LocalFeature* attribute), 43

P

parse_arguments() (*bilbo.libs.opts.Parser* class method), 46

Parser (class in *bilbo.libs.opts*), 46

PositionDecorator (class in *bilbo.components.features.decorator_feature*), 41

predict() (*bilbo.components.component.Estimator* method), 45

predict() (*bilbo.components.svm.svm.Svm* method), 44

print_csv() (*bilbo.eval.Evaluation* method), 53

print_features() (*bilbo.components.features.features.FeatureHandler* method), 43

print_std() (*bilbo.eval.Evaluation* method), 53

print_tokens() (*bilbo.storage.section.Section* method), 47

printToken() (*bilbo.storage.token.Token* method), 47

punc_counter (*bilbo.components.features.xmlfeatures.XmlFeature* attribute), 43

R

RegexFeature (class in

bilbo.components.features.regexfeatures), 43

reset() (*bilbo.utils.timer.Timer* method), 52

resources (*bilbo.tokenizers.tokenizers.DefaultTokenizer* attribute), 49

S

save_features() (*bilbo.components.features.features.FeatureHandler* method), 43

Section (class in *bilbo.storage.section*), 47

SectionDecorator (class in *bilbo.components.features.decorator_feature*), 42

sections2evaluate() (in module *bilbo.utils.crf_datas*), 50

setUp() (*bilbo.tests.test_feature.TestDictionnayFeature* method), 48

setUp() (*bilbo.tests.test_feature.TestListFeature* method), 48

setUp() (*bilbo.tests.test_feature.TestRegexFeature* method), 48

start() (*bilbo.utils.timer.Timer* method), 52

Svm (class in *bilbo.components.svm.svm*), 44

svm_opts() (in module *bilbo.utils.svm_datas*), 52

svmRepport() (in module *bilbo.utils.svm_datas*), 51

T

tail() (*bilbo.utils.timer.Timer* method), 52

tail (*bilbo.storage.token.Token* attribute), 47

test_biblposition() (*bilbo.tests.test_feature.TestLocalFeature* method), 48

test_cap() (*bilbo.tests.test_feature.TestLocalFeature* method), 48

test_dash() (*bilbo.tests.test_feature.TestLocalFeature* method), 48

test_dict() (*bilbo.tests.test_feature.TestDictionnayFeature* method), 48

test_dict() (*bilbo.tests.test_feature.TestListFeature* method), 48

test_initial() (*bilbo.tests.test_feature.TestLocalFeature* method), 48

test_match_regex() (*bilbo.tests.test_feature.TestRegexFeature* method), 48

test_nomatch_regex() (*bilbo.tests.test_feature.TestRegexFeature* method), 48

test_numbersMixed() (*bilbo.tests.test_feature.TestLocalFeature* method), 48

TestDictionnayFeature (class in *bilbo.tests.test_feature*), 48

TestListFeature (class in *bilbo.tests.test_feature*),
48

TestLocalFeature (class in *bilbo.tests.test_feature*),
48

TestRegexFeature (class in *bilbo.tests.test_feature*),
48

Timer (class in *bilbo.utils.timer*), 52

Token (class in *bilbo.storage.token*), 47

tokenize() (*bilbo.tokenizers.en.EnglishTokenizer*
method), 49

tokenize() (*bilbo.tokenizers.fr.FrenchTokenizer*
method), 49

tokenize() (*bilbo.tokenizers.tokenizers.DefaultTokenizer*
method), 49

Tokenizer (class in *bilbo.tokenizers.tokenizers*), 49

train() (*bilbo.components.component.Estimator*
method), 45

train() (*bilbo.components.svm.svm.Svm* method), 44

trainer_opts() (in module *bilbo.utils.crf_datas*), 50

transform() (*bilbo.components.component.Component*
method), 45

transform() (*bilbo.components.component.Estimator*
method), 45

transform() (*bilbo.components.component.Extractor*
method), 45

transform() (*bilbo.components.svm.svm.Svm*
method), 44

Trie (class in *bilbo.storage.trie*), 47

W

word (*bilbo.storage.token.Token* attribute), 47

word_count() (*bilbo.components.svm.svm.Svm*
method), 44

WordDecorator (class in
bilbo.components.features.decorator_feature),
42

words_iterator() (*bilbo.components.svm.svm.Svm*
method), 44

write_data_svm() (*bilbo.components.svm.svm.Svm*
method), 45

X

XmlDecorator (class in
bilbo.components.features.decorator_feature),
42

XmlFeature (class in
bilbo.components.features.xmlfeatures), 43