SIDEKIT Documentation

Release

Anthony LARCHER & Sylvain MEIGNIER & Kong Aik LEE

Mar 16, 2017

CONTENTS

SIDEKIT is an open source package for Speaker and Language recognition.
The aim of **SIDEKIT** is to provide an educational and efficient toolkit for speaker/language recognition including the whole chain of treatment that goes from the audio data to the analysis of the system performance.

Authors Anthony Larcher & Kong Aik Lee & Sylvain Meignier

Version 1.2 of 2017/02/09

See also:

News for **SIDEKIT** 1.2:

- new `sidekit_mpi` module that allows parallel computing on several nodes (cluster) MPI implementations are provided for GMM EM algorithm, TotalVariability matrix EM estimation and i-vector extraction see MPI for more information about MPI
- **new `FactorAnalyser` class that simplifies the interface** Note that FA estimation and i-vector extraction is still available in `StatServer` but deprecated
- i-vector scoring with scaling factor
- uncertainty propagation is available in PLDA scoring

CHAPTER

ONE

WHAT'S HERE?

1.1 An Overview of SIDEKIT

SIDEKIT aims at providing the whole chain of tools required to perform speaker recognition.
The main tools available include:

- Acoustic features extraction
  - Linear-Frequency Cepstral Coefficients (LFCC)
  - Mel-Frequency Cepstral Coefficients (MFCC)
  - RASTA filtering
  - Energy-based Voice Activity Detection (VAD)
  - normalization (CMS, CMVN, Short Term Gaussianization)
- Modeling and classification
  - Gaussian Mixture Models (GMM)
  - *i* - vectors
  - Probabilistic Linear Discriminant Analysis (PLDA)
  - Joint Factor Analysis (JFA)
  - Support Vector Machine (SVM)
  - Deep Neural Network (bridge to THEANO)
- **Presentation of the results**
  - DET plot
  - ROC Convex Hull based DET plot

1.1.1 Implementation

SIDEKIT has been designed and written in Python and released under LGPL *License*
to allow a wider usage of the code that, we hope, could be beneficial to the community.
The structure of the core package makes use of a limited number of classes in order
to facilitate the readability and reusability of the code.
Starting from version 1.1.0 SIDEKIT is no longer tested under Python 2.* In case you want to keep using Python2, you may have modification to do on your own.

SIDEKIT has been tested under Python >3.5 for both Linux and MacOS.

1.1.2 About SIDEKIT

Authors Anthony Larcher & Kong Aik Lee & Sylvain Meignier

Version 1.2 of 2017/02/09

To know about the version and license of **SIDEKIT**

```
sidekit.__version__
sidekit.__license__
```

License

SIDEKIT is released under LGPL license which is an extension of the GPL license.
See below the terms of the license.

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights. Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received. You must make sure that they, too, receive
or can get the source code. And you must show them these terms so they
know their rights.

Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software. For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so. This is fundamentally incompatible with the aim of
protecting users' freedom to change the software. The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable. Therefore, we
have designed this version of the GPL to prohibit the practice for those
products. If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary. To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and
modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form. A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities. However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work. For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

The Corresponding Source for a work in source code form is that
same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met. This License explicitly affirms your unlimited
permission to run the unmodified Program. The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work. This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force. You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do

not control copyright. Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under
the conditions stated below. Sublicensing is not allowed; section 10
makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified
it, and giving a relevant date.

b) The work must carry prominent notices stating that it is
released under this License and any conditions added under section
7. This requirement modifies the requirement in section 4 to
“keep intact all notices”.

c) You must license the entire work, as a whole, under this
License to anyone who comes into possession of a copy. This
License will therefore apply, along with any applicable section 7
additional terms, to the whole of the work, and all its parts,
regardless of how they are packaged. This License gives no
permission to license the work in any other way, but it does not
invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display
Appropriate Legal Notices; however, if the Program has interactive
interfaces that do not display Appropriate Legal Notices, your
work need not make them do so.

A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
“aggregate” if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation’s users
beyond what the individual works permit. Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

a) Convey the object code in, or embodied in, a physical product
(including a physical distribution medium), accompanied by the
Corresponding Source fixed on a durable physical medium
customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product
(including a physical distribution medium), accompanied by a
written offer, valid for at least three years and valid for as
long as you offer spare parts or customer support for that product
model, to give anyone who possesses the object code either (1) a
copy of the Corresponding Source for all the software in the
product that is covered by this License, on a durable physical

medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods,

procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source. The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information. But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed. Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law. If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it. (Additional permissions may be written to require their own
removal in certain cases when you modify the work.) You may place

additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the
> terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or
> author attributions in that material or in the Appropriate Legal
> Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or
> requiring that modified versions of such material be marked in
> reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or
> authors of the material; or
>
> e) Declining to grant rights under trademark law for use of some
> trade names, trademarks, or service marks; or
>
> f) Requiring indemnification of licensors and authors of that
> material by anyone who conveys the material (or modified versions of
> it) with contractual assumptions of liability to the recipient, for
> any liability that these contractual assumptions directly impose on
> those licensors and authors.

All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10. If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term. If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;

the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients. "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License. You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this

License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT

HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License. Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007|

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public

License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

> a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs

whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of
this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from
a header file that is part of the Library. You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the
Library is used in it and that the Library and its use are
covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license
document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

a) Give prominent notice with each copy of the Combined Work that
the Library is used in it and that the Library and its use are
covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license
document.

c) For a Combined Work that displays copyright notices during
execution, include the copyright notice for the Library among
these notices, as well as a reference directing the user to the
copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this
License, and the Corresponding Application Code in a form
suitable for, and under terms that permit, the user to
recombine or relink the Application with a modified version of

the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the

Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Compatibilities

The implementation of **SIDEKIT** benefits from the experience of existing tools and toolkits in the community. The main ones are ALIZE, BOSARIS, HTK and LIBSVM
As far as possible, **SIDEKIT** as been made compatible with those tools by providing read and write functions in the appropriate formats and using similar structures.

ALIZE

SIDEKIT is able to read and write in ALIZE binary format

- a Gaussian Mixture Model
- a label file
- a matrix of statistics computed by using `TotalVariability.exe` or `ComputeJFAStats.exe`.

BOSARIS

A part of the **BOSARIS** toolkit has been translated into Python in order to manipulate

- enrollment lists as *IdMap* objects
- trial lists as *Ndx* objects
- score matrices as *Scores* objects
- trial keys as *Key* objects

to plot Detection Error Trade-off (DET) curves
and compute minimum costs as defined by the NIST .

HTK

SIDEKIT is able to read and write in HTK format

- a feature file (non-compressed)
- a Gaussian Mixture Model (stored as a 3 states HMM)

LIBSVM

SIDEKIT makes use of the LIBSVM library *[Chang11]* and its Python wrapper.
High level interface are provided to train and test using SVMs.

1.2 How to download or install SIDEKIT

All you need to get **SIDEKIT** on your machine.
It is possible to get the sources to manually include in your PYTHONPATH or you can install
via **pip** or **conda**.

1.2.1 Sources

From the GIT (best option for the latest sources)

For the current stable version, use GIT

```
git clone  https://pelennor.univ-lemans.fr/Larcher/sidekit
```

Get the latest version by switching to the `dev` branch of the repository

```
git clone  https://pelennor.univ-lemans.fr/Larcher/sidekit
cd sidekit
git checkout -b dev origin/dev
```

Download sources

You can download here the latest stable version form this page.

1.2.2 Using Conda (recommended)

After installing miniconda:

```
conda install -c anthol sidekit=1.2
```

1.2.3 Using PIP

```
pip install sidekit
```

1.2.4 In a Virtual environment

First, be sure to have *virtualenv* installed.
You can find some documentation on the official website.

Create your virtual environment

```
virtualenv env
```

This will create a directory called `env` in the current directory.
If you want to specify a different python interpreter (for example to test you program with python 3),
you just have to use the *-p* option:

```
virtualenv -p /path/to/python3 env
```

Activate your environment

Each and every time you will want to work on your project, you will have to first activate your virtualenv:

```
. ./env/bin/activate
```

Your prompt should change and you should see the name of your virtualenv between `()`. In our case `(env)`.

1.2.5 Dependencies

SIDEKIT requires the installation of the following tools.

- Python
  SIDEKIT has been developed under Python 3.3, 3.4 and 3.5
  - LINUX: python is natively available on most of LINUX distributions
  - OSX: natively available, you can install a different version of python via Homebrew
  - Windows: Python can be installed on Windows through PythonXY, WinPython or anaconda packages
- **To install other required Python packages use one of the following:**
  - conda
  - pip

The following packages are required to use **SIDEKIT**.

- matplotlib

- mock==1.0.1
- nose==1.3.4
- numpy
- pyparsing==2.0.2
- python-dateutil==2.2
- scipy
- six==1.8.0
- wsgiref==0.1.2
- h5py==2.3.1
- pandas
- theano

Optional linkage

Those packages might be used by **SIDEKIT** if installed. To do so, just make sure they are installed on your machine. When importing, **SIDEKIT** will look for them and link if possible.

- LibSVM: library dedicated to SVM classifiers. This library can be downloaded from
  the official website and easily compiled on all plat-forms
  Compile the library (`libsvm.so.2` on UNIX/Linux and Mac platforms and `libsvm.dll` on windows)
  and create a link or copy this library in `./sidekit/libsvm/`.

1.3 Customize your sidekit via environment variables

SIDEKIT might use LIBSVM, Theno or MPI depending on how and what you intend to do. Here is how to enable/disable the import of the dedicated module to allow running **SIDEKIT** without thoes libraries.

1.3.1 The SIDEKIT environment variable

When importing **SIDEKIT**, it will check the value of an enironment variable called `SIDEKIT`. By default, this variable is set to:

SIDEKIT=”theano=true,theano_config=gpu,libsvm=true,mpi=false”

Thus:

- Theano will be imported and set to use GPU card if available
- LIBSVM will be imported to if the proper library has been compiled for this specific machine
- MPI parallel computation will be disable

1.3.2 How to cut the dependency to Theano?

Set your environment variable to `SIDEKIT="theano=false"` Thus when importing, **SIDEKIT** will not import Theano.

1.3.3 How to switch Theano to CPU mode?

Set your environment variable to `SIDEKIT="theano=true,theano_config=cpu"`

1.3.4 How to disable LIBSVM?

Set your environment variable to `SIDEKIT="libsvm=false"`

1.3.5 How to enable MPI parallel computing?

Set your environment variable to `SIDEKIT="mpi=true"`

1.4 API description

Authors Anthony LACHER, Sylvain MEIGNIER & Kong Aik LEE

Version 1.1.6 of 2016/10/31

This package is the core of the **SIDEKIT** toolkit.
While developing **SIDEKIT**, we tried to keep in mind two
targets:

1. limit the number of classes to allow better readability
2. make **SIDEKIT** compatible with existing tools

To reach this target, we have created four Main Classes
(`FeaturesExtractor`, `FeaturesServer`, `Mixture` and `StatServer`) which can be used
together with a number of tools available in companion
modules (*sidekit_io* and *sv_utils*).

Front-end and back-end processing such as acoustic feature extraction
and score analysis are handled in two packages: *frontend* and *bosaris*.
The *frontend* package include a number of tools to extract and normalize
the acoustic features as well as detecting the high energy frames for voice
activity detection. The *bosaris* package consists of the translation of
a part of the BOSARIS toolkit available on this webpage.
The current python implementation of the BOSARIS toolkit does not include tools
for calibration and fusion but only the core structures that are used to manage
enrollment lists, trial definitions and scores.
The authors would like to thank Niko Brummer and AGNITIO to allow them to distribute
this version of the BOSARIS toolkit.

Note: The bosaris package which is released together with **SIDEKIT** is distributed under a different license. The intellectual property belongs to the original authors of the toolkit.

1.4.1 Main Classes

SIDEKIT is based on three main classes that are described below.

FactorAnalyser

class factor_analyser.**FactorAnalyser**(*input_file_name=None*, *mean=None*, *F=None*, *G=None*, *H=None*, *Sigma=None*)

A class to train factor analyser such as total variability models and Probabilistic Linear Discriminant Analysis (PLDA).

Attr mean mean vector

Attr F between class matrix

Attr G within class matrix

Attr H MAP covariance matrix (for Joint Factor Analysis only)

Attr Sigma residual covariance matrix

extract_ivectors(*ubm*, *stat_server_filename*, *prefix=''*, *batch_size=300*, *uncertainty=False*, *num_thread=1*)

Parallel extraction of i-vectors using multiprocessing module

Parameters

- **ubm** – Mixture object (the UBM)
- **stat_server_filename** – name of the file from which the input StatServer is read
- **prefix** – prefix used to store the StatServer in its file
- **batch_size** – number of sessions to process in a batch
- **uncertainty** – a boolean, if True, return the diagonal of the uncertainty matrices
- **num_thread** – number of process to run in parallel

Returns a StatServer with i-vectors in the stat1 attribute and a matrix of uncertainty matrices (optional)

extract_ivectors_single(*ubm*, *stat_server*, *uncertainty=False*)

Estimate i-vectors for a given StatServer using single process on a single node.

Parameters

- **stat_server** – sufficient statistics stored in a StatServer
- **ubm** – Mixture object (the UBM)
- **uncertainty** – boolean, if True, return an additional matrix with uncertainty matrices (diagonal of the matrices)

Returns a StatServer with i-vectors in the stat1 attribute and a matrix of uncertainty matrices (optional)

plda(*stat_server*, *rank_f*, *nb_iter=10*, *scaling_factor=1.0*, *output_file_name=None*, *save_partial=False*)

Train a simplified Probabilistic Linear Discriminant Analysis model (no within class covariance matrix but full residual covariance matrix)

Parameters

- **stat_server** – StatServer object with training statistics
- **rank_f** – rank of the between class covariance matrix
- **nb_iter** – number of iterations to run
- **scaling_factor** – scaling factor to downscale statistics (value bewteen 0 and 1)
- **output_file_name** – name of the output file where to store PLDA model
- **save_partial** – boolean, if True, save PLDA model after each iteration

static read(*input_filename*)

Read a generic FactorAnalyser model from a HDF5 file

Parameters **input_filename** – the name of the file to read from

Returns a FactorAnalyser object

total_variability(*stat_server_filename*, *ubm*, *tv_rank*, *nb_iter=20*, *min_div=True*, *tv_init=None*, *batch_size=300*, *save_init=False*, *output_file_name=None*, *num_thread=1*)

Train a total variability model using multiple process on a single node. this method is the recommended one to train a Total Variability matrix.

Optimization: Only half of symmetric matrices are stored here process sessions per batch in order to control the memory footprint Batches are processed by a pool of workers running in different process The implementation is based on a multiple producers / single consumer approach

Parameters

- **stat_server_filename** – a list of StatServer file names to process
- **ubm** – a Mixture object
- **tv_rank** – rank of the total variability model
- **nb_iter** – number of EM iteration
- **min_div** – boolean, if True, apply minimum divergence re-estimation
- **tv_init** – initial matrix to start the EM iterations with
- **batch_size** – size of batch to load in memory for each worker
- **save_init** – boolean, if True, save the initial matrix
- **output_file_name** – name of the file where to save the matrix
- **num_thread** – number of process to run in parallel

total_variability_raw(*stat_server*, *ubm*, *tv_rank*, *nb_iter=20*, *min_div=True*, *tv_init=None*, *save_init=False*, *output_file_name=None*)

Train a total variability model using a single process on a single node. This method is provided for didactic purpose and should not be used as it uses to much memory and is to slow. If you want to use a single process run: “total_variability_single”

Parameters

- **stat_server** – the StatServer containing data to train the model
- **ubm** – a Mixture object
- **tv_rank** – rank of the total variability model
- **nb_iter** – number of EM iteration
- **min_div** – boolean, if True, apply minimum divergence re-estimation
- **tv_init** – initial matrix to start the EM iterations with
- **save_init** – boolean, if True, save the initial matrix
- **output_file_name** – name of the file where to save the matrix

total_variability_single(*stat_server_filename*, *ubm*, *tv_rank*, *nb_iter=20*, *min_div=True*, *tv_init=None*, *batch_size=300*, *save_init=False*, *output_file_name=None*)

Train a total variability model using a single process on a single node. Use this method to run a single process on a single node with optimized code.

Optimization: Only half of symmetric matrices are stored here process sessions per batch in order to control the memory footprint

Parameters

- **stat_server_filename** – the name of the file for StatServer, containing data to train the model
- **ubm** – a Mixture object
- **tv_rank** – rank of the total variability model
- **nb_iter** – number of EM iteration
- **min_div** – boolean, if True, apply minimum divergence re-estimation
- **tv_init** – initial matrix to start the EM iterations with
- **batch_size** – number of sessions to process at once to reduce memory footprint
- **save_init** – boolean, if True, save the initial matrix
- **output_file_name** – name of the file where to save the matrix

FeaturesExtractor

class features_extractor.**FeaturesExtractor**(*audio_filename_structure=None*, *feature_filename_structure=None*, *sampling_frequency=None*, *lower_frequency=None*, *higher_frequency=None*, *filter_bank=None*, *filter_bank_size=None*, *window_size=None*, *shift=None*, *ceps_number=None*, *vad=None*, *snr=None*, *pre_emphasis=None*, *save_param=None*, *keep_all_features=None*, *feature_type=None*, *rasta_plp=None*)

A FeaturesExtractor process an audio file in SPHERE, WAVE or RAW PCM format and extract filter-banks, cepstral coefficients, bottle-neck features (in the future), log-energy and perform a speech activity detection.

extract(*show*, *channel*, *input_audio_filename=None*, *output_feature_filename=None*, *backing_store=False*)

Compute the acoustic parameters (filter banks, cepstral coefficients, log-energy and bottleneck features for a single channel from a given audio file.

Parameters

- **show** – ID if the show
- **channel** – channel number (0 if mono file)
- **input_audio_filename** – name of the input audio file to consider if the name of the audio file is independent from the ID of the show
- **output_feature_filename** – name of the output feature file to consider if the name of the feature file is independent from the ID of the show
- **backing_store** – boolean, if False, nothing is writen to disk, if True, the file is writen to disk when closed
- **feature_type** – can be mfcc or plp
- **rasta** – boolean, only for PLP parameters, if True, perform RASTA filtering

Returns an hdf5 file handler

save(*show*, *channel=0*, *input_audio_filename=None*, *output_feature_filename=None*)

Compute the acoustic parameters (filter banks, cepstral coefficients, log-energy and bottleneck features for a single channel from a given audio file and save them to disk in a HDF5 format

Parameters

- **show** –
- **channel** –
- **input_audio_filename** –
- **output_feature_filename** –

Returns

save_list(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns

save_multispeakers(*idmap*, *channel=0*, *input_audio_filename=None*, *output_feature_filename=None*, *keep_all=True*, *skip_existing_file=False*)

Parameters

- **idmap** –
- **channel** –
- **input_audio_filename** –
- **output_feature_filename** –
- **keep_all** –
- **skip_existing_file** –

Returns

FeaturesServer

class features_server.**FeaturesServer**(*features_extractor=None*, *feature_filename_structure=None*, *sources=None*, *dataset_list=None*, *mask=None*, *feat_norm=None*, *global_cmvn=None*, *dct_pca=False*, *dct_pca_config=None*, *sdc=False*, *sdc_config=None*, *delta=None*, *double_delta=None*, *delta_filter=None*, *context=None*, *traps_dct_nb=None*, *rasta=None*, *keep_all_features=True*)

Management of features. FeaturesServer instances load datasets from a HDF5 files (that can be read from disk or produced by a FeaturesExtractor object) Datasets read from one or many files are concatenated and processed

get_context(*feat*, *start=None*, *stop=None*, *label=None*)

Add a left annd right context to each frame. First and last frames are duplicated to provide context at the begining and at the end

Parameters

- **feat** – sequence of feature frames (one fame per line)
- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment
- **label** – vad label if available

Returns a sequence of frames with their left and right context

get_features(*show*, *channel=0*, *input_feature_filename=None*, *label=None*, *start=None*, *stop=None*)

Get the datasets from a single HDF5 file The HDF5 file is loaded from disk or processed on the fly via the FeaturesExtractor of the current FeaturesServer

Parameters

- **show** – ID of the show
- **channel** – index of the channel to read
- **input_feature_filename** – name of the input file in case it does not include the ID of the show
- **label** – vad labels
- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment

Returns acoustic parameters and their vad labels

get_tandem_features(*show*, *channel=0*, *label=None*, *start=None*, *stop=None*)

Read acoustic parameters from multiple HDF5 files (from disk or extracted by FeaturesExtractor objects).

Parameters

- **show** – Id of the show
- **channel** – index of the channel
- **label** – vad labels

- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment

Returns acoustic parameters and their vad labels

get_traps(*feat*, *start=None*, *stop=None*, *label=None*)
Compute TRAP parameters. The input frames are concatenated to add their left and right context, a Hamming window is applied and a DCT reduces the dimensionality of the resulting vector.

Parameters

- **feat** – input acoustic parameters to process
- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment
- **label** – vad label if available

Returns a sequence of TRAP parameters

load(*show*, *channel=0*, *input_feature_filename=None*, *label=None*, *start=None*, *stop=None*)
Depending of the setting of the FeaturesServer, can either:

1. **Get the datasets from a single HDF5 file** The HDF5 file is loaded from disk or processed on the fly via the FeaturesExtractor of the current FeaturesServer
2. **Load datasets from multiple input HDF5 files. The datasets are post-processed separately, then concatenated** and post-process

Parameters

- **show** – ID of the show to load (should be the same for each HDF5 file to read from)
- **channel** – audio channel index in case the parameters are extracted from an audio file
- **input_feature_filename** – name of the input feature file in case it is independent from the ID of the show
- **label** – vad labels
- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment

Returns acoustic parameters and their vad labels

mean_std(*show*, *channel=0*, *start=None*, *stop=None*)
Compute the mean and standard deviation vectors for a segment of acoustic features

Parameters

- **show** – the ID of the show
- **channel** – the index of the channel
- **start** – index of the first frame of the selected segment
- **stop** – index of the last frame of the selected segment

Returns the number of frames, the mean of the frames and their standard deviation

post_processing(*feat*, *label*, *global_mean=None*, *global_std=None*)
After cepstral coefficients, filter banks or bottleneck parameters are computed or read from file post processing is applied.

Parameters

- **feat** – the matrix of acoustic parameters to post-process
- **label** – the VAD labels for the acoustic parameters
- **global_mean** – vector or mean to use for normalization
- **global_std** – vector of standard deviation to use for normalization

Returns the matrix of acoustic parameters ingand their VAD labels after post-process

stack_features(*show_list*, *channel_list=None*, *feature_filename_list=None*, *label_list=None*, *start_list=None*, *stop_list=None*)

Load acoustic features from a list of fils and return them stacked in a 2D-array one line per frame.

Parameters

- **show_list** –
- **channel_list** –
- **label_list** –
- **start_list** –
- **stop_list** –

Returns

stack_features_parallel(*show_list*, *channel_list=None*, *feature_filename_list=None*, *label_list=None*, *start_list=None*, *stop_list=None*, *num_thread=1*)

Load a list of feature files and stack them in a unique ndarray. The list of files to load is splited in sublists processed in parallel

Parameters

- **fileList** – a list of files to load
- **numThread** – numbe of thead (optional, default is 1)

Mixture

class mixture.**Mixture**(*mixture_file_name=''*, *name='empty'*)

A class for Gaussian Mixture Model storage. For more details about Gaussian Mixture Models (GMM) you can refer to *[Bimbot04]*.

Attr w array of weight parameters

Attr mu ndarray of mean parameters, each line is one distribution

Attr invcov ndarray of inverse co-variance parameters, 2-dimensional for diagonal co-variance distribution 3-dimensional for full co-variance

Attr invchol 3-dimensional ndarray containing upper cholesky decomposition of the inverse co-variance matrices

Attr cst array of constant computed for each distribution

Attr det array of determinant for each distribution

EM_diag2full(*diagonal_mixture*, *features_server*, *featureList*, *iterations=2*, *num_thread=1*)

Expectation-Maximization estimation of the Mixture parameters.

Parameters

- **features_server** – sidekit.FeaturesServer used to load data

- **featureList** – list of feature files to train the GMM
- **iterations** – list of iteration number for each step of the learning process
- **num_thread** – number of thread to launch for parallel computing

Return llk a list of log-likelihoods obtained after each iteration

EM_split(*features_server*, *feature_list*, *distrib_nb*, *iterations=(1, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8)*, *num_thread=1*, *llk_gain=0.01*, *save_partial=False*, *output_file_name='ubm'*, *ceil_cov=10*, *floor_cov=0.01*)

Expectation-Maximization estimation of the Mixture parameters.

Parameters

- **features_server** – sidekit.FeaturesServer used to load data
- **feature_list** – list of feature files to train the GMM
- **distrib_nb** – final number of distributions
- **iterations** – list of iteration number for each step of the learning process
- **num_thread** – number of thread to launch for parallel computing
- **llk_gain** – limit of the training gain. Stop the training when gain between two iterations is less than this value
- **save_partial** – name of the file to save intermediate mixtures, if True, save before each split of the distributions
- **ceil_cov** –
- **floor_cov** –

Return llk a list of log-likelihoods obtained after each iteration

EM_uniform(*cep*, *distrib_nb*, *iteration_min=3*, *iteration_max=10*, *llk_gain=0.01*, *do_init=True*)

Expectation-Maximization estimation of the Mixture parameters.

Parameters

- **cep** – set of feature frames to consider
- **cep** – set of feature frames to consider
- **distrib_nb** – number of distributions
- **iteration_min** – minimum number of iterations to perform
- **iteration_max** – maximum number of iterations to perform
- **llk_gain** – gain in term of likelihood, stop the training when the gain is less than this value
- **do_init** – boolean, if True initialize the GMM from the training data

Return llk a list of log-likelihoods obtained after each iteration

compute_log_posterior_probabilities(*cep*, *mu=None*)

Compute log posterior probabilities for a set of feature frames.

Parameters

- **cep** – a set of feature frames in a ndarray, one feature per row
- **mu** – a mean super-vector to replace the ubm's one. If it is an empty vector, use the UBM

Returns A ndarray of log-posterior probabilities corresponding to the input feature set.

compute_log_posterior_probabilities_full(*cep*, *mu=None*)
Compute log posterior probabilities for a set of feature frames.

Parameters

- **cep** – a set of feature frames in a ndarray, one feature per row
- **mu** – a mean super-vector to replace the ubm's one. If it is an empty vector, use the UBM

Returns A ndarray of log-posterior probabilities corresponding to the input feature set.

dim()
Return the dimension of distributions of the Mixture

Returns an integer, size of the acoustic vectors

distrib_nb()
Return the number of distribution of the Mixture

Returns the number of distribution in the Mixture

get_distrib_nb()
Return the number of Gaussian distributions in the mixture :return: then number of distributions

get_invcov_super_vector()
Return Inverse covariance super-vector

Returns an array, super-vector of the inverse co-variance coefficients

get_mean_super_vector()
Return mean super-vector

Returns an array, super-vector of the mean coefficients

init_from_diag(*diag_mixture*)

Parameters **diag_mixture** –

merge(*model_list*)
Merge a list of Mixtures into a new one. Weights are normalized uniformly :param model_list: a list of Mixture objects to merge

read(*mixture_file_name*, *prefix=''*)
Read a Mixture in hdf5 format

Parameters

- **mixture_file_name** – name of the file to read from
- **prefix** –

static read_alize(*file_name*)

Parameters **file_name** –

Returns

static read_htk(*filename*, *begin_hmm=False*, *state2=False*)
Read a Mixture in HTK format

Parameters

- **filename** – name of the file to read from
- **begin_hmm** – boolean
- **state2** – boolean

sv_size()
Return the dimension of the super-vector

Returns an integer, size of the mean super-vector

validate()
Verify the format of the Mixture

Returns a boolean giving the status of the Mixture

static variance_control(*cov*, *flooring*, *ceiling*, *cov_ctl*)
variance_control for Mixture (florring and ceiling)

Parameters

- **cov** – covariance to control
- **flooring** – float, florring value
- **ceiling** – float, ceiling value
- **cov_ctl** – co-variance to consider for flooring and ceiling

StatServer

class statserver.**StatServer**(*statserver_file_name=None*, *distrib_nb=0*, *feature_size=0*, *index=None*)
A class for statistic storage and processing

Attr modelset list of model IDs for each session as an array of strings

Attr segset the list of session IDs as an array of strings

Attr start index of the first frame of the segment

Attr stop index of the last frame of the segment

Attr stat0 a ndarray of float64. Each line contains 0-order statistics from the corresponding session

Attr stat1 a ndarray of float64. Each line contains 1-order statistics from the corresponding session

accumulate_stat(**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

Returns

adapt_mean_map(*ubm*, *r=16*, *norm=False*)

Maximum A Posteriori adaptation of the mean super-vector of ubm, train one model per segment.

Parameters

- **ubm** – a Mixture object to adapt
- **r** – float, the relevant factor for MAP adaptation
- **norm** – boolean, normalize by using the UBM co-variance. Default is False

Returns a StatServer with 1 as stat0 and the MAP adapted super-vectors as stat1

adapt_mean_map_multisession(*ubm*, *r=16*, *norm=False*)

Maximum A Posteriori adaptation of the mean super-vector of ubm, train one model per model in the modelset by summing the statistics of the multiple segments.

Parameters

- **ubm** – a Mixture object to adapt
- **r** – float, the relevant factor for MAP adaptation
- **norm** – boolean, normalize by using the UBM co-variance. Default is False

Returns a StatServer with 1 as stat0 and the MAP adapted super-vectors as stat1

align_models(*model_list*)

Align models of the current StatServer to match a list of models provided as input parameter. The size of the StatServer might be reduced to match the input list of models.

Parameters **model_list** – ndarray of strings, list of models to match

align_segments(*segment_list*)

Align segments of the current StatServer to match a list of segment provided as input parameter. The size of the StatServer might be reduced to match the input list of segments.

Parameters **segment_list** – ndarray of strings, list of segments to match

center_stat1(*mu*)

Center first orde statistics.

Parameters **mu** – array to center on.

estimate_between_class(*itNb*, *V*, *mean*, *sigma_obs*, *batch_size=100*, *Ux=None*, *Dz=None*, *minDiv=True*, *num_thread=1*, *re_estimate_residual=False*, *save_partial=False*)

Estimate the factor loading matrix for the between class covariance

Parameters

- **itNb** –
- **V** – initial between class covariance matrix
- **mean** – global mean vector
- **sigma_obs** – covariance matrix of the input data
- **batch_size** – size of the batches to process one by one to reduce the memory usage
- **Ux** – statserver of supervectors
- **Dz** – statserver of supervectors
- **minDiv** – boolean, if True run the minimum divergence step after maximization
- **num_thread** – number of parallel process to run
- **re_estimate_residual** – boolean, if True the residual covariance matrix is re-estimated (for PLDA)
- **save_partial** – boolean, if True, save FA model for each iteration

Returns the within class factor loading matrix

estimate_hidden(*mean*, *sigma*, *V=None*, *U=None*, *D=None*, *batch_size=100*, *num_thread=1*)
Assume that the statistics have not been whitened :param mean: global mean of the data to subtract :param sigma: residual covariance matrix of the Factor Analysis model :param V: between class covariance matrix :param U: within class covariance matrix :param D: MAP covariance matrix :param batch_size: size of the batches used to reduce memory footprint :param num_thread: number of parallel process to run

estimate_map(*itNb*, *D*, *mean*, *Sigma*, *Vy=None*, *Ux=None*, *num_thread=1*, *save_partial=False*)

Parameters

- **itNb** – number of iterations to estimate the MAP covariance matrix
- **D** – Maximum a Posteriori marix to estimate
- **mean** – mean of the input parameters
- **Sigma** – residual covariance matrix
- **Vy** – statserver of supervectors
- **Ux** – statserver of supervectors
- **num_thread** – number of parallel process to run
- **save_partial** – boolean, if True save MAP matrix after each iteration

Returns the MAP covariance matrix into a vector as it is diagonal

estimate_spectral_norm_stat1(*it=1*, *mode='efr'*)

Compute meta-parameters for Spectral Normalization as described in *[Bousquet11]*

Can be used to perform Eigen Factor Radial or Spherical Nuisance Normalization. Default behavior is equivalent to Length Norm as described in *[Garcia-Romero11]*

Statistics are transformed while the meta-parameters are estimated.

Parameters

- **it** – integer, number of iterations to perform
- **mode** – string, can be - efr for Eigen Factor Radial - sphNorm, for Spherical Nuisance Normalization

Returns a tupple of two lists: - a list of mean vectors - a list of co-variance matrices as ndarrays

estimate_within_class(*it_nb*, *U*, *mean*, *sigma_obs*, *batch_size=100*, *Vy=None*, *Dz=None*, *min_div=True*, *num_thread=1*, *save_partial=False*)
Estimate the factor loading matrix for the within class covariance

Parameters

- **it_nb** – number of iterations to estimate the within class covariance matrix
- **U** – initial within class covariance matrix
- **mean** – mean of the input data
- **sigma_obs** – co-variance matrix of the input data
- **batch_size** – number of sessions to process per batch to optimize memory usage
- **Vy** – statserver of supervectors
- **Dz** – statserver of supervectors
- **min_div** – boolean, if True run the minimum divergence step after maximization

- **num_thread** – number of parallel process to run
- **save_partial** – boolean, if True, save FA model for each iteration

Returns the within class factor loading matrix

factor_analysis(*rank_f*, *rank_g=0*, *rank_h=None*, *re_estimate_residual=False*, *it_nb=(10, 10, 10)*, *min_div=True*, *ubm=None*, *batch_size=100*, *num_thread=1*, *save_partial=False*, *init_matrices=(None, None, None)*)

Parameters

- **rank_f** – rank of the between class variability matrix
- **rank_g** – rank of the within class variab1ility matrix
- **rank_h** – boolean, if True, estimate the residual covariance matrix. Default is False
- **re_estimate_residual** – boolean, if True, the residual covariance matrix is re-estimated (use for PLDA)
- **it_nb** – tupple of three integers; number of iterations to run for F, G, H estimation
- **min_div** – boolean, if True, re-estimate the covariance matrices according to the minimum divergence criteria
- **batch_size** – number of sessions to process in one batch or memory optimization
- **num_thread** – number of thread to run in parallel
- **ubm** – origin of the space; should be None for PLDA and be a Mixture object for JFA or TV
- **save_partial** – name of the file to save intermediate models, if True, save before each split of the distributions
- **init_matrices** – tuple of three optional matrices to initialize the model, default is (None, None, None)

Returns three matrices, the between class factor loading matrix, the within class factor loading matrix the diagonal MAP matrix (as a vector) and the residual covariance matrix

generator()

Create a generator which yield stat0, stat1, of one session at a time

get_between_covariance_stat1()

Compute and return the between-class covariance matrix of the first-order statistics.

Returns the between-class co-variance matrix of the first-order statistics as a ndarray.

get_lda_matrix_stat1(*rank*)

Compute and return the Linear Discriminant Analysis matrix on the first-order statistics. Columns of the LDA matrix are ordered according to the corresponding eigenvalues in descending order.

Parameters **rank** – integer, rank of the LDA matrix to return

Returns the LDA matrix of rank "rank" as a ndarray

get_mahalanobis_matrix_stat1()

Compute and return Mahalanobis matrix of first-order statistics.

Returns the mahalanobis matrix computed on the first-order statistics as a ndarray

get_mean_stat1()
Return the mean of first order statistics

return: the mean array of the first order statistics.

get_model_segments(*mod_id*)
Return the list of segments belonging to model modID

Parameters **mod_id** – string, ID of the model which belonging segments will be returned

Returns a list of segments belonging to the model

get_model_segments_by_index(*mod_idx*)
Return the list of segments belonging to model number modIDX

Parameters **mod_idx** – index of the model which list of segments will be returned

Returns a list of segments belonging to the model

get_model_stat0(*mod_id*)
Return zero-order statistics of a given model

Parameters **mod_id** – ID of the model which stat0 will be returned

Returns a matrix of zero-order statistics as a ndarray

get_model_stat0_by_index(*mod_idx*)
Return zero-order statistics of model number modIDX

Parameters **mod_idx** – integer, index of the unique model which stat0 will be returned

Returns a matrix of zero-order statistics as a ndarray

get_model_stat1(*mod_id*)
Return first-order statistics of a given model

Parameters **mod_id** – string, ID of the model which stat1 will be returned

Returns a matrix of first-order statistics as a ndarray

get_model_stat1_by_index(*mod_idx*)
Return first-order statistics of model number modIDX

Parameters **mod_idx** – integer, index of the unique model which stat1 will be returned

Returns a matrix of first-order statistics as a ndarray

get_nap_matrix_stat1(*co_rank*)

Compute return the Nuisance Attribute Projection matrix from first-order statistics.

Parameters **co_rank** – co-rank of the Nuisance Attribute Projection matrix

Returns the NAP matrix of rank "coRank"

get_segment_stat0(*seg_id*)
Return zero-order statistics of segment which ID is segID

Parameters **seg_id** – string, ID of the segment which stat0 will be returned

Returns a matrix of zero-order statistics as a ndarray

get_segment_stat0_by_index(*seg_idx*)
Return zero-order statistics of segment number segIDX

Parameters **seg_idx** – integer, index of the unique segment which stat0 will be returned

Returns a matrix of zero-order statistics as a ndarray

get_segment_stat1(*seg_id*)
Return first-order statistics of segment which ID is segID

Parameters **seg_id** – string, ID of the segment which stat1 will be returned

Returns a matrix of first-order statistics as a ndarray

get_segment_stat1_by_index(*seg_idx*)
Return first-order statistics of segment number segIDX

Parameters **seg_idx** – integer, index of the unique segment which stat1 will be returned

Returns a matrix of first-order statistics as a ndarray

get_total_covariance_stat1()

Compute and return the total covariance matrix of the first-order statistics.

Returns the total co-variance matrix of the first-order statistics as a ndarray.

get_wccn_choleski_stat1()

Compute and return the lower Cholesky decomposition matrix of the Within Class Co-variance Normalization matrix on the first-order statistics.

Returns the lower Choleski decomposition of the WCCN matrix as a ndarray

get_within_covariance_stat1()

Compute and return the within-class covariance matrix of the first-order statistics.

Returns the within-class co-variance matrix of the first-order statistics as a ndarray.

ivector_extraction_eigen_decomposition(*ubm*, *Q*, *D_bar_c*, *Tnorm*, *delta=array([], dtype=float64)*)

Compute i-vectors using the eigen decomposition approximation. For more information, refers to[Glembeck09]_

Parameters

- **ubm** – a Mixture used as UBM for i-vector estimation
- **Q** – Q matrix as described in [Glembeck11]
- **D_bar_c** – matrices as described in [Glembeck11]
- **Tnorm** – total variability matrix pre-normalized using the co-variance of the UBM
- **delta** – men vector if re-estimated using minimum divergence criteria

Returns a StatServer which zero-order statistics are 1 and first-order statistics are approximated i-vectors.

ivector_extraction_weight(*ubm*, *W*, *Tnorm*, *delta=array([]*, *dtype=float64)*)

Compute i-vectors using the ubm weight approximation. For more information, refers to:

Glembeck, O.; Burget, L.; Matejka, P.; Karafiat, M. & Kenny, P. “Simplification and optimization of I-Vector extraction,” in IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP, 2011, 4516-4519

Parameters

- **ubm** – a Mixture used as UBM for i-vector estimation
- **W** – fix matrix pre-computed using the weights from the UBM and the total variability matrix
- **Tnorm** – total variability matrix pre-normalized using the co-variance of the UBM
- **delta** – men vector if re-estimated using minimum divergence criteria

Returns a StatServer which zero-order statistics are 1 and first-order statistics are approximated i-vectors.

mean_stat_per_model()
Average the zero- and first-order statistics per model and store them in a new StatServer.

Returns a StatServer with the statistics averaged per model

merge(**arg*)
Merge a variable number of StatServers into one. If a pair segmentID is duplicated, keep ony one of them and raises a WARNING

norm_stat1()
Divide all first-order statistics by their euclidian norm.

precompute_svm_kernel_stat1()

Pre-compute the Kernel for SVM training and testing, the output parameter is a matrix that only contains the impostor part of the Kernel. This one has to be completed by the target-dependent part during training and testing.

Returns the impostor part of the SVM Graam matrix as a ndarray

static read(*statserver_file_name*, *prefix=''*)
Read StatServer in hdf5 format

Parameters

- **statserver_file_name** – name of the file to read from
- **prefix** – prefixe of the dataset to read from in HDF5 file

static read_subset(*statserver_filename*, *index*, *prefix=''*)
Given a statserver in HDF5 format stored on disk and an IdMap, create a StatServer object filled with sessions corresponding to the IdMap.

Parameters

- **statserver_filename** – name of the statserver in hdf5 format to read from
- **index** – the IdMap of sessions to load or an array of index to load
- **prefix** – prefix of the group in HDF5 file

Returns a StatServer

rotate_stat1(*R*)
Rotate first-order statistics by a right-product.

Parameters **R** – ndarray, matrix to use for right product on the first order statistics.

spectral_norm_stat1(*spectral_norm_mean*, *spectral_norm_cov*, *is_sqr_inv_sigma=False*)

Apply Spectral Sormalization to all first order statistics. See more details in *[Bousquet11]*

The number of iterations performed is equal to the length of the input lists.

Parameters

- **spectral_norm_mean** – a list of mean vectors
- **spectral_norm_cov** – a list of co-variance matrices as ndarrays
- **is_sqr_inv_sigma** – boolean, True if

subtract_weighted_stat1 (*sts*)
Subtract the stat1 from from the sts StatServer to the stat1 of the current StatServer after multiplying by the zero-order statistics from the current statserver

Parameters **sts** – a StatServer

Returns a new StatServer

sum_stat_per_model ()
Sum the zero- and first-order statistics per model and store them in a new StatServer.

Returns a StatServer with the statistics summed per model

validate (*warn=False*)
Validate the structure and content of the StatServer. Check consistency between the different attributes of the StatServer: - dimension of the modelset - dimension of the segset - length of the modelset and segset - consistency of stat0 and stat1

Parameters **warn** – bollean optional, if True, display possible warning

whiten_cholesky_stat1 (*mu*, *sigma*)
Whiten first-order statistics by using Cholesky decomposition of Sigma

Parameters

- **mu** – array, mean vector to be subtracted from the statistics
- **sigma** – narray, co-variance matrix or covariance super-vector

whiten_stat1 (*mu*, *sigma*, *isSqrInvSigma=False*)
Whiten first-order statistics If sigma.ndim == 1, case of a diagonal covariance If sigma.ndim == 2, case of a single Gaussian with full covariance If sigma.ndim == 3, case of a full covariance UBM

Parameters

- **mu** – array, mean vector to be subtracted from the statistics
- **sigma** – narray, co-variance matrix or covariance super-vector
- **isSqrInvSigma** – boolean, True if the input Sigma matrix is the inverse of the square root of a covariance matrix

1.4.2 Additional modules

sidekit_io

Copyright 2014-2017 Anthony Larcher

sidekit_io provides methods to read and write from and to different formats.

sidekit_io.**h5merge**(*output_filename*, *input_filename_list*)
Merge a list of HDF5 files into a new one.

Parameters

- **output_filename** – the name of the new file resulting from the merge.
- **input_filename_list** – list of thge input files

sidekit_io.**init_logging**(*level=20*, *filename=None*)
Initialize a logger

Parameters

- **level** – level of messages to catch
- **filename** – name of the output file

sidekit_io.**read_dict_hdf5**(*input_filename*)
Read a dictionary from an HDF5 file.

Parameters **input_filename** – name of the file to read from

Returns the dictionary

sidekit_io.**read_fa_hdf5**(*input_filename*)
Read a generic FA model from a HDF5 file

Parameters **input_filename** – the name of the file to read from

Returns a tuple of 5 elements: the mean vector, the between class covariance matrix, the within class covariance matrix, the MAP matrix and the residual covariancematrix

sidekit_io.**read_key_hdf5**(*input_filename*, *key*)
Read key value from a HDF5 file.

Parameters

- **input_filename** – the name of the file to read from
- **key** – the name of the key

Returns a value

sidekit_io.**read_matrix**(*filename*)
Read matrix in ALIZE binary format and return a ndarray

Parameters **filename** – name of the file to read from

Returns a numpy.ndarray object

sidekit_io.**read_norm_hdf5**(*input_filename*)
Read normalization parameters from a HDF5 file.

Parameters **input_filename** – the name of the file to read from

Returns a tuple of two lists. The first list contains mean vectors for each iteration, the second list contains covariance matrices for each iteration

sidekit_io.**read_pickle**(*filename*)
Read a generic pickle file and return the content

Parameters **filename** – name of the pickle file to read

Returns the content of the file

sidekit_io.**read_plda_hdf5**(*input_filename*)
Read a PLDA model from a HDF5 file.

Parameters **input_filename** – the name of the file to read from

Returns a tuple of 4 elements: the mean vector, the between class covariance matrix, the within class covariance matrix and the residual matrix

sidekit_io.**read_tv_hdf5**(*input_filename*)
Read the TotalVariability matrix, the mean and the residual covariance from a HDF5 file.

Parameters **input_filename** – name of the file to read from

Returns a tuple of three elements: the matrix, the mean vector and the inverse covariance vector

sidekit_io.**read_vect**(*filename*)
Read vector in ALIZE binary format and return an array

Parameters **filename** – name of the file to read from

Returns a numpy.ndarray object

iv_scoring

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

iv_scoring provides methods to compare i-vectors

iv_scoring.**PLDA_scoring**(*enroll*, *test*, *ndx*, *mu*, *F*, *G*, *Sigma*, *test_uncertainty=None*, *Vtrans=None*, *p_known=0.0*, *scaling_factor=1.0*, *full_model=False*)
Compute the PLDA scores between to sets of vectors. The list of trials to perform is given in an Ndx object. PLDA matrices have to be pre-computed. i-vectors are supposed to be whitened before.

Implements the appraoch described in *[Lee13]* including scoring for partially open-set identification

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **mu** – the mean vector of the PLDA gaussian
- **F** – the between-class co-variance matrix of the PLDA
- **G** – the within-class co-variance matrix of the PLDA
- **Sigma** – the residual covariance matrix
- **p_known** – probability of having a known speaker for open-set identification case (=1 for the verification task and =0 for the closed-set case)
- **scaling_factor** – scaling factor to be multiplied by the sufficient statistics
- **full_model** – boolean, set to True when using a complete PLDA model (including within class covariance matrix)

Returns a score object

iv_scoring.**PLDA_scoring_uncertainty**(*enroll*, *test*, *ndx*, *mu*, *F*, *Sigma*, *p_known=0.0*, *scaling_factor=1.0*, *test_uncertainty=None*, *Vtrans=None*, *check_missing=True*)

Parameters

- **enroll** –
- **test** –

- **ndx** –
- **mu** –
- **F** –
- **Sigma** –
- **p_known** –
- **scaling_factor** –
- **test_uncertainty** –
- **Vtrans** –
- **check_missing** –

Returns

iv_scoring.**cosine_scoring**(*enroll*, *test*, *ndx*, *wccn=None*, *check_missing=True*)

Compute the cosine similarities between to sets of vectors. The list of trials to perform is given in an Ndx object.

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **wccn** – numpy.ndarray, if provided, the i-vectors are normalized by using a Within Class Covariance Matrix
- **check_missing** – boolean, if True, check that all models and segments exist

Returns a score object

iv_scoring.**fast_PLDA_scoring**(*enroll*, *test*, *ndx*, *mu*, *F*, *Sigma*, *test_uncertainty=None*, *Vtrans=None*, *p_known=0.0*, *scaling_factor=1.0*, *check_missing=True*)

Compute the PLDA scores between to sets of vectors. The list of trials to perform is given in an Ndx object. PLDA matrices have to be pre-computed. i-vectors are supposed to be whitened before.

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **mu** – the mean vector of the PLDA gaussian
- **F** – the between-class co-variance matrix of the PLDA
- **Sigma** – the residual covariance matrix
- **p_known** – probability of having a known speaker for open-set identification case (=1 for the verification task and =0 for the closed-set case)
- **check_missing** – boolean, if True, check that all models and segments exist

Returns a score object

iv_scoring.**full_PLDA_scoring**(*enroll*, *test*, *ndx*, *mu*, *F*, *G*, *Sigma*, *p_known=0.0*, *scaling_factor=1.0*, *check_missing=True*)

Compute PLDA scoring

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **mu** – the mean vector of the PLDA gaussian
- **F** – the between-class co-variance matrix of the PLDA
- **G** – the within-class co-variance matrix of the PLDA
- **Sigma** – the residual covariance matrix
- **p_known** – probability of having a known speaker for open-set identification case (=1 for the verification task and =0 for the closed-set case)
- **check_missing** – boolean, default is True, set to False not to check missing models

iv_scoring.**mahalanobis_scoring**(*enroll*, *test*, *ndx*, *m*, *check_missing=True*)

Compute the mahalanobis distance between to sets of vectors. The list of trials to perform is given in an Ndx object.

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **m** – mahalanobis matrix as a ndarray
- **check_missing** – boolean, default is True, set to False not to check missing models

Returns a score object

iv_scoring.**two_covariance_scoring**(*enroll*, *test*, *ndx*, *W*, *B*, *check_missing=True*)

Compute the 2-covariance scores between to sets of vectors. The list of trials to perform is given in an Ndx object. Within and between class co-variance matrices have to be pre-computed.

Parameters

- **enroll** – a StatServer in which stat1 are i-vectors
- **test** – a StatServer in which stat1 are i-vectors
- **ndx** – an Ndx object defining the list of trials to perform
- **W** – the within-class co-variance matrix to consider
- **B** – the between-class co-variance matrix to consider
- **check_missing** – boolean, default is True, set to False not to check missing models

Returns a score object

gmm_scoring

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

features_server provides methods to test gmm models

gmm_scoring.**gmm_scoring**(*ubm*, *enroll*, *ndx*, *feature_server*, *num_thread=1*)

Compute log-likelihood ratios for sequences of acoustic feature frames between a Universal Background Model (UBM) and a list of Gaussian Mixture Models (GMMs) which only mean vectors differ from the UBM.

Parameters

- **ubm** – a Mixture object used to compute the denominator of the likelihood ratios
- **enroll** – a StatServer object which stat1 attribute contains mean super-vectors of the GMMs to use to compute the numerator of the likelihood ratios.
- **ndx** – an Ndx object which define the list of trials to compute
- **feature_server** – a FeatureServer object to load the features
- **num_thread** – number of thread to launch in parallel

Returns a Score object.

gmm_scoring.**gmm_scoring_singleThread**(*ubm*, *enroll*, *ndx*, *feature_server*, *score_mat*, *seg_idx=None*)

Compute log-likelihood ratios for sequences of acoustic feature frames between a Universal Background Model (UBM) and a list of Gaussian Mixture Models (GMMs) which only mean vectors differ from the UBM.

Parameters

- **ubm** – a Mixture object used to compute the denominator of the likelihood ratios
- **enroll** – a StatServer object which stat1 attribute contains mean super-vectors of the GMMs to use to compute the numerator of the likelihood ratios.
- **ndx** – an Ndx object which define the list of trials to compute
- **feature_server** – sidekit.FeaturesServer used to load the acoustic parameters
- **score_mat** – a ndarray of scores to fill
- **seg_idx** – the list of unique test segments to process. Those test segments should belong to the list of test segments in the ndx object. By setting seg_idx=None, all test segments from the ndx object will be processed

sv_utils

This module provides miscellaneous tools that might be of use
to the different parts of a speaker recognition engine.

Copyright 2014-2017 Anthony Larcher

sv_utils provides utilities to facilitate the work with SIDEKIT.

sv_utils.**check_file_list**(*input_file_list*, *file_name_structure*)

Check the existence of a list of files in a specific directory Return a new list with the existing segments and a list of indices of those files in the original list. Return outputFileList and idx such that inputFileList[idx] = outputFileList

Parameters

- **input_file_list** – list of file names
- **file_name_structure** – structure of the filename to search for

Returns a list of existing files and the indices of the existing files in the input list

sv_utils.**clean_stat_server**(*statserver*)

Parameters **statserver** –

Returns

sv_utils.**initialize_iv_extraction_eigen_decomposition**(*ubm*, *T*)

Estimate matrices Q, D_bar_c and Tnorm, for approximation of the i-vectors. For more information, refers to *[Glembeck09]*

Parameters

- **ubm** – Mixture object, Universal Background Model
- **T** – Raw TotalVariability matrix

Returns Q: Q matrix as described in [Glembeck11] D_bar_c: matrices as described in [Glembeck11] Tnorm: total variability matrix pre-normalized using the co-variance of the UBM

sv_utils.**initialize_iv_extraction_fse**(*ubm*, *T*)

Estimate matrices for approximation of the i-vectors. For more information, refers to *[Cumani13]*

Parameters

- **ubm** – Mixture object, Universal Background Model
- **T** – Raw TotalVariability matrix

Returns Q: Q matrix as described in [Glembeck11] D_bar_c: matrices as described in [Glembeck11] Tnorm: total variability matrix pre-normalized using the co-variance of the UBM

sv_utils.**initialize_iv_extraction_weight**(*ubm*, *T*)

Estimate matrices W and T for approximation of the i-vectors For more information, refers to *[Glembeck09]*

Parameters

- **ubm** – Mixture object, Universal Background Model
- **T** – Raw TotalVariability matrix as a ndarray

Returns

W: fix matrix pre-computed using the weights from the UBM and the total variability matrix

Tnorm: total variability matrix pre-normalized using the co-variance of the UBM

sv_utils.**mean_std_many**(*features_server*, *seg_list*, *in_context=False*, *num_thread=1*)

Compute the mean and standard deviation from a list of segments.

Parameters

- **features_server** –
- **seg_list** – list of file names with start and stop indices
- **in_context** –
- **num_thread** –

Returns a tuple of three values, the number of frames, the mean and the variance

sv_utils.**parse_mask**(*mask*)

Parameters **mask** –

Returns

sv_utils.**read_svm**(*svm_file_name*)
Read SVM model in PICKLE format

Parameters **svm_file_name** – name of the file to read from

Returns a tupple of weight and biais

sv_utils.**save_svm**(*svm_file_name*, *w*, *b*)
Save SVM weights and bias in PICKLE format

Parameters

- **svm_file_name** – name of the file to write
- **w** – weight coefficients of the SVM to store
- **b** – biais of the SVM to store

sv_utils.**segment_mean_std_hdf5**(*input_segment*, *in_context=False*)
Compute the sum and square sum of all features for a list of segments. Input files are in HDF5 format

Parameters

- **input_segment** – list of segments to read from, each element of the list is a tuple of 5 values, the filename, the index of thefirst frame, index of the last frame, the number of frames for the left context and the number of frames for the right context
- **in_context** –

Returns a tuple of three values, the number of frames, the sum of frames and the sum of squares

svm_training

Copyright 2014-2017 Anthony Larcher

svm_training provides utilities to train Support Vector Machines to perform speaker verification.

svm_training.**svm_training**(*svmDir*, *background_sv*, *enroll_sv*, *num_thread=1*)
Train Suport Vector Machine classifiers for two classes task (as implemented for nowbut miht change in the future to include multi-class classification) Training is parallelized on multiple threads.

Parameters

- **svmDir** – directory where to store the SVM models
- **background_sv** – StatServer of super-vectors for background impostors. All super-vectors are used without selection
- **enroll_sv** – StatServer of super-vectors used for the target models
- **num_thread** – number of thread to launch in parallel

svm_training.**svm_training_singleThread**(*K*, *msn*, *bsn*, *svm_dir*, *background_sv*, *models*, *enroll_sv*)
Train Suport Vector Machine classifiers for two classes task (as implemented for nowbut miht change in the future to include multi-class classification)

Parameters

- **K** – pre-computed part of the Gram matrix
- **msn** – maximum number of sessions to train a SVM
- **bsn** – number of session used as background impostors
- **svm_dir** – directory where to store the SVM models

- **background_sv** – StatServer of super-vectors for background impostors. All super-vectors are used without selection
- **models** – list of models to train. The models must be included in the enroll_sv StatServer
- **enroll_sv** – StatServer of super-vectors used for the target models

svm_scoring

Copyright 2014-2017 Anthony Larcher

svm_scoring provides functions to perform speaker verification by using Support Vector Machines.

svm_scoring.**svm_scoring**(*svm_filename_structure*, *test_sv*, *ndx*, *num_thread=1*)
Compute scores for SVM verification on multiple threads (two classes only as implementeed at the moment)

Parameters

- **svm_filename_structure** – structure of the filename where to load the SVM models
- **test_sv** – StatServer object of super-vectors. stat0 are set to 1 and stat1 are the super-vector to classify
- **ndx** – Ndx object of the trials to perform
- **num_thread** – number of thread to launch in parallel

Returns a Score object.

svm_scoring.**svm_scoring_singleThread**(*svm_filename_structure*, *test_sv*, *ndx*, *score*, *seg_idx=None*)
Compute scores for SVM verification on a single thread (two classes only as implementeed at the moment)

Parameters

- **svm_filename_structure** – structure of the filename where to load the SVM models
- **test_sv** – StatServer object of super-vectors. stat0 are set to 1 and stat1 are the super-vector to classify
- **ndx** – Ndx object of the trials to perform
- **score** – Scores object to fill
- **seg_idx** – list of segments to classify. Classify all if the list is empty.

1.4.3 The bosaris package

This package is a translation of a part of the BOSARIS toolkit. The authors thank Niko Brummer and Agnitio for allowing them to translate this code and provide the community with efficient structures and tools.

The BOSARIS Toolkit is a collection of functions and classes in Matlab that can be used to calibrate, fuse and plot scores from speaker recognition (or other fields in which scores are used to test the hypothesis that two samples are from the same source) trials involving a model and a test segment. The toolkit was written at the BOSARIS2010 workshop which took place at the University of Technology in Brno, Czech Republic from 5 July to 6 August 2010. See the User Guide (available on the toolkit website)1 for a discussion of the theory behind the toolkit and descriptions of some of the algorithms used.

The BOSARIS toolkit in MATLAB can be downloaded from the website.

Content

The Python bosaris package released as part of the ** sidekit** contains six modules detailed below.

BOSARIS License

Agnitio Labs

Non-Commercial Use Only

This AGNITIO Labs License Agreement, including all exhibits ("AGN-LA") is a legal agreement between you and AGNITIO S. L. ("AGNITIO" or "we") for the software or data identified above, which may include source code, and any associated materials, text or speech files, associated media and "online" or electronic documentation and any updates we provide in our discretion (together, the "Software").

By installing, copying, or otherwise using this Software, you agree to be bound by the terms of this AGN-LA. If you do not agree, do not install copy or use the Software. The Software is protected by copyright and other intellectual property laws and is licensed, not sold.

SCOPE OF RIGHTS:

You may use, copy, reproduce, and distribute this Software for any non-commercial purpose, subject to the restrictions in this AGN-LA. Some purposes which can be non-commercial are teaching, academic research, public demonstrations and personal experimentation. You may also distribute this Software with books or other teaching materials, or publish the Software on websites, that are intended to teach the use of the Software for academic or other non-commercial purposes.

You may not use or distribute this Software or any derivative works in any form for commercial purposes. Examples of commercial purposes would be running business operations, licensing, leasing, or selling the Software, distributing the Software for use with commercial products, using the Software in the creation or use of commercial products or any other activity which purpose is to procure a commercial gain to you or others.

If the Software includes source code or data, you may create derivative works of such portions of the Software and distribute the modified Software for

non-commercial purposes, as provided herein.

If you distribute the Software or any derivative works of the Software, you
will distribute them under the same terms and conditions as in this license,
and you will not grant other rights to the Software or derivative works that
are different from those provided by this AGN-LA.

If you have created derivative works of the Software, and distribute such
derivative works, you will cause the modified files to carry prominent notices
so that recipients know that they are not receiving the original Software. Such
notices must state: (i) that you have changed the Software; and (ii) the date
of any changes.

In return, we simply require that you agree:

1. That you will not remove any copyright or other notices from the Software.

2. That if any of the Software is in binary format, you will not attempt to
modify such portions of the Software, or to reverse engineer or decompile them,
except and only to the extent authorized by applicable law.

3. That AGNITIO is granted back, without any restrictions or limitations, a
non-exclusive, perpetual, irrevocable, royalty-free, assignable and
sub-licensable license, to reproduce, publicly perform or display, install,
use, modify, post, distribute, make and have made, sell and transfer your
modifications to and/or derivative works of the Software source code or data,
for any purpose.

4. That any feedback about the Software provided by you to us is voluntarily
given, and AGNITIO shall be free to use the feedback as it sees fit without
obligation or restriction of any kind, even if the feedback is designated by
you as confidential.

5. THAT THE SOFTWARE COMES “AS IS”, WITH NO WARRANTIES. THIS MEANS NO EXPRESS,
IMPLIED OR STATUTORY WARRANTY, INCLUDING WITHOUT LIMITATION, WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ANY WARRANTY AGAINST
INTERFERENCE WITH YOUR ENJOYMENT OF THE SOFTWARE OR ANY WARRANTY OF TITLE OR
NON-INFRINGEMENT. THERE IS NO WARRANTY THAT THIS SOFTWARE WILL FULFILL ANY OF
YOUR PARTICULAR PURPOSES OR NEEDS. ALSO, YOU MUST PASS THIS DISCLAIMER ON
WHENEVER YOU DISTRIBUTE THE SOFTWARE OR DERIVATIVE WORKS.

6. THAT NEITHER AGNITIO NOR ANY CONTRIBUTOR TO THE SOFTWARE WILL BE LIABLE FOR
ANY DAMAGES RELATED TO THE SOFTWARE OR THIS AGN-LA, INCLUDING DIRECT, INDIRECT,
SPECIAL, CONSEQUENTIAL OR INCIDENTAL DAMAGES, TO THE MAXIMUM EXTENT THE LAW
PERMITS, NO MATTER WHAT LEGAL THEORY IT IS BASED ON. ALSO, YOU MUST PASS THIS

LIMITATION OF LIABILITY ON WHENEVER YOU DISTRIBUTE THE SOFTWARE OR DERIVATIVE WORKS.

7. That we have no duty of reasonable care or lack of negligence, and we are not obligated to (and will not) provide technical support for the Software.

8. That if you breach this AGN-LA or if you sue anyone over patents that you think may apply to or read on the Software or anyone's use of the Software, this AGN-LA (and your license and rights obtained herein) terminate automatically. Upon any such termination, you shall destroy all of your copies of the Software immediately. Sections 3, 4, 5, 6, 7, 8, 11 and 12 of this AGN-LA shall survive any termination of this AGN-LA.

9. That the patent rights, if any, granted to you in this AGN-LA only apply to | the Software, not to any derivative works you make.

10. That the Software may be subject to Europe export jurisdiction at the time it is licensed to you, and it may be subject to additional export or import laws in other places. You agree to comply with all such laws and regulations that may apply to the Software after delivery of the software to you.

11. That all rights not expressly granted to you in this AGN-LA are reserved.

12. That this AGN-LA shall be construed and controlled by the laws of the Kingdom of Spain, without regard to conflicts of law. If any provision of this AGN-LA shall be deemed unenforceable or contrary to law, the rest of this AGN-LA shall remain in full effect and interpreted in an enforceable manner that most nearly captures the intent of the original language.

DetPlot

class bosaris.**DetPlot**(*window_style='old'*, *plot_title=''*)

A class for creating a plot for displaying detection performance with the axes scaled and labelled so that a normal Gaussian distribution will plot as a straight line.

- The y axis represents the miss probability.
- The x axis represents the false alarm probability.

Attr __plotwindow__ PlotWindow object to plot into

Attr __title__ title of the plot

Attr __sys_name__ list of IDs of the systems

Attr __tar__ list of arrays of of target scores for each system

Attr __non__ list of arrays of the non-target scores for each system

Attr __figure__ figure to plot into

create_figure(*idx=0*)
Create a figure to plot the DET-curve. Default plot everything on one single figure

Parameters **idx** – Index of the figure to create. Default is 0.

plot_DR30_both(*idx=0*, *plot_args_fa=((0, 0, 0), '–', 1)*, *plot_args_miss=((0, 0, 0), '–', 1)*, *legend_string=''*)
Plots two lines indicating Doddington's Rule of 30 points: one for false alarms and one for misses. See the documentation of plot_DR30_fa and plot_DR30_miss for details.

Parameters

- **idx** – index of the figure to plot in
- **plot_args_fa** – A tuple of arguments to be passed to 'plot' that control the appearance of the DR30_fa point.
- **plot_args_miss** – A tuple of arguments to be passed to 'plot' that control the appearance of the DR30_miss point.
- **legend_string** – Optional. A string to describe this curve in the legend.

plot_DR30_fa(*idx=0*, *plot_args=((0, 0, 0), '–', 1)*, *legend_string=''*)
Plots a vertical line indicating the Doddington 30 point for false alarms. This is the point left of which the number of false alarms is below 30, so that the estimate of the false alarm rate is no longer good enough to satisfy Doddington's Rule of 30.

Parameters

- **idx** – index of the figure to plot in
- **plot_args** – A cell array of arguments to be passed to 'plot' that control the appearance of the curve.
- **legend_string** – Optional. A string to describe this curve in the legend.

plot_DR30_miss(*idx=0*, *plot_args=((0, 0, 0), '–', 1)*, *legend_string=''*)
Plots a horizontal line indicating the Doddington 30 point for misses. This is the point above which the number of misses is below 30, so that the estimate of the miss rate is no longer good enough to satisfy Doddington's Rule of 30.

Parameters

- **idx** – index of the figure to plot in
- **plot_args** – A cell array of arguments to be passed to 'plot' that control the appearance of the curve.
- **legend_string** – Optional. A string to describe this curve in the legend.

plot_mindcf_point(*target_prior*, *idx=0*, *plot_args='ok'*, *legend_string=''*)
Places the mindcf point for the current system.

Parameters

- **target_prior** – The effective target prior.
- **idx** – inde of the figure to plot in
- **plot_args** – a list of arguments to be passed to 'plot' that control the appearance of the curve.
- **legend_string** – Optional. A string to describe this curve in the legend.

plot_rocch_det(*idx=0*, *style='color'*, *target_prior=0.001*, *plot_args=''*)
Plots a DET curve using the ROCCH.

Parameters

- **idx** – index of the figure to plot on
- **style** – style of the DET-curve (see DetPlot description)
- **target_prior** – prior of the target trials
- **plot_args** – a list of arguments to be passed to plot that control the appearance of the curve.

plot_steppy_det (*idx=0*, *style='color'*, *plot_args=''*)
Plots a DET curve.

Parameters

- **idx** – the idx of the curve to plot in case tar and non have several dimensions
- **style** – style of the curve, can be gray or color
- **plot_args** – a cell array of arguments to be passed to plot that control the appearance of the curve.

set_system (*tar*, *non*, *sys_name=''*)
Sets the scores to be plotted. This function must be called before plots are made for a system, but it can be called several times with different systems (with calls to plotting functions in between) so that curves for different systems appear on the same plot.

Parameters

- **tar** – A vector of target scores.
- **non** – A vector of non-target scores.
- **sys_name** – A string describing the system. This string will be prepended to the plot names in the legend. You can pass an empty string to this argument or omit it.

set_system_from_scores (*scores*, *key*, *sys_name=''*)
Sets the scores to be plotted. This function must be called before plots are made for a system, but it can be called several times with different systems (with calls to plotting functions in between) so that curves for different systems appear on the same plot.

Parameters

- **scores** – A Scores object containing system scores.
- **key** – A Key object for distinguishing target and non-target scores.
- **sys_name** – A string describing the system. This string will be prepended to the plot names in the legend. You can pass an empty string to this argument or omit it.

set_title (*title*)
Modify the title of a DetPlot object

Parameters **title** – title of the plot to display

IdMap

class bosaris.**IdMap** (*idmap_filename=''*)
A class that stores a map between identifiers (strings). One list is called 'leftids' and the other 'rightids'. The class provides methods that convert a sequence of left ids to a sequence of right ids and vice versa. If *leftids* or *rightids* contains duplicates then all occurrences are used as the index when mapping.

Attr leftids a list of classes in a ndarray

Attr rightids a list of segments in a ndarray

Attr start index of the first frame of the segment

Attr stop index of the last frame of the segment

filter_on_left(*idlist*, *keep*)
Removes some of the information in an idmap. Depending on the value of 'keep', the idlist indicates the strings to retain or the strings to discard.

Parameters

- **idlist** – an array of strings which will be compared with the leftids of the current.
- **keep** – A boolean indicating whether idlist contains the ids to keep or to discard.

Returns a filtered version of the current IdMap.

filter_on_right(*idlist*, *keep*)
Removes some of the information in an idmap. Depending on the value of 'keep', the idlist indicates the strings to retain or the strings to discard.

Parameters

- **idlist** – an array of strings which will be compared with the rightids of the current IdMap.
- **keep** – a boolean indicating whether idlist contains the ids to keep or to discard.

Returns a filtered version of the current IdMap.

map_left_to_right(*leftidlist*)
Maps an array of ids to a new array of ids using the given map. The input ids are matched against the leftids of the map and the output ids are taken from the corresponding rightids of the map.

Beware: if leftids are not unique in the IdMap, only the last value corresponding is kept

Parameters **leftidlist** – an array of strings to be matched against the leftids of the idmap. The rightids corresponding to these leftids will be returned.

Returns an array of strings that are the mappings of the strings in leftidlist.

map_right_to_left(*rightidlist*)
Maps an array of ids to a new array of ids using the given map. The input ids are matched against the rightids of the map and the output ids are taken from the corresponding leftids of the map.

Beware: if rightids are not unique in the IdMap, only the last value corresponding is kept

Parameters **rightidlist** – An array of strings to be matched against the rightids of the idmap. The leftids corresponding to these rightids will be returned.

Returns an array of strings that are the mappings of the strings in rightidlist.

merge(*idmap2*)
Merges the current IdMap with another IdMap or a list of IdMap objects..

Parameters **idmap2** – Another Id_Map object.

Returns an Id_Map object that contains the information from the two input Id_Maps.

static read(*input_file_name*)
Read IdMap in hdf5 format.

Parameters **input_file_name** – name of the file to read from

validate(*warn=False*)
Checks that an object of type Id_Map obeys certain rules that must alows be true.

Parameters **warn** – boolean. If True, print a warning if strings are duplicated in either left or right array

Returns a boolean value indicating whether the object is valid.

Key

class bosaris.**Key**(*key_file_name=''*, *models=array([]*, *dtype=float64)*, *testsegs=array([]*, *dtype=float64)*, *trials=array([]*, *dtype=float64)*)

A class for representing a Key i.e. it classifies trials as target or non-target trials.

Attr modelset list of the models into a ndarray of strings

Attr segset list of the test segments into a ndarray of strings

Attr tar 2D ndarray of booleans which rows correspond to the models and columns to the test segments. True if target trial.

Attr non 2D ndarray of booleans which rows correspond to the models and columns to the test segments. True is non-target trial.

filter(*modlist*, *seglist*, *keep*)

Removes some of the information in a key. Useful for creating a gender specific key from a pooled gender key. Depending on the value of 'keep', the two input lists indicate the strings to retain or the strings to discard.

Parameters

- **modlist** – a cell array of strings which will be compared with the modelset of 'inkey'.
- **seglist** – a cell array of strings which will be compared with the segset of 'inkey'.
- **keep** – a boolean indicating whether modlist and seglist are the models to keep or discard.

Returns a filtered version of 'inkey'.

merge(*key_list*)

Merges Key objects. This function takes as input a list of Key objects to merge in the curent one.

Parameters **key_list** – the list of Keys to merge

static read(*input_file_fame*)

Reads a Key object from an hdf5 file.

Parameters **input_file_fame** – name of the file to read from

static read_txt(*input_file_name*)

Creates a Key object from information stored in a text file.

Parameters **input_file_name** – name of the file to read from

to_ndx()

Create a Ndx object based on the Key object

Returns a Ndx object based on the Key

validate()

Checks that an object of type Key obeys certain rules that must always be true.

Returns a boolean value indicating whether the object is valid.

Ndx

class bosaris.**Ndx**(*ndx_file_name=''*, *models=array([]*, *dtype=float64)*, *testsegs=array([]*, *dtype=float64)*)

A class that encodes trial index information. It has a list of model names and a list of test segment names and a matrix indicating which combinations of model and test segment are trials of interest.

Attr modelset list of unique models in a ndarray

Attr segset list of unique test segments in a ndarray

Attr trialmask 2D ndarray of boolean. Rows correspond to the models and columns to the test segments. True if the trial is of interest.

filter(*modlist*, *seglist*, *keep*)

Removes some of the information in an Ndx. Useful for creating a gender specific Ndx from a pooled gender Ndx. Depending on the value of 'keep', the two input lists indicate the strings to retain or the strings to discard.

Parameters

- **modlist** – a cell array of strings which will be compared with the modelset of 'inndx'.
- **seglist** – a cell array of strings which will be compared with the segset of 'inndx'.
- **keep** – a boolean indicating whether modlist and seglist are the models to keep or discard.

Returns a filtered version of the current Ndx object.

merge(*ndx_list*)

Merges a list of Ndx objects into the current one. The resulting ndx must have all models and segment in the input ndxs (only once). A trial in any ndx becomes a trial in the output ndx

Parameters **ndx_list** – list of Ndx objects to merge

static read(*input_file_name*)

Creates an Ndx object from the information in an hdf5 file.

Parameters **input_file_name** – name of the file to read from

validate()

Checks that an object of type Ndx obeys certain rules that must always be true.

Returns a boolean value indicating whether the object is valid

PlotWindow

class bosaris.**PlotWindow**(*input_type=''*)

A class that is used to define the parameters of a plotting window.

Attr __pfa_limits__ ndarray of two values that determine the limits of the pfa axis. Default is [0.0005, 0.5]

Attr __pmiss_limits__ ndarray of two values that determine the limits of the pmiss axis. Default is [0.0005, 0.5]

Attr __xticks__ coordonates of the ticks on the horizontal axis

Attr __xticklabels__ labels of the ticks on the horizontal axis in a ndarray of strings

Attr __yticks__ coordonates of the ticks on the vertical axis

Attr __yticklabels__ labels of the ticks on the vertical axis in a ndarray of strings

axis_big()
Set axis value to big ones

- pfa ranges from 0.000005 to 0.99
- pmiss ranges from 0.000005 to0.99

axis_new()
Set axis value to new ones

- pfa ranges from 0.000005 to 0.005
- pmiss ranges from 0.01 to 0.99

axis_old()
Set axis value to old ones (NIST-SRE08 style)

- pfa ranges from 0.0005 to 0.5
- pmiss ranges from 0.0005 to 0.5

axis_sre10()
Set axis value to NIST-SRE10 style

- pfa ranges from 0.000003 to 0.5
- pmiss ranges from 0.0003 to 0.9

make_plot_window_from_values(*pfa_limits*, *pmiss_limits*, *xticks*, *xticklabels*, *yticks*, *yticklabels*)
Initialize PlotWindow from provided values

Parameters

- **pfa_limits** – ndarray of two values that determine the limits of the pfa axis.
- **pmiss_limits** – ndarray of two values that determine the limits of the pmiss axis.
- **xticks** – coordonates of the ticks on the horizontal axis.
- **xticklabels** – labels of the ticks on the horizontal axis in a ndarray of strings.
- **yticks** – coordonates of the ticks on the vertical axis.
- **yticklabels** – labels of the ticks on the vertical axis in a ndarray of strings.

Scores

class bosaris.**Scores**(*scores_file_name=''*)
A class for storing scores for trials. The modelset and segset fields are lists of model and test segment names respectively. The element i,j of scoremat and scoremask corresponds to the trial involving model i and test segment j.

Attr modelset list of unique models in a ndarray

Attr segset list of unique test segments in a ndarray

Attr scoremask 2D ndarray of boolean which indicates the trials of interest i.e. the entry i,j in scoremat should be ignored if scoremask[i,j] is False

Attr scoremat 2D ndarray of scores

align_with_ndx(*ndx*)
The ordering in the output Scores object corresponds to ndx, so aligning several Scores objects with the same ndx will result in them being comparable with each other.

Parameters **ndx** – a Key or Ndx object

Returns resized version of the current Scores object to size of 'ndx' and reordered according to the ordering of modelset and segset in 'ndx'.

filter(*modlist*, *seglist*, *keep*)
Removes some of the information in a Scores object. Useful for creating a gender specific score set from a pooled gender score set. Depending on the value of 'keep', the two input lists indicate the models and test segments (and their associated scores) to retain or discard.

Parameters

- **modlist** – a list of strings which will be compared with the modelset of the current Scores object.
- **seglist** – a list of strings which will be compared with the segset of 'inscr'.
- **keep** – a boolean indicating whether modlist and seglist are the models to keep or discard.

Returns a filtered version of 'inscr'.

get_score(*modelID*, *segID*)
return a score given a model and segment identifiers raise an error if the trial does not exist :param modelID: id of the model :param segID: id of the test segment

get_tar_non(*key*)
Divides scores into target and non-target scores using information in a key.

Parameters **key** – a Key object.

Returns a vector of target scores. :return: a vector of non-target scores.

merge(*score_list*)
Merges a list of Scores objects into the current one. The resulting must have all models and segment in the input Scores (only once) and the union of all the scoremasks. It is an error if two of the input Scores objects have a score for the same trial.

Parameters **score_list** – the list of Scores object to merge

static **read**(*input_file_name*)
Read a Scores object from information in a hdf5 file.

Parameters **input_file_name** – name of the file to read from

static **read_matlab**(*input_file_name*)
Read a Scores object from information in a hdf5 file in Matlab BOSARIS format.

Parameters **input_file_name** – name of the file to read from

set_missing_to_value(*ndx*, *value*)
Sets all scores for which the trialmask is true but the scoremask is false to the same value, supplied by the user.

Parameters

- **ndx** – a Key or Ndx object.
- **value** – a value for the missing scores.

Returns a Scores object (with the missing scores added and set to value).

sort()
Sort models and segments

validate()
Checks that an object of type Scores obeys certain rules that must always be true.

return a boolean value indicating whether the object is valid.

1.4.4 The frontend package

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

frontend provides methods to process an audio signal in order to extract useful parameters for speaker verification.

The *frontend* packgae provides tools to extract, normalize and select
acoustic feature frames for speaker recognition. This package includes
4 modules, each dedicated to a different step of the process.

Features

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

frontend provides methods to process an audio signal in order to extract useful parameters for speaker verification.

frontend.features.**audspec**(*power_spectrum*, *fs=16000*, *nfilts=None*, *fbtype='bark'*, *minfreq=0*, *maxfreq=8000*, *sumpower=True*, *bwidth=1.0*)

Parameters

- **power_spectrum** –
- **fs** –
- **nfilts** –
- **fbtype** –
- **minfreq** –
- **maxfreq** –
- **sumpower** –
- **bwidth** –

Returns

frontend.features.**bark2hz**(*z*)

Converts frequencies Bark to Hertz (Hz)

Parameters **z** –

Returns

frontend.features.**compute_delta**(*features*, *win=3*, *method='filter'*, *filt=array([0.25, 0.5, 0.25, 0., -0.25, -0.5, -0.25])*)

features is a 2D-ndarray each row of features is a a frame

Parameters

- **features** – the feature frames to compute the delta coefficients
- **win** – parameter that set the length of the computation window. The size of the window is (win x 2) + 1
- **method** – method used to compute the delta coefficients can be diff or filter

- **filt** – definition of the filter to use in "filter" mode, default one is similar to SPRO4: filt=numpy.array([.2, .1, 0, -.1, -.2])

Returns the delta coefficients computed on the original features.

frontend.features.**dct_basis**(*nbasis*, *length*)

Parameters

- **nbasis** – number of CT coefficients to keep
- **length** – length of the matrix to process

Returns a basis of DCT coefficients

frontend.features.**dolpc**(*x*, *model_order=8*)

compute autoregressive model from spectral magnitude samples

Parameters

- **x** –
- **model_order** –

Returns

frontend.features.**fft2barkmx**(*n_fft*, *fs*, *nfilts=0*, *width=1.0*, *minfreq=0.0*, *maxfreq=8000*)

Generate a matrix of weights to combine FFT bins into Bark bins. n_fft defines the source FFT size at sampling rate fs. Optional nfilts specifies the number of output bands required (else one per bark), and width is the constant width of each band in Bark (default 1). While wts has n_fft columns, the second half are all zero. Hence, Bark spectrum is fft2barkmx(n_fft,fs) * abs(fft(xincols, n_fft)); 2004-09-05 dpwe@ee.columbia.edu based on rastamat/audspec.m

Parameters

- **n_fft** – the source FFT size at sampling rate fs
- **fs** – sampling rate
- **nfilts** – number of output bands required
- **width** – constant width of each band in Bark (default 1)
- **minfreq** –
- **maxfreq** –

Returns a matrix of weights to combine FFT bins into Bark bins

frontend.features.**fft2melmx**(*n_fft*, *fs=8000*, *nfilts=0*, *width=1.0*, *minfreq=0*, *maxfreq=4000*, *htkmel=False*, *constamp=False*)

Generate a matrix of weights to combine FFT bins into Mel bins. n_fft defines the source FFT size at sampling rate fs. Optional nfilts specifies the number of output bands required (else one per "mel/width"), and width is the constant width of each band relative to standard Mel (default 1). While wts has n_fft columns, the second half are all zero. Hence, Mel spectrum is fft2melmx(n_fft,fs)*abs(fft(xincols,n_fft)); minfreq is the frequency (in Hz) of the lowest band edge; default is 0, but 133.33 is a common standard (to skip LF). maxfreq is frequency in Hz of upper edge; default fs/2. You can exactly duplicate the mel matrix in Slaney's mfcc.m as fft2melmx(512, 8000, 40, 1, 133.33, 6855.5, 0); htkmel=1 means use HTK's version of the mel curve, not Slaney's. constamp=1 means make integration windows peak at 1, not sum to 1. frqs returns bin center frqs.

% 2004-09-05 dpwe@ee.columbia.edu based on fft2barkmx

Parameters

- **n_fft** –
- **fs** –

- **nfilts** –
- **width** –
- **minfreq** –
- **maxfreq** –
- **htkmel** –
- **constamp** –

Returns

frontend.features.**framing**(*sig*, *win_size*, *win_shift=1*, *context=(0, 0)*, *pad='zeros'*)

Parameters

- **sig** – input signal, can be mono or multi dimensional
- **win_size** – size of the window in term of samples
- **win_shift** – shift of the sliding window in terme of samples
- **context** – tuple of left and right context
- **pad** – can be zeros or edge

frontend.features.**hz2bark**(*f*)

Convert frequencies (Hertz) to Bark frequencies

Parameters **f** – the input frequency

Returns

frontend.features.**hz2mel**(*f*, *htk=True*)

Convert an array of frequency in Hz into mel.

Parameters **f** – frequency to convert

Returns the equivalence on the mel scale.

frontend.features.**levinson**(*r*, *order=None*, *allow_singularity=False*)

Levinson-Durbin recursion.

Find the coefficients of a length(r)-1 order autoregressive linear process

Parameters

- **r** – autocorrelation sequence of length N + 1 (first element being the zero-lag autocorrelation)
- **order** – requested order of the autoregressive coefficients. default is N.
- **allow_singularity** – false by default. Other implementations may be True (e.g., octave)

Returns

- the *N+1* autoregressive coefficients $A = (1, a_1 ... a_N)$
- the prediction errors
- the *N* reflections coefficients values

This algorithm solves the set of complex linear simultaneous equations using Levinson algorithm.

$$\mathbf{T}_M \begin{pmatrix} 1 \\ \mathbf{a}_M \end{pmatrix} = \begin{pmatrix} \rho_M \\ \mathbf{0}_M \end{pmatrix}$$

where $\mathbf{T}_M$ is a Hermitian Toeplitz matrix with elements $T_0, T_1, \ldots, T_M$.

Note: Solving this equations by Gaussian elimination would require M^3 operations whereas the levinson algorithm requires $M^2 + M$ additions and $M^2 + M$ multiplications.

This is equivalent to solve the following symmetric Toeplitz system of linear equations

$$\begin{pmatrix} r_1 & r_2^* & \ldots & r_n^* \\ r_2 & r_1^* & \ldots & r_{n-1}^* \\ \ldots & \ldots & \ldots & \ldots \\ r_n & \ldots & r_2 & r_1 \end{pmatrix} \begin{pmatrix} a_2 \\ a_3 \\ \ldots \\ a_{N+1} \end{pmatrix} = \begin{pmatrix} -r_2 \\ -r_3 \\ \ldots \\ -r_{N+1} \end{pmatrix}$$

where $r = (r_1 ... r_{N+1})$ is the input autocorrelation vector, and r_i^* denotes the complex conjugate of r_i. The input r is typically a vector of autocorrelation coefficients where lag 0 is the first element r_1.

```
>>> import numpy; from spectrum import LEVINSON
>>> T = numpy.array([3., -2+0.5j, .7-1j])
>>> a, e, k = LEVINSON(T)
```

frontend.features.**lifter**(*x*, *lift=0.6*, *invs=False*)

Apply lifter to matrix of cepstra (one per column) lift = exponent of x i^n liftering or, as a negative integer, the length of HTK-style sin-curve liftering. If inverse == 1 (default 0), undo the liftering.

Parameters

- **x** –
- **lift** –
- **invs** –

Returns

frontend.features.**lpc2cep**(*a*, *nout*)

Convert the LPC 'a' coefficients in each column of lpcas into frames of cepstra. nout is number of cepstra to produce, defaults to size(lpcas,1) 2003-04-11 dpwe@ee.columbia.edu

Parameters

- **a** –
- **nout** –

Returns

frontend.features.**lpc2spec**(*lpcas*, *nout=17*)

Convert LPC coeffs back into spectra nout is number of freq channels, default 17 (i.e. for 8 kHz)

Parameters

- **lpcas** –
- **nout** –

Returns

frontend.features.**mel2hz**(*z*, *htk=True*)

Convert an array of mel values in Hz.

Parameters **m** – ndarray of frequencies to convert in Hz.

Returns the equivalent values in Hertz.

frontend.features.**mel_filter_bank**(*fs*, *nfft*, *lowfreq*, *maxfreq*, *widest_nlogfilt*, *widest_lowfreq*, *widest_maxfreq*)

Compute triangular filterbank for cepstral coefficient computation.

Parameters

- **fs** – sampling frequency of the original signal.
- **nfft** – number of points for the Fourier Transform
- **lowfreq** – lower limit of the frequency band filtered
- **maxfreq** – higher limit of the frequency band filtered
- **widest_nlogfilt** – number of log filters
- **widest_lowfreq** – lower frequency of the filter bank
- **widest_maxfreq** – higher frequency of the filter bank
- **widest_maxfreq** – higher frequency of the filter bank

Returns the filter bank and the central frequencies of each filter

frontend.features.**mfcc**(*input_sig*, *lowfreq=100*, *maxfreq=8000*, *nlinfilt=0*, *nlogfilt=24*, *nwin=0.025*, *fs=16000*, *nceps=13*, *shift=0.01*, *get_spec=False*, *get_mspec=False*, *prefac=0.97*)

Compute Mel Frequency Cepstral Coefficients.

Parameters

- **input_sig** – input signal from which the coefficients are computed. Input audio is supposed to be RAW PCM 16bits
- **lowfreq** – lower limit of the frequency band filtered. Default is 100Hz.
- **maxfreq** – higher limit of the frequency band filtered. Default is 8000Hz.
- **nlinfilt** – number of linear filters to use in low frequencies. Default is 0.
- **nlogfilt** – number of log-linear filters to use in high frequencies. Default is 24.
- **nwin** – length of the sliding window in seconds Default is 0.025.
- **fs** – sampling frequency of the original signal. Default is 16000Hz.
- **nceps** – number of cepstral coefficients to extract. Default is 13.
- **shift** – shift between two analyses. Default is 0.01 (10ms).
- **get_spec** – boolean, if true returns the spectrogram
- **get_mspec** – boolean, if true returns the output of the filter banks
- **prefac** – pre-emphasis filter value

Returns the cepstral coefficients in a ndaray as well as the Log-spectrum in the mel-domain in a ndarray.

Note: MFCC are computed as follows:

•Pre-processing in time-domain (pre-emphasizing)

•Compute the spectrum amplitude by windowing with a Hamming window

•**Filter the signal in the spectral domain with a triangular filter-bank, whose filters are approximatively** linearly spaced on the mel scale, and have equal bandwith in the mel scale

•Compute the DCT of the log-spectrom

•Log-energy is returned as first coefficient of the feature vector.

For more details, refer to *[Davis80]*.

frontend.features.**pca_dct**(*cep*, *left_ctx=12*, *right_ctx=12*, *p=None*)
Apply DCT PCA as in [McLaren 2015] paper: Mitchell McLaren and Yun Lei, 'Improved Speaker Recognition Using DCT coefficients as features' in ICASSP, 2015

A 1D-dct is applied to the cepstral coefficients on a temporal sliding window. The resulting matrix is then flatten and reduced by using a Principal Component Analysis.

Parameters

- **cep** – a matrix of cepstral cefficients, 1 line per feature vector
- **left_ctx** – number of frames to consider for left context
- **right_ctx** – number of frames to consider for right context
- **p** – a PCA matrix trained on a developpment set to reduce the dimension of the features. P is a portait matrix

frontend.features.**plp**(*input_sig*, *nwin=0.025*, *fs=16000*, *plp_order=13*, *shift=0.01*, *get_spec=False*, *get_mspec=False*, *prefac=0.97*, *rasta=True*)
output is matrix of features, row = feature, col = frame

% fs is sampling rate of samples, defaults to 8000 % dorasta defaults to 1; if 0, just calculate PLP % modelorder is order of PLP model, defaults to 8. 0 -> no PLP

Parameters

- **input_sig** –
- **fs** – sampling rate of samples default is 8000
- **rasta** – default is True, if False, juste compute PLP
- **model_order** – order of the PLP model, default is 8, 0 means no PLP

Returns matrix of features, row = features, column are frames

frontend.features.**postaud**(*x*, *fmax*, *fbtype='bark'*, *broaden=0*)
do loudness equalization and cube root compression

Parameters

- **x** –
- **fmax** –
- **fbtype** –
- **broaden** –

Returns

frontend.features.**power_spectrum**(*input_sig*, *fs=8000*, *win_time=0.025*, *shift=0.01*, *prefac=0.97*)
Compute the power spectrum of the signal. :param input_sig: :param fs: :param win_time: :param shift: :param prefac: :return:

frontend.features.**shifted_delta_cepstral**(*cep*, *d=1*, *p=3*, *k=7*)
Compute the Shifted-Delta-Cepstral features for language identification

Parameters

- **cep** – matrix of feature, 1 vector per line
- **d** – represents the time advance and delay for the delta computation
- **k** – number of delta-cepstral blocks whose delta-cepstral coefficients are stacked to form the final feature vector
- **p** – time shift between consecutive blocks.

return: cepstral coefficient concatenated with shifted deltas

frontend.features.**spec2cep**(*spec*, *ncep=13*, *type=2*)

Calculate cepstra from spectral samples (in columns of spec) Return ncep cepstral rows (defaults to 9) This one does type II dct, or type I if type is specified as 1 dctm returns the DCT matrix that spec was multiplied by to give cep.

Parameters

- **spec** –
- **ncep** –
- **type** –

Returns

frontend.features.**trfbank**(*fs*, *nfft*, *lowfreq*, *maxfreq*, *nlinfilt*, *nlogfilt*, *midfreq=1000*)

Compute triangular filterbank for cepstral coefficient computation.

Parameters

- **fs** – sampling frequency of the original signal.
- **nfft** – number of points for the Fourier Transform
- **lowfreq** – lower limit of the frequency band filtered
- **maxfreq** – higher limit of the frequency band filtered
- **nlinfilt** – number of linear filters to use in low frequencies
- **nlogfilt** – number of log-linear filters to use in high frequencies
- **midfreq** – frequency boundary between linear and log-linear filters

Returns the filter bank and the central frequencies of each filter

io

Copyright 2014-2017 Anthony Larcher

frontend provides methods to process an audio signal in order to extract useful parameters for speaker verification.

frontend.io.**pcmu2lin**(*p*, *s=4004.189931*)

Convert Mu-law PCM to linear X=(P,S) lin = pcmu2lin(pcmu) where pcmu contains a vector of mu-law values in the range 0 to 255. No checking is performed to see that numbers are in this range.

Output values are divided by the scale factor s:

s Output Range 1 +-8031 (integer values) 4004.2 +-2.005649 (default) 8031 +-1 8159 +-0.9843118 (+-1 nominal full scale)

The default scaling factor 4004.189931 is equal to sqrt((2207^2 + 5215^2)/2) this follows ITU standard G.711. The sine wave with PCM-Mu values [158 139 139 158 30 11 11 30] has a mean square value of unity corresponding to 0 dBm0. :param p: input signal encoded in PCM mu-law to convert :param s: conversion value from mu-scale oto linear scale

frontend.io.**read_audio**(*input_file_name*, *framerate=None*)

Read a 1 or 2-channel audio file in SPHERE, WAVE or RAW PCM format. The format is determined from the file extension. If the sample rate read from the file is a multiple of the one given as parameter, we apply a decimation function to subsample the signal.

Parameters

- **input_file_name** – name of the file to read from
- **framerate** – frame rate, optional, if lower than the one read from the file, subsampling is applied

Returns the signal as a numpy array and the sampling frequency

frontend.io.**read_feature_segment**(*input_file_name*, *feature_id=None*, *feature_mask=None*, *file_format='hdf5'*, *start=0*, *stop=None*)

Parameters

- **input_file_name** –
- **feature_id** –
- **feature_mask** –
- **file_format** –
- **start** –
- **stop** –

Returns

frontend.io.**read_hdf5**(*h5f*, *show*, *dataset_list=('cep', 'fb', 'energy', 'vad', 'bnf')*)

Parameters

- **h5f** – HDF5 file handler to read from
- **show** – identifier of the show to read
- **dataset_list** – list of datasets to read and concatenate

Returns

frontend.io.**read_hdf5_segment**(*file_name*, *dataset*, *mask*, *start*, *end*)

Read a segment from a stream in HDF5 format. Return the features in the range start:end In case the start and end cannot be reached, the first or last feature are copied so that the length of the returned segment is always end-start

Parameters

- **file_name** – name of the file to open
- **dataset** – identifier of the dataset in the HDF5 file
- **mask** –
- **start** –
- **end** –

:return:read_hdf5_segment

frontend.io.**read_htk**(*input_file_name*, *label_file_name=''*, *selected_label=''*, *frame_per_second=100*)

Read a sequence of features in HTK format

Parameters

- **input_file_name** – name of the file to read from
- **label_file_name** – name of the label file to read from
- **selected_label** – label to select
- **frame_per_second** – number of frames per second

Returns a tupple (d, fp, dt, tc, t) described below

Note:

•d = data: column vector for waveforms, 1 row per frame for other types

•fp = frame period in seconds

•dt = data type (also includes Voicebox code for generating data)

0.WAVEFORM Acoustic waveform

1.LPC Linear prediction coefficients

2.LPREFC LPC Reflection coefficients: -lpcar2rf([1 LPC]);LPREFC(1)=[];

3.LPCEPSTRA LPC Cepstral coefficients

4.LPDELCEP LPC cepstral+delta coefficients (obsolete)

5.IREFC LPC Reflection coefficients (16 bit fixed point)

6.MFCC Mel frequency cepstral coefficients

7.FBANK Log Fliter bank energies

8.MELSPEC linear Mel-scaled spectrum

9.USER User defined features

10.DISCRETE Vector quantised codebook

11.PLP Perceptual Linear prediction

12.ANON

•tc = full type code = dt plus (optionally)

one or more of the following modifiers

- 64 _E Includes energy terms
- 128 _N Suppress absolute energy
- 256 _D Include delta coefs
- 512 _A Include acceleration coefs
- 1024 _C Compressed
- 2048 _Z Zero mean static coefs
- 4096 _K CRC checksum (not implemented yet)
- 8192 _0 Include 0'th cepstral coef

- 16384 _V Attach VQ index
- 32768 _T Attach delta-delta-delta index

•t = text version of type code e.g. LPC_C_K

This function is a translation of the Matlab code from VOICEBOX is a MATLAB toolbox for speech processing. by Mike Brookes Home page: *VOICEBOX <http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>*

`frontend.io.`**`read_htk_segment`**(*input_file_name*, *start=0*, *stop=None*)

Read a segment from a stream in SPRO4 format. Return the features in the range start:end In case the start and end cannot be reached, the first or last feature are copied so that the length of the returned segment is always end-start

Parameters

- **`input_file_name`** – name of the feature file to read from or file-like object alowing to seek in the file
- **`start`** – index of the first frame to read (start at zero)
- **`stop`** – index of the last frame following the segment to read. end < 0 means that end is the value of the right_context to add at the end of the file

Returns a sequence of features in a ndarray of length end-start

`frontend.io.`**`read_label`**(*input_file_name*, *selected_label='speech'*, *frame_per_second=100*)

Read label file in ALIZE format

Parameters

- **`input_file_name`** – the label file name
- **`selected_label`** – the label to return. Default is 'speech'.
- **`frame_per_second`** – number of frame per seconds. Used to convert the frame number into time. Default is 100.

Returns a logical array

`frontend.io.`**`read_pcm`**(*input_file_name*)

Read signal from single channel PCM 16 bits

Parameters **`input_file_name`** – name of the PCM file to read.

Returns the audio signal read from the file in a ndarray encoded on 16 bits, None and 2 (depth of the encoding in bytes)

`frontend.io.`**`read_sph`**(*input_file_name*, *mode='p'*)

Read a SPHERE audio file

Parameters

- **`input_file_name`** – name of the file to read
- **`mode`** – specifies the following (* =default)

Note:

•Scaling:

–'s' Auto scale to make data peak = +-1 (use with caution if reading in chunks)

–'r' Raw unscaled data (integer values)

– 'p' Scaled to make +-1 equal full scale

– **'o' Scale to bin centre rather than bin edge (e.g. 127 rather than 127.5 for 8 bit values,** can be combined with n+p,r,s modes)

– **'n' Scale to negative peak rather than positive peak (e.g. 128.5 rather than 127.5 for 8 bit values,** can be combined with o+p,r,s modes)

•Format

– 'l' Little endian data (Intel,DEC) (overrides indication in file)

– 'b' Big endian data (non Intel/DEC) (overrides indication in file)

•File I/O

– 'f' Do not close file on exit

– 'd' Look in data directory: voicebox('dir_data')

– 'w' Also read the annotation file *.wrd if present (as in TIMIT)

– 't' Also read the phonetic transcription file *.phn if present (as in TIMIT)

•NMAX maximum number of samples to read (or -1 for unlimited [default])

•**NSKIP number of samples to skip from start of file (or -1 to continue from previous read when FFX** is given instead of FILENAME [default])

Returns a tupple such that (Y, FS)

Note:

•Y data matrix of dimension (samples,channels)

•FS sample frequency in Hz

•**WRD{*,2} cell array with word annotations: WRD{*,:)={[t_start t_end],'text'} where times are in seconds** only present if 'w' option is given

•**PHN{*,2} cell array with phoneme annotations: PHN{*,:)={[t_start t_end],'phoneme'} where times** are in seconds only present if 't' option is present

•FFX Cell array containing

1.filename

2.header information

1.first header field name

2.first header field value

3.format string (e.g. NIST_1A)

4.(a)file id

(b)current position in file

(c)dataoff byte offset in file to start of data

(d)order byte order (l or b)

(e)nsamp number of samples

(f)number of channels

(g)nbytes bytes per data value

(h)bits number of bits of precision

(i)fs sample frequency

(j)min value

(k)max value

(l)coding 0=PCM,1=uLAW + 0=no compression, 0=shorten,20=wavpack,30=shortpack

(m)file not yet decompressed

5.temporary filename

If no output parameters are specified, header information will be printed. The code to decode shorten-encoded files, is not yet released with this toolkit.

frontend.io.**read_spro4**(*input_file_name*, *label_file_name=''*, *selected_label=''*, *frame_per_second=100*)

Read a feature stream in SPRO4 format

Parameters

- **input_file_name** – name of the feature file to read from
- **label_file_name** – name of the label file to read if required. By Default, the method assumes no label to read from.
- **selected_label** – label to select in the label file. Default is none.
- **frame_per_second** – number of frame per seconds. Used to convert the frame number into time. Default is 0.

Returns a sequence of features in a numpy array

frontend.io.**read_spro4_segment**(*input_file_name*, *start=0*, *end=None*)

Read a segment from a stream in SPRO4 format. Return the features in the range start:end In case the start and end cannot be reached, the first or last feature are copied so that the length of the returned segment is always end-start

Parameters

- **input_file_name** – name of the feature file to read from
- **start** – index of the first frame to read (start at zero)
- **end** – index of the last frame following the segment to read. end < 0 means that end is the value of the right_context to add at the end of the file

Returns a sequence of features in a ndarray of length end-start

frontend.io.**read_wav**(*input_file_name*)

Parameters **input_file_name** –

Returns

frontend.io.**write_hdf5**(*show*, *fh*, *cep*, *cep_mean*, *cep_std*, *energy*, *energy_mean*, *energy_std*, *fb*, *fb_mean*, *fb_std*, *bnf*, *bnf_mean*, *bnf_std*, *label*)

Parameters

- **show** – identifier of the show to write
- **fh** – HDF5 file handler
- **cep** – cepstral coefficients to store
- **cep_mean** – pre-computed mean of the cepstral coefficient
- **cep_std** – pre-computed standard deviation of the cepstral coefficient
- **energy** – energy coefficients to store
- **energy_mean** – pre-computed mean of the energy
- **energy_std** – pre-computed standard deviation of the energy
- **fb** – filter-banks coefficients to store
- **fb_mean** – pre-computed mean of the filter bank coefficient
- **fb_std** – pre-computed standard deviation of the filter bank coefficient
- **bnf** – bottle-neck features to store
- **bnf_mean** – pre-computed mean of the bottleneck features
- **bnf_std** – pre-computed standard deviation of the bottleneck features
- **label** – vad labels to store

Returns

Normfeat

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

frontend provides methods to process an audio signal in order to extract useful parameters for speaker verification.

frontend.normfeat.**cep_sliding_norm**(*features*, *win=301*, *label=None*, *center=True*, *reduce=False*)

Performs a cepstal mean substitution and standard deviation normalization in a sliding windows. MFCC is modified.

Parameters

- **features** – the MFCC, a numpy array
- **win** – the size of the sliding windows
- **label** – vad label if available
- **center** – performs mean subtraction
- **reduce** – performs standard deviation division

frontend.normfeat.**cms**(*features*, *label=None*, *global_mean=None*)

Performs cepstral mean subtraction

Parameters

- **features** – a feature stream of dimension dim x nframes where dim is the dimension of the acoustic features and nframes the number of frames in the stream
- **label** – a logical vector
- **global_mean** – pre-computed mean to use for feature normalization if given

Returns a feature stream

frontend.normfeat.**cmvn**(*features*, *label=None*, *global_mean=None*, *global_std=None*)
Performs mean and variance normalization

Parameters

- **features** – a feature stream of dimension dim x nframes where dim is the dimension of the acoustic features and nframes the number of frames in the stream
- **global_mean** – pre-computed mean to use for feature normalization if given
- **global_std** – pre-computed standard deviation to use for feature normalization if given
- **label** – a logical verctor

Returns a sequence of features

frontend.normfeat.**rasta_filt**(*x*)
Apply RASTA filtering to the input signal.

Parameters **x** – the input audio signal to filter. cols of x = critical bands, rows of x = frame same for y but after filtering default filter is single pole at 0.94

frontend.normfeat.**stg**(*features*, *label=None*, *win=301*)
Performs feature warping on a sliding window

Parameters

- **features** – a feature stream of dimension dim x nframes where dim is the dimension of the acoustic features and nframes the number of frames in the stream
- **label** – label of selected frames to compute the Short Term Gaussianization, by default, al frames are used
- **win** – size of the frame window to consider, must be an odd number to get a symetric context on left and right

Returns a sequence of features

vad

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

frontend provides methods to process an audio signal in order to extract useful parameters for speaker verification.

frontend.vad.**label_fusion**(*label*, *win=3*)
Apply a morphological filtering on the label to remove isolated labels. In case the input is a two channel label (2D ndarray of boolean of same length) the labels of two channels are fused to remove overlaping segments of speech.

Parameters

- **label** – input labels given in a 1D or 2D ndarray
- **win** – parameter or the morphological filters

frontend.vad.**pre_emphasis**(*input_sig*, *pre*)
Pre-emphasis of an audio signal. :param input_sig: the input vector of signal to pre emphasize :param pre: value that defines the pre-emphasis filter.

frontend.vad.**segment_axis**(*a*, *length*, *overlap=0*, *axis=None*, *end='cut'*, *endvalue=0*)

Generate a new array that chops the given array along the given axis into overlapping frames.

This method has been implemented by Anne Archibald, as part of the talk box toolkit example:

```
segment_axis(arange(10), 4, 2)
array([[0, 1, 2, 3],
   ( [2, 3, 4, 5],
     [4, 5, 6, 7],
     [6, 7, 8, 9]])
```

Parameters

- **a** – the array to segment
- **length** – the length of each frame
- **overlap** – the number of array elements by which the frames should overlap
- **axis** – the axis to operate on; if None, act on the flattened array
- **end** – what to do with the last frame, if the array is not evenly divisible into pieces. Options are: - 'cut' Simply discard the extra values - 'wrap' Copy values from the beginning of the array - 'pad' Pad with a constant value
- **endvalue** – the value to use for end='pad'

Returns a ndarray

The array is not copied unless necessary (either because it is unevenly strided and being flattened or because end is set to 'pad' or 'wrap').

frontend.vad.**speech_enhancement**(*X*, *Gain*, *NN*)

This program is only to process the single file seperated by the silence section if the silence section is detected, then a counter to number of buffer is set and pre-processing is required.

Usage: SpeechENhance(wavefilename, Gain, Noise_floor)

Parameters

- **X** – input audio signal
- **Gain** – default value is 0.9, suggestion range 0.6 to 1.4, higher value means more subtraction or noise redcution
- **NN** –

Returns a 1-dimensional array of boolean that is True for high energy frames.

Copyright 2014 Sun Han Wu and Anthony Larcher

frontend.vad.**vad_percentil**(*log_energy*, *percent*)

Parameters

- **log_energy** –
- **percent** –

Returns

frontend.vad.**vad_snr**(*sig*, *snr*, *fs=16000*, *shift=0.01*, *nwin=256*)

Select high energy frames based on the Signal to Noise Ratio of the signal. Input signal is expected encoded on 16 bits

Parameters

- **sig** – the input audio signal
- **snr** – Signal to noise ratio to consider
- **fs** – sampling frequency of the input signal in Hz. Default is 16000.
- **shift** – shift between two frames in seconds. Default is 0.01
- **nwin** – number of samples of the sliding window. Default is 256.

1.4.5 The libsvm package

LIBSVM

is an integrated software
for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR)
and distribution estimation (one-class SVM)

SIDEKIT only makes use of the library and Python wrapper provided in **LIBSVM**
if a fully functional version of the **LIBSVM** library is available in the `sidekit/libsvm/` directory
The `libsvm` package released with **SIDEKIT** provides high level interfaces
to use Support Vector Machines for speaker recognition.
For more details about **LIBSVM** you can refer to the original website

Warning: **SIDEKIT** requires a version of the libsvm library that is compatible with your machine. Before running **SIDEKIT**, download, and compile the libsvm library to make sure you have the corresponding file (libsvm.dll for windows and libsvm.so.2 for UNIX-like OS) in the `sidekit/libsvm/` directory.

LIBSVM Core library

svm

Copyright (c) 2000-2014 Chih-Chung Chang and Chih-Jen Lin All rights reserved.

`libsvm.svm.`**`toPyModel`**(*model_ptr*) → svm_model
Convert a ctypes POINTER(svm_model) to a Python svm_model

svmutil

Copyright (c) 2000-2014 Chih-Chung Chang and Chih-Jen Lin All rights reserved.

`libsvm.svmutil.`**`evaluations`**(*ty*, *pv*) -> (*ACC*, *MSE*, *SCC*)
Calculate accuracy, mean squared error and squared correlation coefficient using the true values (ty) and predicted values (pv).

`libsvm.svmutil.`**`read_svm`**(*svm_file_name*)
Read SVM model in PICKLE format

Parameters **svm_file_name** – name of the file to read from

libsvm.svmutil.**save_svm**(*svm_file_name*, *w*, *b*)
Save SVM weights and biais in PICKLE format :return:

libsvm.svmutil.**svm_load_model**(*model_file_name*) → model
Load a LIBSVM model from model_file_name and return. :param model_file_name: file name to load from

libsvm.svmutil.**svm_predict**(*y, x, m [, options]) -> (p_labels, p_acc, p_vals*)
Predict data (y, x) with the SVM model m. options:

- **"-b" probability_estimates: whether to predict probability estimates,** 0 or 1 (default 0); for one-class SVM only 0 is supported.
- "-q" : quiet mode (no outputs).

The return tuple contains

- p_labels: a list of predicted labels
- **p_acc: a tuple including accuracy (for classification),** mean-squared error, and squared correlation coefficient (for regression).
- **p_vals: a list of decision values or probability estimates** (if '-b 1' is specified). If k is the number of classes, for decision values, each element includes results of predicting k(k-1)/2 binary-class SVMs. For probabilities, each element contains k values indicating the probability that the testing instance is in each class.

Note: that the order of classes here is the same as 'model.label' field in the model structure.

---

libsvm.svmutil.**svm_read_problem**(*data_file_name*) → [y, x]
Read LIBSVM-format data from data_file_name and return labels y and data instances x. :param data_file_name: name of the file to load from

libsvm.svmutil.**svm_save_model**(*model_file_name*, *model*) → None
Save a LIBSVM model to the file model_file_name. :param model_file_name: file name to write to :param model: model to save

libsvm.svmutil.**svm_train**(*y*, *x*[, *options*]) → model | ACC | MSE
svm_train(prob [, options]) -> model | ACC | MSE svm_train(prob, param) -> model | ACC| MSE

Train an SVM model from data (y, x) or an svm_problem prob using 'options' or an svm_parameter param. If '-v' is specified in 'options' (i.e., cross validation) either accuracy (ACC) or mean-squared error (MSE) is returned. options:

- -s svm_type : set type of SVM (default 0)
  - –0 – C-SVC (multi-class classification)
  - –1 – nu-SVC (multi-class classification)
  - –2 – one-class SVM
  - –3 – epsilon-SVR (regression)
  - –4 – nu-SVR (regression)
- -t kernel_type : set type of kernel function (default 2)
  - –0 – linear: u'*v
  - –1 – polynomial: (gamma*u'*v + coef0)^degree
  - –2 – radial basis function: exp(-gamma*|u-v|^2)

–3 – sigmoid: tanh(gamma*u'*v + coef0)

–4 – precomputed kernel (kernel values in training_set_file)

- -d degree : set degree in kernel function (default 3)
- -g gamma : set gamma in kernel function (default 1/num_features)
- -r coef0 : set coef0 in kernel function (default 0)
- -c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
- -n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
- -p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
- -m cachesize : set cache memory size in MB (default 100)
- -e epsilon : set tolerance of termination criterion (default 0.001)
- -h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
- -b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
- -wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
- -v n: n-fold cross validation mode
- -q : quiet mode (no outputs)

1.4.6 The nnet package

Copyright 2014-2017 Anthony Larcher and Sylvain Meignier

nnet provides methods to manage Neural Networks using Theano

The *nnet* package provides tools to train and use Neural Networks using Theano.

feed_forward

Copyright 2014-2017 Anthony Larcher

theano_utils provides utilities to facilitate the work with SIDEKIT and THEANO.

The authors would like to thank the BUT Speech@FIT group (http://speech.fit.vutbr.cz) and Lukas BURGET for sharing the source code that strongly inspired this module. Thank you for your valuable contribution.

class nnet.feed_forward.**FForwardNetwork**(*filename=None*, *input_size=0*, *input_mean=array([]*, *dtype=float64)*, *input_std=array([]*, *dtype=float64)*, *hidden_layer_sizes=()*, *layers_activations=()*, *n_classes=0*)

Class FForwardNetwork that implement a feed-forward neural network for multiple purposes

compute_ubm_dnn(*training_list*, *dnn_features_server*, *features_server*, *viterbi=False*)

Parameters

- **training_list** – list of files to process to train the model
- **dnn_features_server** – FeaturesServer to feed the network

- **features_server** – FeaturesServer providing features to compute the first and second order statistics
- **viterbi** – boolean, if True, keep only one coefficient to one and the others at zeros

Returns a Mixture object

feed_forward(*data*, *layer_number*)

Function used to extract bottleneck features or embeddings from an existing Neural Network. The first bottom layers of the neural network are loaded and all feature files are process through the network to get the output and save them as feature files. If specified, the output features can be normalized (cms, cmvn, stg) given input labels

Parameters

- **data** – data to process (a Numpy array)
- **layer_number** – number of layers to load from the model

Returns the transform data

feed_forward_acoustic(*feature_file_list*, *features_server*, *layer_number*, *output_file_structure*)

Function used to extract bottleneck features or embeddings from an existing Neural Network. The first bottom layers of the neural network are loaded and all feature files are process through the network to get the output and save them as feature files. If specified, the output features can be normalized (cms, cmvn, stg) given input labels

Parameters

- **feature_file_list** – list of feature files to process through the feed formward network
- **features_server** – FeaturesServer used to load the data
- **layer_number** – number of layers to load from the model
- **output_file_structure** – structure of the output file name

Returns

instantiate_network()

Create Theano variables and initialize the weights and biases of the neural network Create the different funtions required to train the NN

instantiate_partial_network(*layer_number*)

Instantiate a neural network with only the bottom layers of the network. After instantiating, the function display the structure of the network in the root logger if it exists :param layer_number: number of layers to load from

static read(*input_filename*)

Parameters **input_filename** –

Returns

replace_layer(*layer_number*, *hidden_unit_number*, *activation_function=None*)

Parameters

- **layer_number** –
- **hidden_unit_number** –
- **activation_function** –

Returns

train(*training_set*, *cross_validation_set*, *lr=0.008*, *batch_size=512*, *max_iters=20*, *tolerance=0.003*, *output_file_name=''*, *save_tmp_nnet=False*, *num_thread=1*)

Parameters

- **training_set** – list of segments to use for training It is a list of 4 dimensional tuples which first argument is the absolute file name second argument is the index of the first frame of the segment third argument is the index of the last frame of the segment and fourth argument is a numpy array of integer, labels corresponding to each frame of the segment
- **cross_validation_set** – is a list of segments to use for cross validation. Same format as train_seg_list
- **lr** – initial learning rate
- **batch_size** – size of the minibatches as number of frames
- **max_iters** – macimum number of epochs
- **tolerance** –
- **output_file_name** – root name of the files to save Neural Network parameters
- **save_tmp_nnet** – boolean, if True, save the parameters after each epoch
- **num_thread** – number of parallel process to run (for CPU part of the code)

Returns

train_acoustic(*training_seg_list*, *cross_validation_seg_list*, *features_server*, *feature_size*, *lr=0.008*, *segment_buffer_size=200*, *batch_size=512*, *max_iters=20*, *tolerance=0.003*, *output_file_name=''*, *save_tmp_nnet=False*, *traps=False*, *num_thread=1*)

Parameters

- **training_seg_list** – list of segments to use for training It is a list of 4 dimensional tuples which first argument is the absolute file name second argument is the index of the first frame of the segment third argument is the index of the last frame of the segment and fourth argument is a numpy array of integer, labels corresponding to each frame of the segment
- **cross_validation_seg_list** – is a list of segments to use for cross validation. Same format as train_seg_list
- **features_server** – FeaturesServer used to load data
- **feature_size** – dimension of the acoustic feature
- **lr** – initial learning rate
- **segment_buffer_size** – number of segments loaded at once
- **batch_size** – size of the minibatches as number of frames
- **max_iters** – macimum number of epochs
- **tolerance** –
- **output_file_name** – root name of the files to save Neural Betwork parameters
- **save_tmp_nnet** – boolean, if True, save the parameters after each epoch
- **traps** – boolean, if True, compute TRAPS on the input data, if False jsut use concatenated frames

- **num_thread** – number of parallel process to run (for CPU part of the code)

Returns

train_per_layer(*layer_training_sequence*, *training_accuracy_limit*, *training_seg_list*, *cross_validation_seg_list*, *features_server*, *feature_size*, *lr=0.008*, *segment_buffer_size=200*, *batch_size=512*, *max_iters=20*, *tolerance=0.003*, *output_file_name=''*, *save_tmp_nnet=False*, *traps=False*, *num_thread=1*)

Parameters

- **layer_training_sequence** –
- **training_accuracy_limit** –
- **training_seg_list** –
- **cross_validation_seg_list** –
- **features_server** –
- **feature_size** –
- **lr** –
- **segment_buffer_size** –
- **batch_size** –
- **max_iters** –
- **tolerance** –
- **output_file_name** –
- **save_tmp_nnet** –
- **traps** –
- **num_thread** –

Returns

nnet.feed_forward.**compute_stat_dnn**(*idmap*, *feed_forward_nnet*, *dnn_features_server*, *features_server*, *num_thread=1*)

Parameters

- **idmap** – IdMap that describes segment to process
- **feed_forward_nnet** – neural network to load
- **dnn_features_server** – FeaturesServer to feed the Neural Network
- **features_server** – FeaturesServer that provide additional features to compute first order statistics
- **num_thread** – number of parallel process to run

Returns

nnet.feed_forward.**compute_stat_dnn_parallel**(*feed_forward_nnet*, *segset*, *stat0*, *stat1*, *dnn_features_server*, *features_server*, *seg_indices=None*)

Single thread version of the statistic computation using a DNN.

Parameters

- **feed_forward_nnet** – weights and biaises of the network stored in npz format

- **segset** – list of segments to process
- **stat0** – local matrix of zero-order statistics
- **stat1** – local matrix of first-order statistics
- **dnn_features_server** – FeaturesServer that provides input data for the DNN
- **features_server** – FeaturesServer that provide additional features to compute first order statistics
- **seg_indices** – indices of the

Returns a StatServer with all computed statistics

nnet.feed_forward.**export_params**(*params*, *param_dict*)
Export network parameters into Numpy format

Parameters

- **params** – dictionary of variables in Theano format
- **param_dict** – dictionary of variables in Numpy format

nnet.feed_forward.**get_params**(*params*)
Return parameters of into a Python dictionary format

Parameters **params** – a list of Theano shared variables

Returns the same variables in Numpy format in a dictionary

nnet.feed_forward.**kaldi_to_hdf5**(*input_file_name*, *output_file_name*)
Convert a text file containing frame alinment from Kaldi into an HDF5 file with the following structure:

show/start/labels

Parameters

- **input_file_name** –
- **output_file_name** –

Returns

nnet.feed_forward.**mean_std_many**(*features_server*, *feature_size*, *seg_list*, *traps=False*, *num_thread=1*)
Compute the mean and standard deviation from a list of segments.

Parameters

- **features_server** – FeaturesServer used to load data
- **feature_size** – dimension o the features to accumulate
- **seg_list** – list of file names with start and stop indices
- **traps** – apply traps processing on the features in context
- **traps** – apply traps processing on the features in context
- **num_thread** – number of parallel processing to run

Returns a tuple of three values, the number of frames, the mean and the standard deviation

nnet.feed_forward.**segment_mean_std_hdf5**(*input_segment*)
Compute the sum and square sum of all features for a list of segments. Input files are in HDF5 format

Parameters **input_segment** – list of segments to read from, each element of the list is a tuple of 5 values, the filename, the index of thefirst frame, index of the last frame, the number of frames for the left context and the number of frames for the right context

Returns a tuple of three values, the number of frames, the sum of frames and the sum of squares

nnet.feed_forward.**set_params**(*params*, *param_dict*)
Set the parameters in a list of Theano variables from a dictionary

Parameters

- **params** – dictionary to read from
- **param_dict** – list of variables in Theano format

1.5 Tutorials

See now how to start with **SIDEKIT** with some basic tutorials and advanced evaluations on standard databases.

1.5.1 Enter the SIDEKIT

How to manage the data: IdMap, Ndx, Key, Scores and StatServer

IdMap

Description

IdMap are used to store two lists of strings and to map between them. Most of the time, IdMap are used to associate names of segments (also referred to as sessions or shows) ‘ stored in *leftids* with the ID of their class (that could be a speaker ID, a language ID or any other acoustic class) stored in *rightids*. Duplicated entries are allowed in each list.

Additionally, and in order to allow more flexibility, IdMap includes two other vectors: *start‘and ‘stop* which are vectors of floats and can be used to store boudaries of audio segments.

An IdMap object is often used to store together: speaker IDs, segment IDs, start and stop time of the segment and to initialize a *StatServer*.

Note: When not used, start and stop are set to *None* meaning that the entire audio segment is selected.

Attribute	Type
leftids	ndarray of strings
rightids	ndarray of strings
start	ndarray of floats
stop	ndarray of floats

Note: all four vectors: leftids, rightids, start, stop must have the same length.

Example

We create here an *IdMap* where the *leftids* are the model names and *rightids* are the segment names. As we consider that all segments are used entirely, *start* and *stop* valuesare set to *None*.

```
import numpy
import sidekit

idmap = sidekit.IdMap()
idmap.leftids = numpy.array(["model_1", "model_2", "model_2"])
idmap.rightids = numpy.array(["segment_1", "segment_2", "segment_3"])
idmap.start = numpy.empty((3), dtype="|O")
idmap.stop = numpy.empty((3), dtype="|O")

idmap.validate()
```

In this example, the first model is associated to the first segment while the second model is linked to two segments.

The last line will return *True* if the format of idmap is correct and *False* otherwise.

Ndx

Description

Ndx objects store trials index information, i.e., combination of model and segment IDs that should be evaluated by the system which will produce a score for those trials.

The *trialmask* is a *m*-by-*n* matrix of boolean where *m* is the number of unique models and *n* is the number of unique segments. If *trialmask(i,j)* is *true* then the score between model *i* and segment *j* will be computed.

Note: it is possible to use different *Ndx* with a single *Scores* object in order to evaluate different subsets of the trials.

Attribute	Type
modelset	ndarray of strings
segset	ndarray of strings
trialmask	matrix of boolean

Example

The code below creates an *Ndx* object with two models and three segments. All trials will be computed as the *trialmask* is set to *True*.

```
import numpy
import sidekit

ndx = sidekit.Ndx()
ndx.modelset = numpy.array(["model_1", "model_2"])
ndx.segset = numpy.array(["segment_1", "segment_2", "segment_3"])
ndx.trialmask = numpy.ones((2,3), dtype='bool')

ndx.validate()
```

Keys

Description

Key are used to store information about which trial is a target trial and which one is a non-target (or impostor) trial. *tar(i,j)* is *true* if the test between model *i* and segment *j* is target. *non(i,j)* is *true* if the test between model *i* and segment *j* is non-target.

Attribute	Type
modelset	ndarray of strings
segset	ndarray of strings
tar	matrix of boolean
non	matrix of boolean

Example

We create a *Key* object that corresponds to the previously created *Ndx*.

```
import numpy
import sidekit

key = sidekit.Key()
key.modelset = ndx.modelset
key.segset = ndx.segset
key.tar = numpy.zeros((2,3), dtype='bool')
key.tar[0, 0] = True
key.tar[1:, 1:] = True
key.non = numpy.zeros((2,3), dtype='bool')
key.non[0, 1:] = True
key.non[1, 0] = True

key.validate()
```

Scores

Description

Scores include information about trials, including the lists of unique models and segments as well as the score output by the system. This class duplicate information contained in an *Ndx* in order not to depend on any *Ndx* object.

This class has four fields:

Attribute	Type
modelset	ndarray of strings
segset	ndarray of strings
scoremask	matrix of boolean
scoremat	matrix of float (scores)

StatServer

Description

StatServer are used to store and process statistics.

This class has six attributes:

- a list of models (or class ID)
- a list of segment IDs (also called shows or sessions)
- a vector of start time (one for each segment)
- a vector of stop time (one for each segment)
- zero-order statistics
- first-order statistics.

Note: that in **SIDEKIT** as an abuse of language, i-vectors and super-vectors are referred to as first order statistics.

When *Statserver* are used to store i-vectors or super-vectors as *StatServer.stat1*, *StatServer.stat0* contains the number of segments (also called sessions or shows) which have been used to estimate the i-vector or super-vector. An advantage of this abuse of language is that a single Factor Analysis implementation can be used to train Joint Factor Analysis (JFA), Total Variability or Probabilistic Linear Discriminant Analysis (PLDA).

Attribute	Type
modelset	ndarray of strings
segset	ndarray of strings
start	ndarray of floats
stop	ndarray of floats
stat0	2D-ndarray of floats
stat1	2D-ndarray of floats

Note: The size of *modelset*, *segset*, *start*, *stop*, as well as the first dimension of *stat0* and *stat1* must be equal. The second dimension of *stat1* must be a multiple of the second dimension of *stat0* (usually, stat0.shape[1] is the number of distributions of a GMM and stat1.shape[1] is the number of distributions times the dimension of the acoustic features).

Note: *StatServer* are often instantiated using an *IdMap*.

Example

Using the previously defined *IdMap*, a *FeaturesServer* (see *FeaturesServer* for more details) and a *Mixture* the following code initialize a *StatServer* and accumulates sufficient statistics.

The new *StatServer* verify:

```
stat_server.modelset == idmap.leftids
stat_server.segset == idmap.rightids
```

And statstics are coherent with the size of the GMM.

Parallel computation in SIDEKIT

Multiprocessing

SIDEKIT makes an extensive use of parallel computating to speed up the process of massive quantity of data. Implementations of **SIDEKIT** method rely on the Multiprocessing module which is part of the Python standard modules and allows the use of multiple cores on a single machine.

All methods making use the Multiprocessing parallelisation ability are using the `num_thread` parameter that defines the number of parallel process to run.

Parallelisation using Multiprocessing module is done in two ways depending on the nature of the computation.

- some methods use a Multiprocessing.Pool of process
- other methods are parallelised via a decorator that allows the ode to be written for a single Process (more readable) and to be parallelized at running time. The reading of the decorator might be tideous but the main rule when using it is: **explicit all argument names passed to the method** as the decorator parallelises the code according to the named arguments. The use of unnamed arguments might disable the parallel processing or even worse: duplicate the work made (for instance process a given list of file on each process instead of sharing the list amongst process).

MPI

Since version 1.2, SIDEKIT offers a MPI implementation of the most computational demanding methods:

- estimation of a UBM-GMM via EM
- estimation of a total variability model via EM
- extraction of i-vectors

In Python, most of the MPI functionnalities are accessible via the `mpi4py` module. MPI will launch process on one or many nodes according to your command.

The use of MPI allows to make use of multiple nodes or a full cluster by using SLURM or TORQUE for instance.

The remaining of this page describes how to run a Python script on a given list of machine using the `mpi4py` module.

To see an example of code using MPI, refer to the *Train an i-vector system using MPI*.

Writing code for MPI

When writing code for MPI you first need to create an instance of `MPI.COMM_WORLD` that manages the communication between nodes.

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
```

From this point onward, each process has a unique *rank* starting form 0. Code writen for MPI directly specifies *within the code* on which process to run which instruction. Every line of code is executed in every process unless explicitly specified.

```
print("This is Process: {} over {}".format(comm.rank, comm.size))

if comm.rank == 0:
    print("I'm process 0")
```

The code above will display:

```
This is Process: 0 over 10
This is Process: 1 over 10
This is Process: 2 over 10
This is Process: 3 over 10
This is Process: 4 over 10
```

```
This is Process: 5 over 10
This is Process: 6 over 10
This is Process: 7 over 10
This is Process: 8 over 10
This is Process: 9 over 10
I'm process 0
```

As you see, only process 0 executes the last instruction. In **SIDEKIT**, this conditional statement is mostly used to separate the master node from the others when using MAP/REDUCE approach where accumulators are summed on the master node or information are spread in all nodes from this master node.

When calling for **SIDEKIT** MPI functions, the `MPI.COMM_WORLD` instance is created within the function and should not be created outside.

To launch 10 process on a single node run

```
mpirun -np 10 ./my_script.py
```

Warning: Make sure you *my_script.py* file starts with the proper header: `#!/usr/bin/env python` as MPI needs to know what interpreter to call to execute your script.

To launch 10 process on multiple nodes run

```
mpirun --hostfile my_server_list ./my_script.py
```

Where *my_server_list* is a text file that looks like:

```
192.168.0.81:4
192.168.0.156:1
192.168.0.153:1
192.168.0.152:2
192.168.0.154:2
```

Each line of the *HOSTFILE* consists of the IP address of the node and the number of process to run on this node, both information separated by a column. In this example, the script will run on 5 nodes with a total of 10 process.

Note: each process launch by MPI is not able to fork other process on the node unless you explicitly specify (refer to the MPI documentation for more information).

At that point in time, SIDEKIT does not mix multiprocessing and MPI.

Acoustic parametrization

This part of the documentation details the different tools for acoustic parameters extraction, storage and usage. In **SIDEKIT**, low level interface for acoustic parametrization is implemented in the `frontend` module. Two high level classes allow a fast and simple extraction of acoustic parameters:

- `FeaturesExtractor`
- `FeaturesServer`

Before introducing those objects, we give a brief description of the HDF5 format that is used to store and exchange acoustic features. The HDF5 format is the prefered serialization format in **SIDEKIT**.

1. Save the features in HDF5 format

HDF5 is a portable file format that runs on different platforms and allos easy and readable serialization of data and metadata by using a hierarchical architecture.

HDF5 is the prefered file format in **SIDEKIT**, it is used to store all **SIDEKIT**'s objects such as Mixtures, StatServers, Keys, Ndx, IdMaps and feature files.

The hierarchical architecture of HDF5 files allows to save several *datasets* or *groups* in the same file.

Note: that a dataset can have different reallities. It can be a scalar value, a matrix or a complete sub directory including several sub-datasets.

Saving features per audio channel

Consider the case where your audio files have one or several audio channels (like stereo files). In this example, we consider that all the features extracted from a single audio channel are saved into one single HDF5 file.

This architecture is illustrated by the following figure:

In this example, we see two parts in the architecture. The part in the grey box exists inside the HDF5 file while the part outside the grey box is part of the OS file architecture.

In this case, we stored all our feature files in one directory: **feat** that includes two sub-directories: **sre04** and **sre05**. In **sre04**, we store two HDF5 files: **taaa.h5** and **taaf.h5**.

In our example, each of those file has the same internal organization. It includes 13 datasets (in the sense of HDF5 datasets); which are:

- **bnf** for the bottleneck features
- **bnf_mean** the mean vector of selected bottleneck features
- **bnf_std** the standard deviation vector of the selected bottleneck features
- **cep** cepstral coefficients
- **cep_mean** the mean vector of selected cepstral coefficients
- **cep_std** the standard deviation vector of the selected cepstral coefficients
- **energy** a vector of log-energy values
- **energy_mean** a scalar: mean value of the log-energy vector
- **energy_std** a scalar: standard deviation of the log-energy vector
- **fb** the filter-bank coefficients
- **fb_mean** the mean vector of selected filter-bank coefficients
- **fb_std** the standard deviation vector of the selected filter-bank coefficients
- **vad** a vector of binary values that indicates which are the selected frames

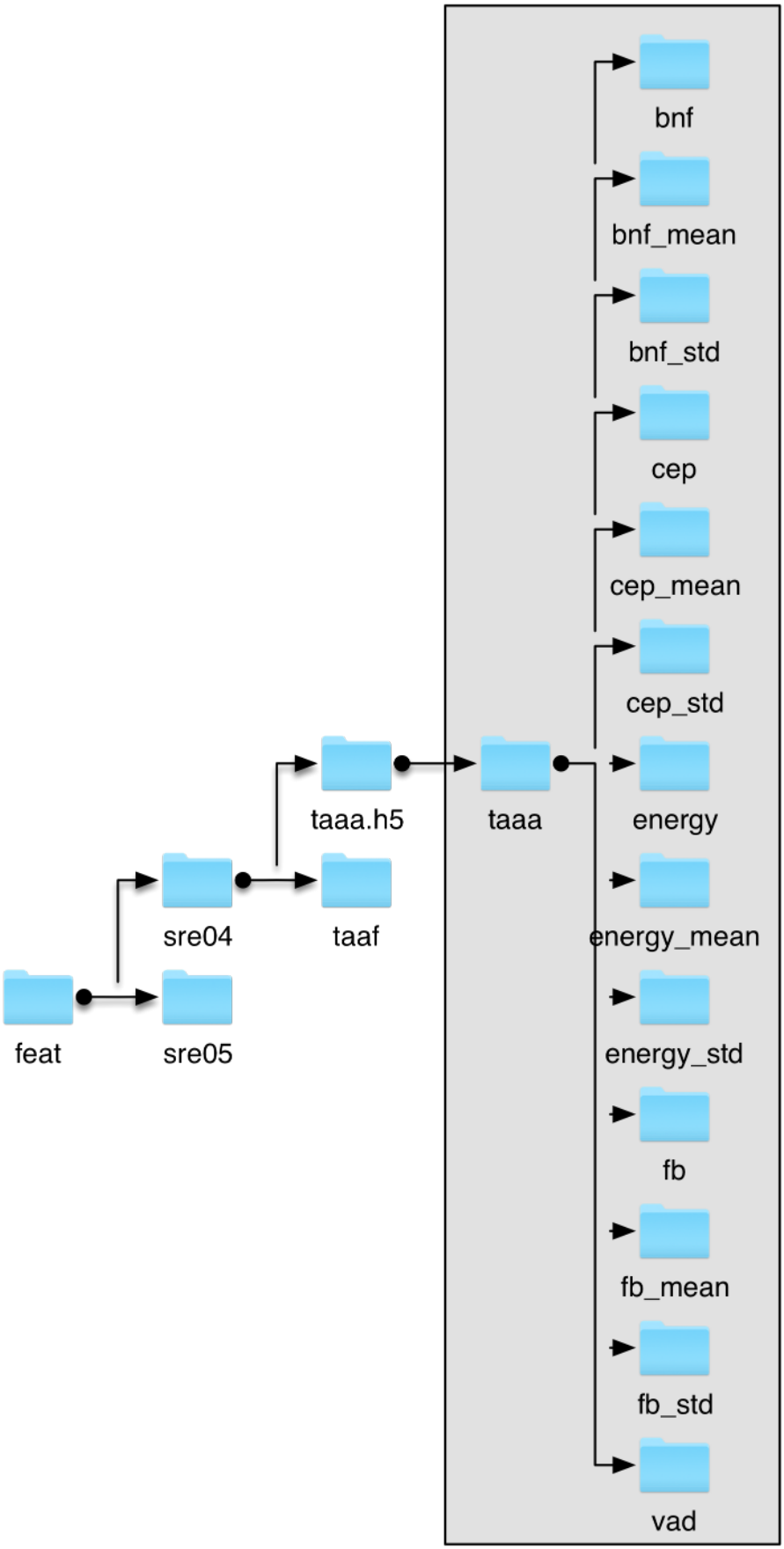


Fig. 1.1: First HDF5 structure

Saving features for a collection of audio files

In a second example, we chose tho store all features extracted from a collection of audio data in a single HDF5 file. That is: parameters extracted from all audio channels from all audio files from this collection will be store in the same HDF5 file.

An example of this structure can be seen on the following figure:

In this example, the architecture is exactly the same as the one in the firs example (see above), except that a single HDF5 file: **sre04.h5** contains features extracted from two audio channels: **taaa** and **taaf**. The two corresponding datasets in the **sre04.h5** file have the same structure as the two separated HDF5 files from the previous example (taaa.h5 and taaf.h5).

Modifications in the Python code that will use the two structures are minor but may have a great impact in term of usability and speed depending on your constraints.

Loading features from a HDF5 file

One advantage of storing different types of features into a single file is that we can load, at run time, one or several types of feature and combine them to feed our speaker/language recognition system. For instance, we will see in the following tutorials that is it easy to load the log-energy and cepstral coefficients or to combine the log-energy with the 10 first filter-bank coefficients.

2. The FeaturesExtractor object

The *FeaturesExtractor* takes audio files (WAV, SPHERE, raw PCM...) and returns feature files in HDF5 format (log-energy, cepstral coefficients, filter-bank coefficient, bottleneck features).

The interface of the *FeaturesExtractor* is as simple as possible as the main focus here is to define the parameters of the feature extraction and not to process complex operations on features.

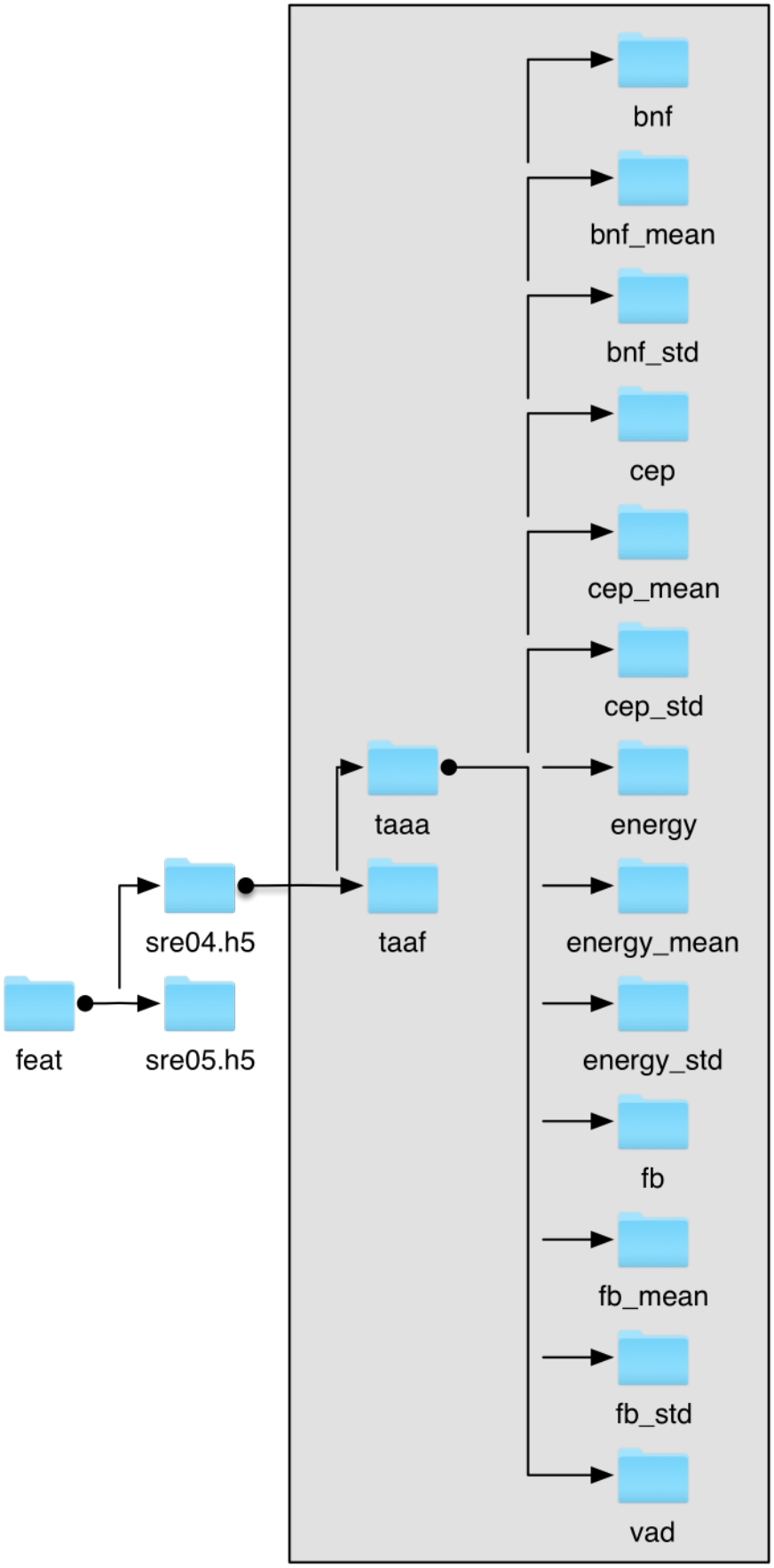


Fig. 1.2: Another example of HDF5 structure

Option	Value (default is bold)	
audio_filename_structure	**None**, a string	Structure of the input audio file name if recurrent. In case all input file name have the same structure with a different identifier, the filename is completed at run time by adding the identifier into the filename structure (see examples below).
feature_filename_structure	**{}**, a string	Structure of the output feature filename. In case all output file name have the same structure with a different identifier, the filename is completed at run time by adding the identifier into the filename structure (see examples below).
sampling_frequency	**8000**, integer	Sampling frequency of the input audio file. In case this value is lower than the real sampling frequency, the input is downsampled to match this value.
lower_frequency	**None**, float	Lower frequency of the frequency filter bank scale.
higher_frequency	**None**, float	Higher frequency of the frequency filter bank scale.
filter_bank	**None**, "lin" or "log"	Type of frequency filter bank, can be "lin" for linear scale and "log" for MEL scale.
filter_bank_size	**None**, integer	Number of filter in the filter bank
window_size	**None**, float	Size of the FFT window in seconds.
shift	**None**, float	Shift of the FFT window between two samples, in seconds.
ceps_number	**None**, integer	Number of cesptral coefficients retained.
vad	**None**, "snr", "energy", "percentil, "dnn", "lbl"	Type of Voice Activity Detection algorithm to apply. "lbl" reads from labels from file.
snr	**None**, float	Parameter of the "snr" VAD.
pre_emphasis	**0.97**, float between 0 and 1	Value used for the pre-emphasis filtering.
save_param	**["energy", "cep", "fb", "bnf", "vad"]**, list	Type of features to store in the output HDF5 file. The types are given in a Python list.
keep_all_features	**None**, boolean	If True, only store feature frames selected by the VAD. If False, store all frames.

Extract features with standardized input and output filenames

In this example, we extract features from audio files which names follow the patttern: `audio/nist_2004/{filename}.sph` where filename is a unique identifier that will be refered as `show` in the rest of the documentation See figure below:

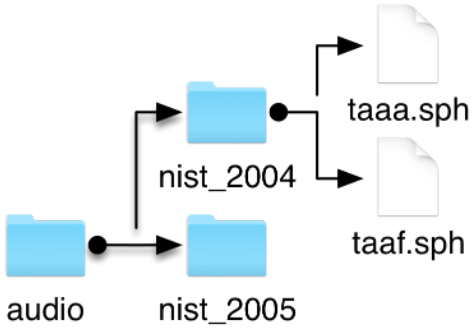


Fig. 1.3: Input audio files.

The features will be stored in files with filename pattern: `feat/sre04/{filename}.h5`, see structure below:

The `FeaturesExtractor` is instantiated with the following code:

```
extractor = sidekit.FeaturesExtractor(audio_filename_structure="audio/nist_2004/{}.sph
↪",
                                      feature_filename_structure="feat/sre04/{}.h5",
                                      sampling_frequency=None,
                                      lower_frequency=200,
                                      higher_frequency=3800,
                                      filter_bank="log",
                                      filter_bank_size=24,
                                      window_size=0.025,
                                      shift=0.01,
                                      ceps_number=20,
                                      vad="snr",
                                      snr=40,
                                      pre_emphasis=0.97,
                                      save_param=["vad", "energy", "cep", "fb"],
                                      keep_all_features=True)
```

As you can see, the `audio_filename_structure` and `feature_filename_pattern` will completed at run time and the call of:

```
extractor.save("taaa")
```

will process the file `audio/nist_2004/taaa.sph` and store features in `feat/sre04/taaa.h5`.

In case you're not interested in saving the parameters to disk, you can ask your FeaturesExtractor to return a HDF5 file handler as follow:

```
fh = extractor.extract("taaa")
```

In this case, `fh` is a HDF5 file handler.

It is also possible to process a list of audio files with a single command. The processing of the audio files can also be parallelized to speed up the process. The command is as follow:

```
show_list = ["taaa", "taaf"]
channel_list = [0, 0]

extractor.save_list(show_list=show_list,
                    channel_list=channel_list,
                    num_thread=10)
```

In this example, the processing of the audio file list will be parallelized on 10 process (useless here as there are only 2 files but you get the idea...).

Extract features with non-standardized filenames

In case your input audio file names or output feature file names don't follow any pattern, it is possible to specify the complete input filename and complete output filname as follow.

For instance, you want to process the following audio files such that:

Input audio filename	Output feature filename
audio/sre04/taaa.sph	feat/nist/taaa.h5
data/nist2005/xllb.sph	output/nist/xllb_a.h5

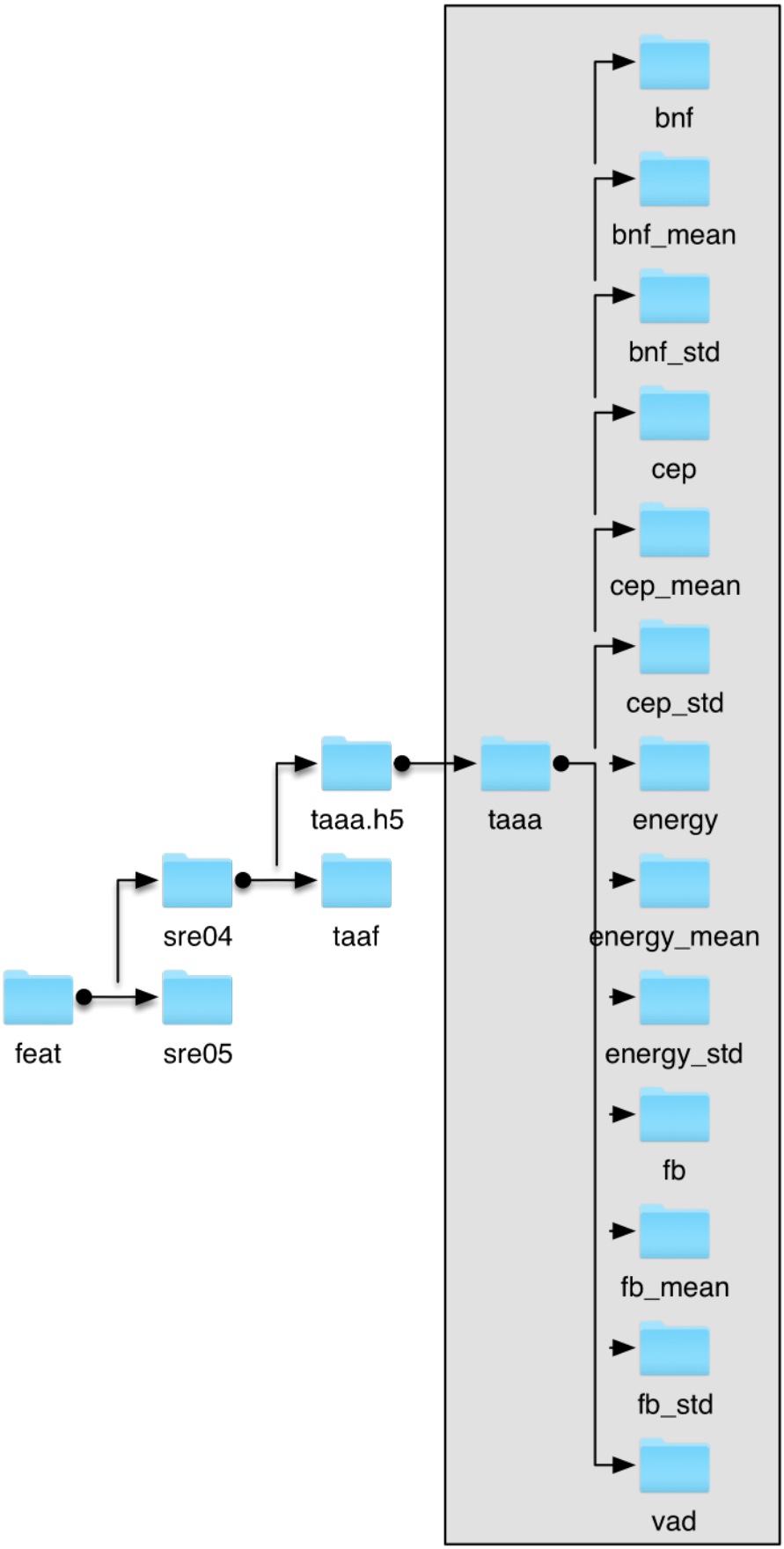


Fig. 1.4: Output HDF5 files

Let define a new FeaturesExtractor to process those files:

```
extractor = sidekit.FeaturesExtractor(audio_filename_structure=None,
                                      feature_filename_structure=None,
                                      sampling_frequency=None,
                                      lower_frequency=200,
                                      higher_frequency=3800,
                                      filter_bank="log",
                                      filter_bank_size=24,
                                      window_size=0.025,
                                      shift=0.01,
                                      ceps_number=20,
                                      vad="snr",
                                      snr=40,
                                      pre_emphasis=0.97,
                                      save_param=["vad", "energy", "cep", "fb"],
                                      keep_all_features=True)
```

And process the first file to save the features to disk:

```
extractor.save(show="taaa",
               channel=0,
               input_audio_filename="audio/sre04/taaa.sph",
               output_feature_filename="feat/nist/taaa.h5")


extractor.save(show="xllb",
               channel=0,
               input_audio_filename="data/nist2005/xllb.sph",
               output_feature_filename="output/nist/xllb_a.h5")
```

Same thing without saving to disk:

```
fh = extractor.extract(show="taaa",
                       channel=0,
                       input_audio_filename="audio/sre04/taaa.sph",
                       output_feature_filename="feat/nist/taaa.h5")
```

In order to process a list of files you'll run:

```
show_list = ["taaa", "xllb"]
input_file_list = ["audio/sre04/taaa.sph", "data/nist2005/xllb.sph"]
output_feature_list = ["feat/nist/taaa.h5", "output/nist/xllb_a.h5"]

extractor.save_list(show_list=show_list,,
                    channel_list=channel_list,
                    num_thread=10)
```

Of course you can combine input filenames without pattern and output filenames with patterns or the opposite.

Note: When using input or output filenames without patterns, you see that the `show` parameter is still used. Indeed, this parameter is used in the structure of the HDF5 file. The output feature file will look like:

As you can see, the `show` identifier is used inside the HDF5 file in order to allow storage of several feature sets in a single file. Note that it also exists although you store one single feature set in a HDF5 file.

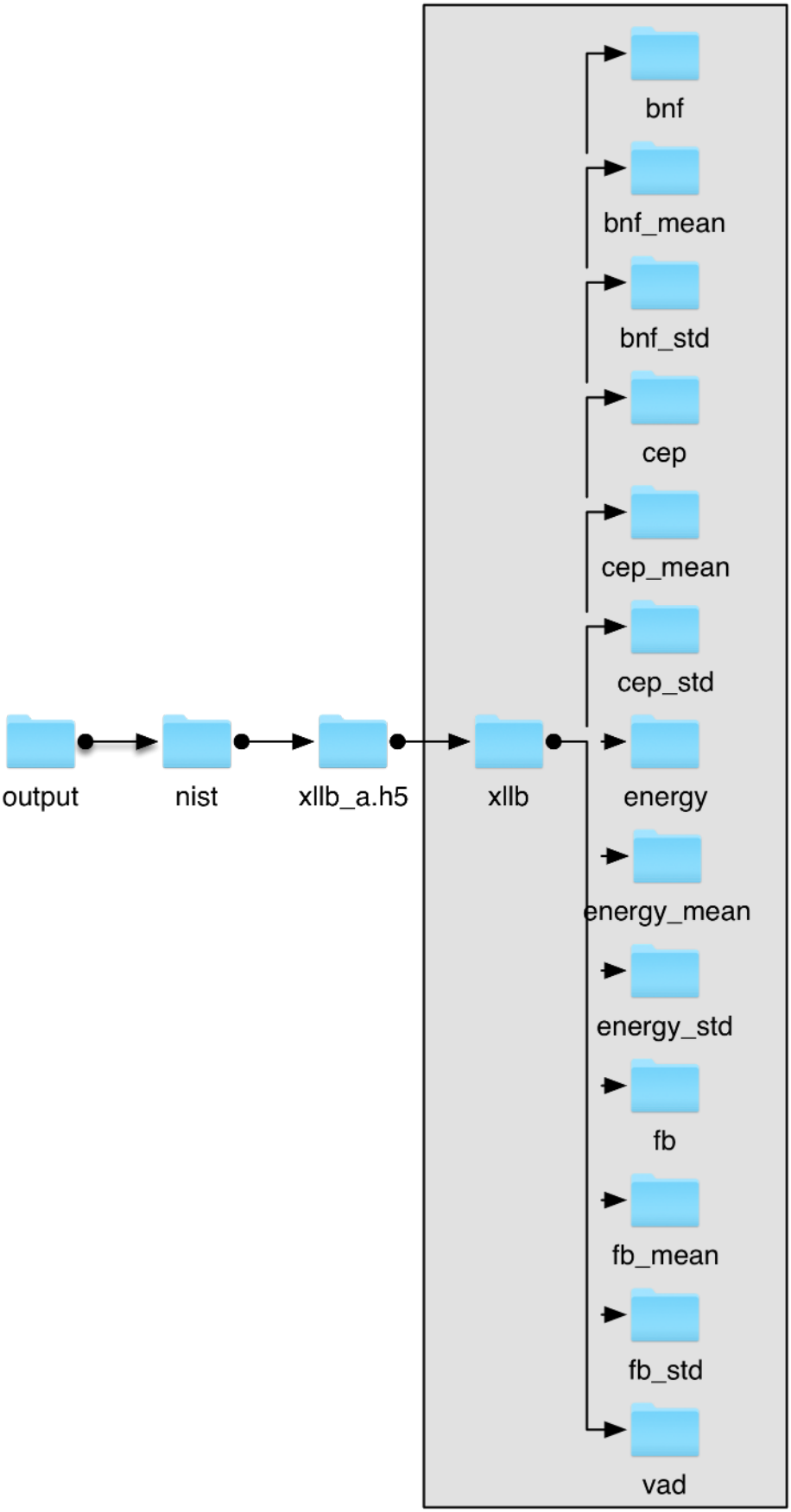


Fig. 1.5: Structure of the HDF5 feature file.

3. The FeaturesServer object

The *FeaturesServer* loads one or several datasets from one or several HDF5 files and post-process the features (normalization, addition of the temporal context, rasta filtering, feature selection...).

The *FeaturesServer* can also encapsulate one or several *FeaturesExtractor* in order to take audio files as inputs.

The *FeaturesServer* is used to feed all other objects in **SIDEKIT**.

3.1 Options of the *FeaturesServer*

Option	Value (default is bold)	
features_extractor	**None**, a FeaturesExtractor	*FeaturesExtractor* that is used to process audio files
feature_filename_structure	**{}**, a string	Structure of the output feature filename. In case all output file name have the same structure with a different identifier, the filename is completed at run time by adding the identifier into the filename structure (see examples below).
sources	**None**, tuple of tuples	tuple of sources to load features different files (optional: for the case where datasets are loaded from several files and concatenated. Each tuple includes two values, a *FeaturesServer* and a boolean. If True, VAD labels are loaded from this source
dataset_list	**None**, list of features to load	string of the form '["cep", "fb", vad", energy", "bnf"]' (only when loading datasets from a single file) list of datasets to load.
mask	**None**	string of the form '[1-3,10,15-20]' mask to apply on the concatenated dataset to select specific components. In this example, coefficients 1,2,3,10,15,16,17,18,19,20 are kept in this example,
feat_norm	**None**, "cmvn", "cms", "stg"	Type of normalization to apply as post-processing
global_cmvn	**None**, boolean	If True, use a global mean and std when normalizing the
dct_pca	**False**, boolean	if True, add temporal context by using a PCA-DCT approach
dct_pca_config	**(12, 12, None)**	Configuration of the PCA-DCT
sdc	**False**, boolean	if True, compute shifted delta cepstra coefficients
sdc_config	**(1,3,7)**,	Configuration to compute sdc coefficients
delta	**False**, float	If True, append the first order derivative
double_delta	**False**	If True, append the second order derivative
context	**(0,0)**	Add a left and right context
traps_dct_nb	**0**, integer	Number of DCT coefficients to keep when computing TRAP coefficients
rasta	**False**, boolean	If True, perform RASTA filtering
keep_all_features	**True**, boolean	If True, keep all features, if False, keep frames according to the vad labels

3.2 Get features from a single file

The simpler case is to use a *FeaturesServer* to load and process features from a single file. Natively, a *FeaturesServer* is developed to take HDF5 as input but you can provide it with its own *FeaturesExtractor* in order to extract features from an audio file and apply some post-processing on them.

Get features from a single HDF5 file

The simplest case is to use a *FeaturesServer* in order to load and post-process acoustic features from an HDF5 file. Such a *FeaturesServer* is instantiated as follow:

```
server = sidekit.FeaturesServer(features_extractor=None,
                                feature_filename_structure="feat/sre04/{}.h5",
                                sources=None,
                                dataset_list=["energy", "cep", "vad"],
                                mask="[0-12]",
                                feat_norm="cmvn",
                                global_cmvn=None,
                                dct_pca=False,
                                dct_pca_config=None,
                                sdc=False,
                                sdc_config=None,
                                delta=True,
                                double_delta=True,
                                delta_filter=None,
                                context=None,
                                traps_dct_nb=None,
                                rasta=True,
                                keep_all_features=True)
```

In this example, the *FeaturesServer* will be used to load and concatenate cepstral coefficients and log-energy from a single HDF5 file. The selected features (which can be: energy, cep, fb and bnf) will be concatenated in a predefined order so the order of the list given as a parameter is not important. This order is: energy, cep, fb and bnf. Once these features loaded, only the first 13 coefficients are retained and post-processed (from index 0 to 12 included as given by the mask parameter).

The post-processing can include the following steps in this order:

- rasta filtering
- addition of the temporal context first and second derivatives, DCT-PCA or Shifted Delta Cepstra.
- normalization of the features using either Cepstral Mean Variance Normalization (cmvn), Cepstral Mean Subtraction (cms) or Short term Gaussianization (stg).
- frame selection according to the VAD labels that are loaded if "vad" is included in the dataset_list. If "vad" is not in the dataset_list, then all frames are kept

This *FeaturesServer* is then used as follow:

```
load(self, show, channel=0, input_feature_filename=None, label=None, start=None,␣
↪stop=None)
```

Get features from a single audio file

In case you don't want to store your features on disk as an HDF5 file it is possible to use a *FeaturesServer* including a *FeaturesExtractor* in order to compute the acoustic parameters from an audio file and to select and post-process the features on-the-fly.

In this case, the *FeaturesServer* is created as follow:

```
server = sidekit.FeaturesServer(features_extractor=extractor,
                                feature_filename_structure=None,
                                sources=None,
                                dataset_list=["energy", "cep", "vad"],
                                mask="[0-12]",
                                feat_norm="cmvn",
                                global_cmvn=None,
                                dct_pca=False,
                                dct_pca_config=None,
                                sdc=False,
                                sdc_config=None,
                                delta=True,
                                double_delta=True,
                                delta_filter=None,
                                context=None,
                                traps_dct_nb=None,
                                rasta=True,
                                keep_all_features=True)
```

Note: The *FeaturesExtractor* has to be created before.

This *FeaturesServer* can then be used as follow:: features, label = server.load(show, channel=0, input_feature_filename=featureFileName, label=None, start=None, stop=None)

3.3 Get features from several files

Sometimes you might want to combine features coming from different files.

Get features from several HDF5 files

Using a *FeaturesServer* it is possible to combine features coming from different HDF5 files In the following example, we have two sets of featurefiles which have been saved in HDF5 format. Files from the first set have a name with the pattern *filename.h5* while file names from the second set follow the pattern *filename_2.h5*.

We are going to load energy from the first set and a few cepstral coefficients from the second set to combine them. The VAD labels will also be taken from the second set. For this purpose, we create two feature servers (one for each set) as follow:

```
fs_1 = sidekit.FeaturesServer(feature_filename_structure="{}.h5",
                              dataset_list=["energy"],
                              context=None)

fs_2 = sidekit.FeaturesServer(feature_filename_structure="{}_2.h5",
                             dataset_list=["cep", "vad"],
                             mask="[0-12]",
```

```
                                delta=True,
                                double_delta=True,
                                rasta=True)
```

As you can see, no post processing is applied on the log-energy from the first file while derivatives are added to the 13 first cepstral coefficients from the second set after applying rasta filtering.

The last step consists now in ccreating a third FeaturesServer that will call *fs_1* and *fs_2* and then combine the two types of feature before applying a post processing on the complete features:

```
fs = sidekit.FeaturesServer(sources=((fs_1, False), (fs_2, True)),
                                feat_norm="cmvn",
                                keep_all_features=False)
```

Energy form the first set is concatenated to the cepstral coefficients from the second set together with their first and second derivatives. Eventually, CMVN is applied on the entire features and only selected frames are kept based on the VAD label from the second set. All this is done by calling:

```
feat, label = fs.load("taaa")
```

The resulting features are 40 dimensional feature frames (13 cepstral coefficients + 13 deltas + 13 delta-deltas and the log-energy).

Get features from one audio file and one HDF5 file

You can use more complex combinations by concatenating features extracted on-line from one audio file and features from an HDF5 feature file. We'll get the same features as in the previous example except that the energy from set 1 is directly computed from the audio file. Cepstral coefficients are still taken from the already extracted features.

We first create a *FeaturesExtractor* to process the audio file and the associated *FeaturesServer* that will manage the *FeaturesExtractor*:

```
extractor = sidekit.FeaturesExtractor(audio_filename_structure="{}.wav",
                                      sampling_frequency=8000,
                                      lower_frequency=0,
                                      higher_frequency=4000,
                                      filter_bank="log",
                                      filter_bank_size=40,
                                      window_size=0.025,
                                      shift=0.01,
                                      ceps_number=20,
                                      vad="snr",
                                      snr=40,
                                      pre_emphasis=0.97,
                                      save_param=["energy"],
                                      keep_all_features=True)

fs_1 = sidekit.FeaturesServer(features_extractor=extractor,
                                feature_filename_structure=None,
                                sources=None,
                                vad="snr",
                                snr=40,
                                dataset_list=["energy"],
                                keep_all_features=True)
```

Then, we create a second *FeaturesServer* that will load cepstral coefficients from the second set of feature files and perform some post processing:

```
fs_2 = sidekit.FeaturesServer(feature_filename_structure="{}_2.h5",
                              dataset_list=["cep", "vad"],
                              mask="[0-12]",
                              delta=True,
                              double_delta=True,
                              rasta=True)
```

We now combine the two *FeaturesExtractor* in a third one and perform CMVN:

```
fs = sidekit.FeaturesServer(sources=((fs_1, False), (fs_2, True)),
                                feat_norm="cmvn",
                                keep_all_features=False)
```

The resulting features are obtained by:

```
feat, label = fs.load("taab")
```

Train a Universal Background Model

Universal Background (Gaussian mixture) Models (UBM) are trained via EM algorithm using the Mixture class from **SIDEKIT**.

UBM are trained using acoustic features that can be extracted on-line or loaded and post-processed from existing HDF5 feature files. We acknowledge that UBM training might not be the most efficient as post processing of the acoustic features is performed on-line (computation of the derivatives, concatenation of the different types of features, normalization) and that iterating over the data might be time consuming. However, given the performance of parallel computing and the fact that a large quantity of data is not necessary to train good quality models, we chose to use this approach which greatly reduces the feature storage on disk.

1. Training using EM split

Training of a GMM-UBM with diagonal covariance is straightforward:

- create a Mixture
- perform EM training

The two instructions are:

```
ubm = sidekit.Mixture()

ubm.EM_split(features_server=fs,
             feature_list=ubm_list,
             distrib_nb=1024,
             iterations=(1, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8),
             num_thread=10,
             save_partial=False,
             ceil_cov=10,
             floor_cov=1e-2
             )
```

In the above example, a GMM is trained with 1024 distributions. Note that to perform the EM training you need to provide the following parameters:

- a FeaturesServer that will be used to load data from disk or to process on the fly
- a list of feature files to process (following the FeaturesServer requirements)

- the final expected number of distribution as a power of 2
- a tuple of iteration numbers where the i_th component is the number of iteration to run for the i_th size of model
- a number of parallel process to run

You might also save the model after each iteration by setting *save_partial* to True. You can constrain the covariance of the distribution by providing a ceiling and flooring values.

The training process is as follow:

- initialize one Gaussian distribution given all the training data
- Iterate n_i iterations of EM with the current size of model (fixed number of distributions)
- split all distributions in two according to their variance in order to double the number of distributions of the GMM
- save the resulting model

As an example, given the above *iterations* parameter, if training a model with 1024 distributions SIDEKIT will perform:

- 1 iteration of EM with 1 distribution
- 2 iterations of EM with 2 distributions
- 2 iterations of EM with 4 distributions
- 4 iterations of EM with 8 distributions
- 4 iterations of EM with 16 distributions
- 4 iterations of EM with 32 distributions
- 4 iterations of EM with 64 distributions
- 8 iterations of EM with 128 distributions
- 8 iterations of EM with 256 distributions
- 8 iterations of EM with 512 distributions
- 8 iterations of EM with 1024 distributions

2. Training using simple EM with fixed number of distributions

It is also possible to train a GMM with EM algorithm by directly setting the number of distributions. In this case, the distributions will be initialized by taking the mean and covariance of random subsets of the training data. The code is as follow:

```
ubm = sidekit.Mixture()

ubm.EM_uniform(cep,
               distrib_nb,
               iteration_min=3,
               iteration_max=10,
               llk_gain=0.01,
               do_init=True
               )
```

Be careful that the arguments of this method are a bit different from the previous one. Indeed, this method doesn't use any FeaturesServer but takes a ndarray containing all features as rows.

3. Training using EM split on several nodes

SIDEKIT allows parallel training of GMM using several nodes (machines) via the Message Passing Interface (MPI).

First, make sure MPI is installed on each node you intend to use.

Then enable the use of MPI by setting your environment variable to something like: `SIDEKIT="mpi=true"`.

You're now ready to train your GMM by running:

```
ubm = sidekit.Mixture()
sidekit.sidekit_mpi.EM_split(ubm=ubm,
                             features_server=fs,
                             feature_list=ubm_list,
                             distrib_nb=2048,
                             output_filename="ubm_tandem_mpi",
                             iterations=(1, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8, 8, 8),
                             llk_gain=0.01,
                             save_partial=True,
                             ceil_cov=10,
                             floor_cov=1e-2,
                             num_thread=30)
```

Where:

- `fs` is a `FeaturesServer` object used to load the acoustic features
- `ubm_list` is a list of shows (sessions) to process

Parameter `num_thread` is related to Multiprocessing that is used to load the features at first on Node 0. Note that Multiprocessing is not used later in the process.

Refer to the *Parallel computation in SIDEKIT*. page to see how to launch your computation on several nodes.

4 Full covariance UBM

In order to train full covariance GMMs you can first train a GMM with diagonal covariance using one of the two above methods then perform a number of EM iterations to estimate the full covariance matrices. This can be implementated as follows:

```
ubm = sidekit.Mixture()

ubm.EM_split(features_server,
             feature_list,
             distrib_nb,
             iterations=(1, 2, 2, 4, 4, 4, 4, 8, 8, 8, 8, 8, 8),
             num_thread=10,
             llk_gain=0.01,
             save_partial=False,
             ceil_cov=10,
             floor_cov=1e-2
             )

ubm.EM_convert_full(features_server,
                    featureList,
                    distrib_nb,
                    iterations=2,
                    num_thread=10
                    )
```

The method *EM_convert_full* can be applied on a previously trained diagonal Mixture. We use here a FeaturesServer to access the acoustic frames and a list of shows (sessions). The only two other parameters are the number of EM iterations to run and the number of thread in case you want to paralllize the process.

Train an i-vector extractor

Total Variability models (TV) are trained via EM algorithm using the FactorAnalyser class from **SIDEKIT**.

TV are trained using sufficient statistics that are accumulated using a `StatServer` object (or a neural network). The training also required a UBM of type Mixture.

SIDEKIT provides four implementations of the Total Variability EM estimation. Three are methods of the FactorAnalyser class while the fourth one is available in the `sidekit_mpi` module and required the installation of the MPI library.

1. **total_variability_raw** that is provided for didactic purpose, the code is written using the plain (raw) mathematical formulas without any optimization.
2. **total_variability_single** that provides a single process implementation of the EM algorithm. this version runs on a single process on a single machine but has been optimized
3. **total_variability** is the parallelised and optimised implementation. This method makes use of the Multiprocessing module to parallelise computation on a single machine.

1. Get to know the algorithm with total_variability_raw

We strongly **encourage** you to **READ** the code if this method to understand how the EM algorithm works for total variability model.

We strongly **discourage** you to **USE** this method as it is absolutely not optimized.

For a usable version of the same method refer to section 3 a(or 2) below.

2. Using a single process on one machine

Training of a TV model on a single machine, single process. Before running:

- train a GMM-UBM of type Mixture
- accumulate sufficient statistics using a StatServer object

You can then train the TV model by running:

```
fa = sidekit.FactorAnalyser()

fa.total_variability_single(stat_server_filename,
                            ubm,
                            tv_rank,
                            nb_iter=20,
                            min_div=True,
                            tv_init=None,
                            batch_size=300,
                            save_init=False,
                            output_file_name=None)
```

In this example:

- **stat_server_filename** is a list of file names for StatServer containing sufficient statistics of all sessions to train the TV model
- **ubm** is the Mixture object for which the sufficient statistics have been computed
- **tv_rank** is an integer, it is the rank of the resulting Total Variability matrix (size of the i-vectors)
- **nb_iter** is the number of iterations to run for the EM algorithm
- **min_div** is a boolean, if True every iteration include a Minimum divergence re-estimation step
- **tv_init** is a matrix used to initialize the training if None, the matrix is initialized randomly
- **batch_size** is the number of session that are processed at once to reduce memory footprint
- **save_init** is a boolean, if True, the initial model is saved
- **output_file_name** is the name of the file the model will be saved to

3. Using multiple process on one machine with Python MultiProcessing

Training of a TV model on a single machine, multiple process. Before running:

- train a GMM-UBM of type Mixture
- accumulate sufficient statistics using a StatServer object

You can then train the TV model by running:

```
fa = sidekit.FactorAnalyser()

fa.total_variability(stat_server_filename,
                     ubm,
                     tv_rank,
                     nb_iter=20,
                     min_div=True,
                     tv_init=None,
                     batch_size=300,
                     save_init=False,
                     output_file_name=None,
                     num_thread=1)
```

In this example:

- **stat_server_filename** is a list of file names for StatServer containing sufficient statistics of all sessions to train the TV model
- **ubm** is the Mixture object for which the sufficient statistics have been computed
- **tv_rank** is an integer, it is the rank of the resulting Total Variability matrix (size of the i-vectors)
- **nb_iter** is the number of iterations to run for the EM algorithm
- **min_div** is a boolean, if True every iteration include a Minimum divergence re-estimation step
- **tv_init** is a matrix used to initialize the training if None, the matrix is initialized randomly
- **batch_size** is the number of session that are processed at once to reduce memory footprint
- **save_init** is a boolean, if True, the initial model is saved
- **output_file_name** is the name of the file the model will be saved to
- **num_thread** is the number of process to run on the machine

Warning: The batchsize parameter might cause troubles due to the limitation of the Pickle module. Objects and data are exchanged between process via pickling which does not accept "too big" objects.

Note that Numpy and Scipy are linked to the low level BLAS library that might also parallelise the computation on multiple cores. Thus don't set a number of process that is too high.

We recommend setting the number of parallel process between 5 and 10 depending on your machine.

4. Using multiple process on multiple nodes with MPI

See *Parallel computation in SIDEKIT* for details about MPI installation and use.

Training of a TV model on a single machine, multiple process. Before running:

- train a GMM-UBM of type Mixture
- accumulate sufficient statistics using a StatServer object

You can then train the TV model by running:

```
fa = sidekit.FactorAnalyser()

fa = sidekit.sidekit_mpi.total_variability(stat_server_filename,
                                           ubm,
                                           tv_rank=10,
                                           nb_iter=10,
                                           min_div=True,
                                           tv_init=fa_init.F,
                                           save_init=False,
                                           output_file_name="tv_mpi")
```

In this example:

- **stat_server_filename** is a list of file names for StatServer containing sufficient statistics of all sessions to train the TV model
- **ubm** is the Mixture object for which the sufficient statistics have been computed
- **tv_rank** is an integer, it is the rank of the resulting Total Variability matrix (size of the i-vectors)
- **nb_iter** is the number of iterations to run for the EM algorithm
- **min_div** is a boolean, if True every iteration include a Minimum divergence re-estimation step
- **tv_init** is a matrix used to initialize the training if None, the matrix is initialized randomly
- **save_init** is a boolean, if True, the initial model is saved
- **output_file_name** is the name of the file the model will be saved to

Extract your I-Vectors

Once trained a Universal Background Model (GMM or DNN) and a Total Variability matrix, you are now ready to extract i-vectors.

Starting from version 1.2 of **SIDEKIT**, the extraction process is managed with a *FactorAnalyser*.

Considering that you have already created:

- a *Mixture*, *ubm*, to use as a UBM

- a *FeaturesServer*, *features_server*, to load acoustic features
- a *FactorAnalyser*

and that you have computed sufficient statistics on one or multiple set of segments and that those statistics are stored in one or multiple *StatServer*, *stat_server*.

1. Extract i-vectors in a single process

The following code wil extract i-vectors for the set of segments which statistics are in *stat_server*.

```
fa = sidekit.FactorAnalyser()

iv, iv_uncertainty = fa.extract_ivectors_single(ubm,
                                                stat_server,
                                                uncertainty=True)
```

Where:

- **ubm** is a `Mixture`
- **stat_server** is an object of type `StatServer`
- **uncertainty** is a boolean, if True, the method also returns a matrix where each line is the diagonal of the of the uncerainty matrix of the corresponding i-vector.

Note: `iv` is a *StatServer* that contains i-vectors in *stat1* and ones in *stat0*.

---

2. Extract i-vectors on multiple process on a single node

The following code wil extract i-vectors for the set of segments which statistics are in *stat_server* using multiple process on a single machine.

Due to the limitations of the Multiprocessing module (related to the pickling of objects), we advertise to keep *batchsize* of a few hundred sessions.

```
fa = sidekit.FactorAnalyser()

iv, iv_uncertainty = fa.extract_ivectors(ubm,
                                         stat_server_filename,
                                         prefix='',
                                         batch_size=300,
                                         uncertainty=False,
                                         num_thread=1)
```

Where:

- **ubm** is a `Mixture`
- **stat_server_filename** is the name of an HDF5 containing a `StatServer`
- **prefix** is the prefix of the statistic data set within the HDF5 file
- **batch_size** number of sessions to process on each process
- **uncertainty** is a boolean, if True, the method also returns a matrix where each line is the diagonal of the of the uncertainty matrix of the corresponding i-vector.

- **num_thread**, number of process to run in parallel

3. Extract i-vectors on multiple nodes

SIDEKIT also provide a function to extract i-vectors on several nodes (machines) which is especially appropriate for big size models (> 4000 distributions).

Refer to the *Parallel computation in SIDEKIT*. page to see how to launch your computation on several nodes.

The code to execute should look like this:

```
fa = sidekit.FactorAnalyser()

sidekit.sidekit_mpi.extract_ivector(stat_server_file_name,
                                    ubm,
                                    output_file_name,
                                    uncertainty=False,
                                    prefix='')
```

Where:

- **stat_server_filename** is a filename of a **StatServer** containing sufficient statistics that will be used to generate i-vectors
- **ubm** is a `Mixture` object
- **output_file_name** name of the HDF5 file where i-vectors will be stored
- **uncertainty** is a boolean, if True, the method also returns a matrix where each line is the diagonal of the of the uncertainty matrix of the corresponding i-vector. This matrix is stored on disk in a HDF5 file.
- **prefix** is the prefix of the sufficient statistics in the HDF5 file

Train a deep neural network with Theano and SIDEKIT

Requirement

Train an ASR system using **'KALDI <>'**_ or use the alignments provided in *Download lists for standard datasets*.

The following example makes use of an alignment file generated using KALDI for Switchboard 1.

Train your Feed-Forward DNN

Bottleneck features extraction

How to train a DNN

How to extract the features

Phonetically aware Neural Network for speaker recognition

How to train a DNN

how to estimate the UBM

How to accumulate the statistics

1.5.2 Run a standard experiment

In this section, you will find short tutorials on how to use the different components of the toolkt individually and longer complete tutorial to train and run a speaker verification system on standard tasks such as the RSR2015 database or the NIST Speaker Recognition Evaluation.

RSR2015

RSR2015 is a database collected by the Institute for Infocomm Research (I2R),
A*STAR in Singapore to support the development and evaluation of
text-dependent speaker verification algorithms.
This database can be bought from I2R at a very low price.
For more details, visit RSR2015 website.

Metat data (Keys, IdMap and Ndx) can be downloaded from *Download lists for standard datasets*.

Prepare to run experiments on RSR2015

Before running experiments on the RSR2015 database
you need to run the script `rsr2015_init.py`
in order to prepare the lists of file and indexes required.

To work, this script only requires you to modify the path
where the RSR2015 database is stored

We assume here that the sphere files from the RSR2015 have been
decompressed in the original directory and that the architecture
of the directories follows the original architecture provided by the I2R.

i.e.:

rsr2015_root_directory
    - key
    - sph
        - male
        - female

Enter the path where the **RSR2015** database is stored and run *python rsr2015_init.py*. This script generates the following files:

- `task/3sess-pwd_eval_m_back.h5`
- `task/3sess-pwd_eval_m_key.h5`
- `task/3sess-pwd_eval_m_key.h5`
- `task/3sess-pwd_eval_m_nap.h5`
- `task/3sess-pwd_eval_m_ndx.h5`

- task/3sesspwd_eval_m_trn.h5
- task/ubm_list.txt

Below is a description of this script.

First, loads the required PYTHON packages.

```
import numpy as np
import sidekit
import os
import sys
import re
import random
import pandas as pd
pd.set_option('display.mpl_style', 'default')
```

Before running this script, don't forget to enter the path of the directory where the RSR2015 database is stored:

```
rsr2015Path = '/Users/larcher/LIUM/RSR2015/RSR2015_V1/'
```

The rest of the script generates the files defining the enrollment data, the trials to process and the key for scoring from the original files provided with the **RSR2015**.

The IdMap object is created from the *3sesspwd_eval_m.trn* file and save to disk:

```
rsrEnroll = pd.read_csv(rsr2015Path + '/key/part1/trn/3sesspwd_eval_m.trn', delimiter=
↪'[,\s*]', header=None, engine='python')

# remove the extension of the files (.sph)
for i in range(1,4):
    rsrEnroll[i] = rsrEnroll[i].str.replace('.sph$', '')
    rsrEnroll[i] = rsrEnroll[i].str.replace('^male/', '')

# Create the list of models and enrollment sessions
models = []
segments = []
for idx, mod in enumerate(rsrEnroll[0]):
    models.extend([mod, mod, mod])
    segments.extend([rsrEnroll[1][idx], rsrEnroll[2][idx], rsrEnroll[3][idx]])

# Create and fill the IdMap with the enrollment definition
enroll_idmap = sidekit.IdMap()
enroll_idmap.leftids = np.asarray(models)
enroll_idmap.rightids = np.asarray(segments)
enroll_idmap.start = np.empty(enroll_idmap.rightids.shape, '|O')
enroll_idmap.stop = np.empty(enroll_idmap.rightids.shape, '|O')
enroll_idmap.validate()
enroll_idmap.write('task/3sesspwd_eval_m_trn.h5')
```

The file *3sess-pwd_eval_m.ndx* is read and we extract information to process **target** trials as well as **nontarget** trials that correspond to the case of an impostor pronouncing the correct sentence. The Key object is stored in HDF5 format:

```
rsrKey = pd.read_csv(rsr2015Path + '/key/part1/ndx/3sess-pwd_eval_m.ndx', delimiter=
↪'[,\s*]', header=None, engine='python')
rsrKey[1] = rsrKey[1].str.replace('.sph$', '')

models = []
testsegs = []
trials = []
```

```
for idx,model in enumerate(list(rsrKey[0])):
    if (rsrKey[2][idx] == 'Y'):
        models.append(rsrKey[0][idx])
        testsegs.append(rsrKey[1][idx])
        trials.append('target')
    elif (rsrKey[4][idx] == 'Y'):
        models.append(rsrKey[0][idx])
        testsegs.append(rsrKey[1][idx])
        trials.append('nontarget')

key = sidekit.Key(models=np.array(models), testsegs=np.array(testsegs), trials=np.
↪array(trials))

key.write('task/3sess-pwd_eval_m_key.h5')
```

The index file that defines the trials to process is derived from the `Key` object and stored to disk in HDF5 format:

```
ndx = key.to_ndx()
ndx.write('task/3sess-pwd_eval_m_ndx.h5')
```

The following block creates a list of files that will be used to train a Universal Background Model. This list is stored in ASCII format. All the 30 sentences from the PART I of the **RSR2015** database from the 50 male speakers of the background set are used to train the UBM:

```
ubmList = []
p = re.compile('(.*)((m0[0-4][0-9])|(m050))(.*)((0[0-2][0-9])|(030))(\.sph$)')
for dir_, _, files in os.walk(rsr2015Path):
    for fileName in files:
        if p.search(fileName):
            relDir = os.path.relpath(dir_, rsr2015Path + "/sph/male")
            relFile = os.path.join(relDir, fileName)
            ubmList.append(os.path.splitext(relFile)[0])
with open('task/ubm_list.txt','w') as of:
    of.write("\n".join(ubmList))
```

The next section creates the list of files used to train the Nuisance Projection Attribute matrix that can be used for SVM-GMM tutorial:

```
napSegments = ubmList[::7]
napSpeakers = [seg.split('/')[0] for seg in napSegments]
nap_idmap = sidekit.IdMap()
nap_idmap.leftids = np.array(napSpeakers)
nap_idmap.rightids = np.array(napSegments)
nap_idmap.start = np.empty(nap_idmap.rightids.shape, '|O')
nap_idmap.stop = np.empty(nap_idmap.rightids.shape, '|O')
nap_idmap.validate()
nap_idmap.write('task/3sess-pwd_eval_m_nap.h5')
```

Generate now the list of models that will be used as blacklist to train the Support Vector Machines:

```
backSegments = random.sample(ubmList, 200)
backSpeakers = [seg.split('/')[0] for seg in backSegments]
back_idmap = sidekit.IdMap()
back_idmap.leftids = np.array(backSpeakers)
back_idmap.rightids = np.array(backSegments)
back_idmap.start = np.empty(back_idmap.rightids.shape, '|O')
back_idmap.stop = np.empty(back_idmap.rightids.shape, '|O')
```

```
back_idmap.validate()
back_idmap.write('task/3sess-pwd_eval_m_back.h5')
```

Eventually creates the `IdMap` to compute statistics of the test segments for the tutorial on SVMs:

```
test_idmap = sidekit.IdMap()
test_idmap.leftids = ndx.segset
test_idmap.rightids = ndx.segset
test_idmap.start = np.empty(test_idmap.rightids.shape, '|O')
test_idmap.stop = np.empty(test_idmap.rightids.shape, '|O')
test_idmap.validate()
test_idmap.write('task/3sess-pwd_eval_m_test.h5')
```

Run a GMM-UBM system

This script run an experiment on the male evaluation part of the RSR2015 database. The protocols used here is based on the one described in [Larcher14]. In this version, we only consider the non-target trials where impostors pronounce the correct text (Imp Correct).

The number of Target trials performed is then - TAR correct: 10,244 - IMP correct: 573,664

[Larcher14] Anthony Larcher, Kong Aik Lee, Bin Ma and Haizhou Li, "Text-dependent speaker verification: Classifiers, databases and RSR2015," in Speech Communication 60 (2014) 56–77

Input/Output

Enter:

the number of distribution for the Gaussian Mixture Models the root directory where the RSR2015 database is stored

Generates the following outputs:

- a Mixture in HDF5 format (ubm)
- a StatServer of zero and first-order statistics (enroll_stat)
- a StatServer containing the super vectors of MAP adapted GMM models for each speaker (enroll_sv)
- a score file
- a DET plot

First, loads the required PYTHON packages:

```
import sidekit
import os
import sys
import multiprocessing
import matplotlib.pyplot as mpl
import logging
import numpy as np

logging.basicConfig(filename='log/rsr2015_ubm-gmm.log',level=logging.DEBUG)
```

Set your own parameters

```
distribNb = 512  # number of Gaussian distributions for each GMM
rsr2015Path = '/lium/corpus/audio/tel/en/RSR2015_v1/'

# Default for RSR2015
audioDir = os.path.join(rsr2015Path , 'sph/male')

# Automatically set the number of parallel process to run.
# The number of threads to run is set equal to the number of cores available
# on the machine minus one or to 1 if the machine has a single core.
nbThread = max(multiprocessing.cpu_count()-1, 1)
```

Load IdMap, Ndx, Key from HDF5 files and ubm_list

Note that these files are generated when running rsr2015_init.py:

```
print('Load task definition')
enroll_idmap = sidekit.IdMap('task/3sesspwd_eval_m_trn.h5')
test_ndx = sidekit.Ndx('task/3sess-pwd_eval_m_ndx.h5')
key = sidekit.Key('task/3sess-pwd_eval_m_key.h5')
with open('task/ubm_list.txt') as inputFile:
    ubmList = inputFile.read().split('\n')
```

Process the audio to save MFCC on disk

```
logging.info("Initialize FeaturesExtractor")
extractor = sidekit.FeaturesExtractor(audio_filename_structure=audioDir+"/{}.wav",
                                      feature_filename_structure="./features/{}.h5",
                                      sampling_frequency=16000,
                                      lower_frequency=133.3333,
                                      higher_frequency=6955.4976,
                                      filter_bank="log",
                                      filter_bank_size=40,
                                      window_size=0.025,
                                      shift=0.01,
                                      ceps_number=19,
                                      vad="snr",
                                      snr=40,
                                      pre_emphasis=0.97,
                                      save_param=["vad", "energy", "cep"],
                                      keep_all_features=False)

# Get the complete list of features to extract
show_list = np.unique(np.hstack([ubmList, enroll_idmap.rightids, np.unique(test_ndx.
↪segset)]))
channel_list = np.zeros_like(show_list, dtype = int)

logging.info("Extract features and save to disk")
extractor.save_list(show_list=show_list,
                    channel_list=channel_list,
                    num_thread=nbThread)
```

Create a FeaturesServer

From this point, all objects that need to process acoustic features will do it through a *FeaturesServer*. This object is initialized here. We define the type of parameters to load (log-energy + cepstral coefficients) and the post-process to apply on the fly (RASTA filtering, CMVN, addition iof the first and second derivatives, feature selection).

```
# Create a FeaturesServer to load features and feed the other methods
features_server = sidekit.FeaturesServer(features_extractor=None,
                                         feature_filename_structure="./features/{}.h5
↪",
                                         sources=None,
                                         dataset_list=["energy", "cep", "vad"],
                                         mask=None,
                                         feat_norm="cmvn",
                                         global_cmvn=None,
                                         dct_pca=False,
                                         dct_pca_config=None,
                                         sdc=False,
                                         sdc_config=None,
                                         delta=True,
                                         double_delta=True,
                                         delta_filter=None,
                                         context=None,
                                         traps_dct_nb=None,
                                         rasta=True,
                                         keep_all_features=False)
```

Train the Universal background Model (UBM)

```
print('Train the UBM by EM')
# Extract all features and train a GMM without writing to disk
ubm = sidekit.Mixture()
llk = ubm.EM_split(features_server, ubmList, distribNb, num_thread=nbThread, save_
↪partial=True)
ubm.write('gmm/ubm.h5')
```

Compute the sufficient statistics on the UBM

Make use of the new UBM to compute the sufficient statistics of all enrolement sessions that should be used to train the speaker GMM models. An empty StatServer is initialized from the enroll_idmap IdMap. Statistics are then computed in the enroll_stat StatServer which is then stored in compressed pickle format:

```
print('Compute the sufficient statistics')
# Create a StatServer for the enrollment data and compute the statistics
enroll_stat = sidekit.StatServer(enroll_idmap,
                                 distrib_nb=512,
                                 feature_size=60)
enroll_stat.accumulate_stat(ubm=ubm,
                            feature_server=features_server,
                            seg_indices=range(enroll_stat.segset.shape[0]),
                            num_thread=nbThread)
enroll_stat.write('data/stat_rsr2015_male_enroll.h5')
```

Adapt the GMM speaker models from the UBM via a MAP adaptation

Train a GMM for each speaker. Only adapt the mean supervector and store all of them in the enrol_sv StatServer that is then stored to disk:

```
print('MAP adaptation of the speaker models')
regulation_factor = 3  # MAP regulation factor
enroll_sv = enroll_stat.adapt_mean_map_multisession(ubm, regulation_factor)
enroll_sv.write('data/sv_rsr2015_male_enroll.h5')
```

Compute all trials and save scores in HDF5 format

```
print('Compute trial scores')
scores_gmm_ubm = sidekit.gmm_scoring(ubm,
                                     enroll_sv,
                                     test_ndx,
                                     features_server,
                                     num_thread=nbThread)
scores_gmm_ubm.write('scores/scores_gmm-ubm_rsr2015_male.h5')
```

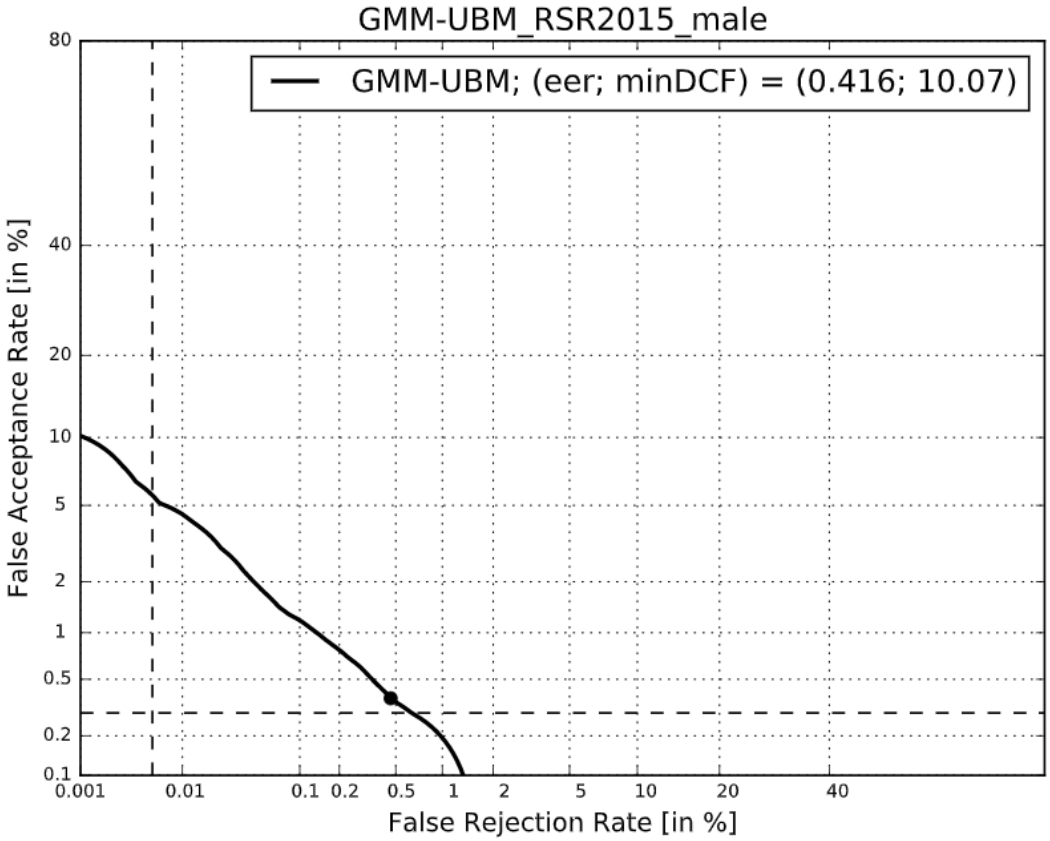
Plot DET curve and compute minDCF and EER

```
print('Plot the DET curve')
# Set the prior following NIST-SRE 2008 settings
prior = sidekit.logit_effective_prior(0.01, 10, 1)

# Initialize the DET plot to 2008 settings
dp = sidekit.DetPlot(window_style='sre10', plot_title='GMM-UBM_RSR2015_male')
dp.set_system_from_scores(scores_gmm_ubm, key, sys_name='GMM-UBM')
dp.create_figure()
dp.plot_rocch_det(0)
dp.plot_DR30_both(idx=0)
dp.plot_mindcf_point(prior, idx=0)
```

Compute equal error rate and minDCF, plot the DET curve.

```
print('Plot DET curves')
prior = sidekit.logit_effective_prior(0.001, 1, 1)
minDCF, Pmiss, Pfa, prbep, eer = sidekit.bosaris.detplot.fast_minDCF(dp.__tar__[0],␣
↪dp.__non__[0], prior, normalize=True)
print("UBM-GMM 128g, minDCF = {}, eer = {}".format(minDCF, eer))
```

The following results should be obtained at the end of this tutorial:



Run a SVM GMM system on the RSR2015 database

This script run an experiment on the male evaluation part of the **RSR2015** database. The protocol used here is based on the one described in [Larcher14]. In this version, we only consider the non-target trials where impostors pronounce the correct text (Imp Correct).

The number of Target trials performed is then - TAR correct: 10,244 - IMP correct: 573,664

[Larcher14] Anthony Larcher, Kong Aik Lee, Bin Ma and Haizhou Li, "Text-dependent speaker verification: Classifiers, databases and RSR2015," in Speech Communication 60 (2014) 56–77

Input/Output

Enter:

- the number of distribution for the Gaussian Mixture Models
- the root directory where the RSR2015 database is stored

Generates the following outputs:

- a Mixture in compressed pickle format (ubm)

- a StatServer of zero and first-order statistics (enroll_stat)
- a StatServer of zero and first-order statistics (back_stat)
- a StatServer of zero and first-order statistics (nap_stat)
- a StatServer of zero and first-order statistics (test_stat)
- a StatServer containing the super vectors of MAP adapted GMM models for each speaker (enroll_sv)
- a StatServer containing the super vectors of MAP adapted GMM models for each speaker (back_sv)
- a StatServer containing the super vectors of MAP adapted GMM models for each speaker (nap_sv)
- a StatServer containing the super vectors of MAP adapted GMM models for each speaker (test_sv)
- a score file
- a DET plot

```
import numpy as np
import sidekit
import multiprocessing
import os
import sys
import matplotlib.pyplot as mpl
import logging

logging.basicConfig(filename='log/rsr2015_svm-gmm.log',level=logging.DEBUG)
```

Set your own parameters

```
distrib_nb = 512  # number of Gaussian distributions for each GMM
NAP = True  # activate the Nuisance Attribute Projection
nap_rank = 40

rsr2015Path = '/lium/corpus/vrac/RSR2015_V1/'

# Set the number of parallel process to run.
nbThread = 10
```

Load IdMap, Ndx, Key from HDF5 files and ubm_list

that define the task. Note that these files are generated when running `rsr2015_init.py`:

```
logging.info('Load task definition')
enroll_idmap = sidekit.IdMap('task/3sesspwd_eval_m_trn.h5')
nap_idmap = sidekit.IdMap('task/3sess-pwd_eval_m_nap.h5')
back_idmap = sidekit.IdMap('task/3sess-pwd_eval_m_back.h5')
test_ndx = sidekit.Ndx('task/3sess-pwd_eval_m_ndx.h5')
test_idmap = sidekit.IdMap('task/3sess-pwd_eval_m_test.h5')
key = sidekit.Key('task/3sess-pwd_eval_m_key.h5')

with open('task/ubm_list.txt') as inputFile:
    ubmList = inputFile.read().split('\n')
```

Process the audio to save MFCC on disk

```
logging.info("Initialize FeaturesExtractor")
extractor = sidekit.FeaturesExtractor(audio_filename_structure=audioDir+"/{}.wav",
                                      feature_filename_structure="./features/{}.h5",
                                      sampling_frequency=16000,
                                      lower_frequency=133.3333,
                                      higher_frequency=6955.4976,
                                      filter_bank="log",
                                      filter_bank_size=40,
                                      window_size=0.025,
                                      shift=0.01,
                                      ceps_number=19,
                                      vad="snr",
                                      snr=40,
                                      pre_emphasis=0.97,
                                      save_param=["vad", "energy", "cep"],
                                      keep_all_features=False)

# Get the complete list of features to extract
show_list = np.unique(np.hstack([ubmList, enroll_idmap.rightids, np.unique(test_ndx.
↪segset)]))
channel_list = np.zeros_like(show_list, dtype = int)

logging.info("Extract features and save to disk")
extractor.save_list(show_list=show_list,
                    channel_list=channel_list,
                    num_thread=nbThread)
```

Create a FeaturesServer

From this point, all objects that need to process acoustic features will do it through a *FeaturesServer*. This object is initialized here. We define the type of parameters to load (log-energy + cepstral coefficients) and the post-process to apply on the fly (RASTA filtering, CMVN, addition iof the first and second derivatives, feature selection).

```
# Create a FeaturesServer to load features and feed the other methods
features_server = sidekit.FeaturesServer(features_extractor=None,
                                         feature_filename_structure="./features/{}.h5
↪",
                                         sources=None,
                                         dataset_list=["energy", "cep", "vad"],
                                         mask=None,
                                         feat_norm="cmvn",
                                         global_cmvn=None,
                                         dct_pca=False,
                                         dct_pca_config=None,
                                         sdc=False,
                                         sdc_config=None,
                                         delta=True,
                                         double_delta=True,
                                         delta_filter=None,
                                         context=None,
                                         traps_dct_nb=None,
                                         rasta=True,
                                         keep_all_features=False)
```

Train the Universal background Model (UBM)

An empty Mixture is initialized and an EM algorithm is run to estimate the UBM before saving it to disk. Covariance matrices are diagonal in this example.

```
logging.info('Train the UBM by EM')
# load all features in a list of arrays
ubm = sidekit.Mixture()
llk = ubm.EM_split(features_server,
                   ubmList,
                   distrib_nb,
                   num_thread=nbThread)
ubm.write('gmm/ubm.h5')
```

Compute the sufficient statistics on the UBM

Make use of the new UBM to compute the sufficient statistics of all enrolement sessions that should be used to train the speaker GMM models, models for the SVM training blacklist, segments to train the NAP matrix and test segments. An empty StatServer is initialized. Statistics are then computed in the StatServer which is then stored to disk:

```
logging.info()
enroll_stat = sidekit.StatServer(enroll_idmap,
                                 distrib_nb=512,
                                 feature_size=60)
enroll_stat.accumulate_stat(ubm=ubm,
                            feature_server=features_server,
                            seg_indices=range(enroll_stat.segset.shape[0]),
                            num_thread=nbThread)
enroll_stat.write('data/stat_rsr2015_male_enroll.h5')

back_stat = sidekit.StatServer(back_idmap,
                                 distrib_nb=512,
                                 feature_size=60)
back_stat.accumulate_stat(ubm=ubm,
                          feature_server=features_server,
                          seg_indices=range(back_stat.segset.shape[0]),
                          num_thread=nbThread)
back_stat.write('data/stat_rsr2015_male_back.h5')

nap_stat = sidekit.StatServer(nap_idmap,
                                 distrib_nb=512,
                                 feature_size=60)
nap_stat.accumulate_stat(ubm=ubm,
                         feature_server=features_server,
                         seg_indices=range(nap_stat.segset.shape[0]),
                         num_thread=nbThread)
nap_stat.write('data/stat_rsr2015_male_nap.h5')

test_stat = sidekit.StatServer(test_idmap,
                                 distrib_nb=512,
                                 feature_size=60)
test_stat.accumulate_stat(ubm=ubm,
                          feature_server=features_server,
                          seg_indices=range(test_stat.segset.shape[0]),
                          num_thread=nbThread)
test_stat.write('data/stat_rsr2015_male_test.h5')
```

Train a GMM for each session

Only adapt the mean super-vector and store all of them in the enrol_sv StatServer that is then stored in compressed picked format:

```
logging.info('MAP adaptation of the speaker models')
regulation_factor = 3  # MAP regulation factor

enroll_sv = enroll_stat.adapt_mean_map(ubm, regulation_factor, norm=True)
enroll_sv.write('data/sv_norm_rsr2015_male_enroll.h5')

back_sv = back_stat.adapt_mean_map(ubm, regulation_factor, norm=True)
back_sv.write('data/sv_rsr2015_male_back.h5')

nap_sv = nap_stat.adapt_mean_map(ubm, regulation_factor, norm=True)
nap_sv.write('data/sv_rsr2015_male_nap.h5')

test_sv = test_stat.adapt_mean_map(ubm, regulation_factor, norm=True)
test_sv.write('data/sv_rsr2015_male_test.h5')
```

Apply Nuisance Attribute Projection if required

If `NAP == True`, estimate and apply the Nuisance Attribute Projection on all supervectors:

```
if NAP:
    logging.info('Estimate and apply NAP')
    napMat = back_sv.get_nap_matrix_stat1(nap_rank);
    back_sv.stat1 = back_sv.stat1 - np.dot(np.dot(back_sv.stat1, napMat), napMat.
↪transpose())
    enroll_sv.stat1 = enroll_sv.stat1 - np.dot(np.dot(enroll_sv.stat1, napMat),␣
↪napMat.transpose())
    test_sv.stat1 = test_sv.stat1 - np.dot(np.dot(test_sv.stat1, napMat), napMat.
↪transpose())
```

Train the Support Vector Machine models

Train a Support Vector Machine for each speaker by considering the three sessions of this speaker:

```
logging.info('Train the SVMs')
sidekit.svm_training('svm/', back_sv, enroll_sv, num_thread=nbThread)
```

Compute all trials and save scores in HDF5 format

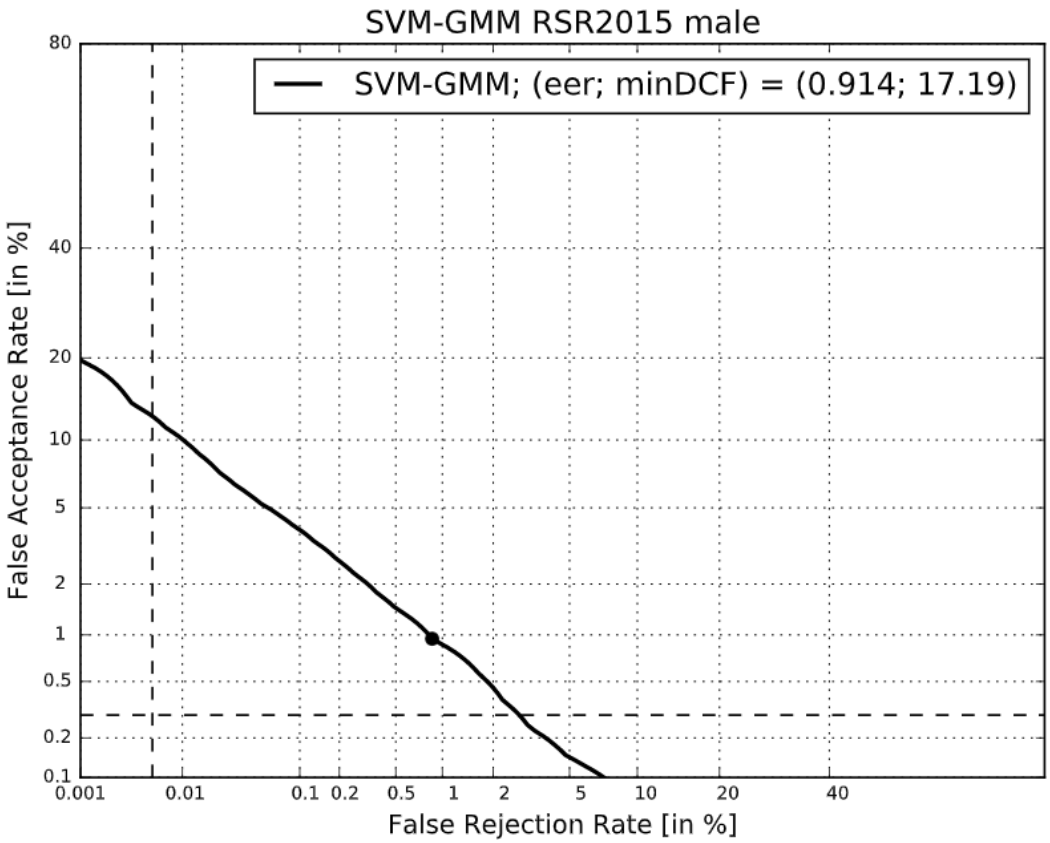
Compute the scores for all trials:

```
logging.info('Compute trial scores')
scores_gmm_svm = sidekit.svm_scoring('svm/{}.svm', test_sv, test_ndx, num_
↪thread=nbThread)
if NAP:
    scores_gmm_svm.write('scores/scores_svm-gmm_NAP_rsr2015_male.h5')
else:
    scores_gmm_svm.write('scores/scores_svm-gmm_rsr2015_male.h5')
```

Plot DET curve and compute minDCF and EER

```
logging.info('Plot the DET curve')
prior = sidekit.logit_effective_prior(0.01, 10, 1)

# Initialize the DET plot to 2008 settings
dp = sidekit.DetPlot(window_style='sre10', plot_title='SVM-GMM RSR2015 male')
dp.set_system_from_scores(scores_gmm_svm, key, sys_name='SVM-GMM')
dp.create_figure()
dp.plot_rocch_det(0)
dp.plot_DR30_both(idx=0)
dp.plot_mindcf_point(prior, idx=0)

minDCF, Pmiss, Pfa, prbep, eer = sidekit.bosaris.detplot.fast_minDCF(dp.__tar__[0],␣
↪dp.__non__[0], prior, normalize=True)
logging.info("minDCF = {}, eer = {}".format(minDCF, eer))
```

After running this script you should obtain the following curve



NIST-SRE 2010

NIST Speaker Recognition Evaluation have been organized by the

National Institute for Standard and Technologies (US) since 1997.
Those evaluation have become a standard amongst the scientific community
and are well used to develop and evaluate the speaker recognition systems.
The proposed tutorials are based on the NIST-SRE 2010 extended protocol and makes use
of previous evaluation data to train the systems (NIST-SRE 2004, 2005, 2006 and 2008)

More information can be found on NIST-SRE website.

Three tutorials are proposed here to train three i-vectors systems.

Run an *i*-vector system

This script runs an experiment on the male NIST Speaker Recognition Evaluation 2010 extended core task. For more details about the protocol, refer to the NIST-SRE website.

In order to get this scirpt running on your machine, you will need to modify a limited number of options to indicate where your features are located and how many threads you want to run in parallel.

Getting ready

Load your favorite modules before going further.

```
import sidekit
```

Set parameters of your system:

```
distrib_nb = 2048  # number of Gaussian distributions for each GMM
rank_TV = 400  # Rank of the total variability matrix
tv_iteration = 10  # number of iterations to run
plda_rk = 400  # rank of the PLDA eigenvoice matrix
feature_dir = '/lium/spk1/larcher/mfcc_24/'  # directory where to find the features
feature_extension = 'h5'  # Extension of the feature files
nbThread = 10  # Number of parallel process to run
```

Load list of files to process. All the files neede to run this tutorial are available at *Download lists for standard datasets*.

```
with open("task/ubm_list.txt", "r") as fh:
    ubm_list = np.array([line.rstrip() for line in fh])
tv_idmap = sidekit.IdMap("task/tv_idmap.h5")
plda_male_idmap = sidekit.IdMap("task/plda_male_idmap.h5")
enroll_idmap = sidekit.IdMap("task/core_male_sre10_trn.h5")
test_idmap = sidekit.IdMap("task/test_sre10_idmap.h5")
```

The lists needed are:

- the list of files to train the GMM-UBM
- an IdMap listing the files to train the total variability matrix
- an IdMap to train the PLDA, WCCN, Mahalanobis matrices
- the IdMap listing the enrolment segments and models
- the IdMap describing the test segments

Load Key and Ndx:

```
test_ndx = sidekit.Ndx("task/core_core_all_sre10_ndx.h5")
keys = sidekit.Key('task/core_core_all_sre10_cond5_key.h5')
```

Define the FeaturesServer to load the acoustic features:

```
fs = sidekit.FeaturesServer(feature_filename_structure="{dir}/{{}}.{ext}".
↪format(dir=feature_dir, ext=feature_extension),
                            dataset_list=["energy", "cep", "vad"],
                            mask="[0-12]",
                            feat_norm="cmvn",
                            keep_all_features=False,
                            delta=True,
                            double_delta=True,
                            rasta=True,
                            context=None)
```

Train your system

Train now the UBM-GMM using EM algorithm and write it to disk. After each iteration, the current version of the mixture is written to disk.

```
ubm = sidekit.Mixture()
llk = ubm.EM_split(fs, ubm_list, distrib_nb, num_thread=nbThread, save_partial='gmm/
↪ubm')
ubm.write('gmm/ubm_{}.h5'.format(distrib_nb))
```

Create StatServers for the enrollment, test and background data and compute the statistics:

```
enroll_stat = sidekit.StatServer(enroll_idmap, ubm)
enroll_stat.accumulate_stat(ubm=ubm, feature_server=fs, seg_indices=range(enroll_stat.
↪segset.shape[0]) ,num_thread=nbThread)
enroll_stat.write('data/stat_sre10_core-core_enroll_{}.h5'.format(distrib_nb))

test_stat = sidekit.StatServer(test_idmap, ubm)
test_stat.accumulate_stat(ubm=ubm, feature_server=fs, seg_indices=range(test_stat.
↪segset.shape[0]), num_thread=nbThread)
test_stat.write('data/stat_sre10_core-core_test_{}.h5'.format(distrib_nb))

back_idmap = plda_all_idmap.merge(tv_idmap)
back_stat = sidekit.StatServer(back_idmap, ubm)
back_stat.accumulate_stat(ubm=ubm, feature_server=fs, seg_indices=range(back_stat.
↪segset.shape[0]), num_thread=nbThread)
back_stat.write('data/stat_back_{}.h5'.format(distrib_nb))
```

Train Total Variability Matrix for i-vector extraction. After each iteration, the matrix is saved to disk.

```
tv_stat = sidekit.StatServer.read_subset('data/stat_back_{}.h5'.format(distrib_nb),␣
↪tv_idmap)
tv_mean, tv, _, __, tv_sigma = tv_stat.factor_analysis(rank_f = rank_TV,
                                                       rank_g = 0,
                                                       rank_h = None,
                                                       re_estimate_residual = False,
                                                       it_nb = (tv_iteration,0,0),
                                                       min_div = True,
                                                       ubm = ubm,
```

```
                                                    batch_size = 100,
                                                    num_thread = nbThread,
                                                    save_partial = "data/TV_{}".
↪format(distrib_nb))
sidekit.sidekit_io.write_tv_hdf5((tv, tv_mean, tv_sigma), "data/TV_{}".format(distrib_
↪nb))
```

Extract i-vectors for target models, training and test segments:

```
enroll_stat = sidekit.StatServer('data/stat_sre10_core-core_enroll_{}.h5'.
↪format(distrib_nb))
enroll_iv = enroll_stat.estimate_hidden(tv_mean, tv_sigma, V=tv, batch_size=100, num_
↪thread=nbThread)[0]
enroll_iv.write('data/iv_sre10_core-core_enroll_{}.h5'.format(distrib_nb))

test_stat = sidekit.StatServer('data/stat_sre10_core-core_test_{}.h5'.format(distrib_
↪nb))
test_iv = test_stat.estimate_hidden(tv_mean, tv_sigma, V=tv, batch_size=100, num_
↪thread=nbThread)[0]
test_iv.write('data/iv_sre10_core-core_test_{}.h5'.format(distrib_nb))

plda_stat = sidekit.StatServer.read_subset('data/stat_back_{}.h5'.format(distrib_nb),␣
↪plda_all_idmap)
plda_iv = plda_stat.estimate_hidden(tv_mean, tv_sigma, V=tv, batch_size=100, num_
↪thread=nbThread)[0]
plda_iv.write('data/iv_plda_{}.h5'.format(distrib_nb))
```

Run the tests

```
keys = []
for cond in range(9):
    keys.append(sidekit.Key('/lium/buster1/larcher/nist/sre10/core_core_{}_sre10_cond
↪{}_key.h5'.format("all", cond + 1)))

enroll_iv = sidekit.StatServer('data/iv_sre10_core-core_enroll_{}.h5'.format(distrib_
↪nb))
test_iv = sidekit.StatServer('data/iv_sre10_core-core_test_{}.h5'.format(distrib_nb))
plda_iv = sidekit.StatServer.read_subset('data/iv_plda_{}.h5'.format(distrib_nb),␣
↪plda_male_idmap)
```

Using Cosine similarity

A simple cosine scoring without any normalization of the i-vectors.

```
scores_cos = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv, test_ndx, wccn =␣
↪None)
```

A version where *i*-vectors are normalized using Within Class Covariance normalization (WCCN).

```
wccn = plda_iv.get_wccn_choleski_stat1()
scores_cos_wccn = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv, test_ndx,␣
↪wccn=wccn)
```

The same with a Linear Discriminant Analysis performed first to reduce the dimension of *i*-vectors to 150 dimensions.

```
LDA = plda_iv.get_lda_matrix_stat1(150)

plda_iv_lda = copy.deepcopy(plda_iv)
enroll_iv_lda = copy.deepcopy(enroll_iv)
test_iv_lda = copy.deepcopy(test_iv)

plda_iv_lda.rotate_stat1(LDA)
enroll_iv_lda.rotate_stat1(LDA)
test_iv_lda.rotate_stat1(LDA)

scores_cos_lda = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda, test_iv_lda, test_
↪ndx, wccn=None)
```

And now combine LDA and WCCN:

```
wccn = plda_iv_lda.get_wccn_choleski_stat1()
scores_cos_lda_wcnn = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda, test_iv_lda,␣
↪test_ndx, wccn=wccn)
```

Using Mahalanobis distance

If the scoring is 'mahalanobis', *i*-vectors are normalized using one iteration of the Eigen Factor Radial algorithm (equivalent to the so called length-normalization). Then scores are computed using a Mahalanobis distance.

```
meanEFR, CovEFR = plda_iv.estimate_spectral_norm_stat1(3)

plda_iv_efr1 = copy.deepcopy(plda_iv)
enroll_iv_efr1 = copy.deepcopy(enroll_iv)
test_iv_efr1 = copy.deepcopy(test_iv)

plda_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
enroll_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
test_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
M1 = plda_iv_efr1.get_mahalanobis_matrix_stat1()
scores_mah_efr1 = sidekit.iv_scoring.mahalanobis_scoring(enroll_iv_efr1, test_iv_efr1,
↪ test_ndx, M1)
```

Using Two-covariance scoring

If the scoring is '2cov', two 2-covariance models are trained with and without *i*-vector normalization. The normalization applied consists of one iteration of Spherical Noramlization.

```
W = plda_iv.get_within_covariance_stat1()
B = plda_iv.get_between_covariance_stat1()
scores_2cov = sidekit.iv_scoring.two_covariance_scoring(enroll_iv, test_iv, test_ndx,␣
↪W, B)

meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')

plda_iv_sn1 = copy.deepcopy(plda_iv)
enroll_iv_sn1 = copy.deepcopy(enroll_iv)
test_iv_sn1 = copy.deepcopy(test_iv)

plda_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
```

```
enroll_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
test_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])

W1 = plda_iv_sn1.get_within_covariance_stat1()
B1 = plda_iv_sn1.get_between_covariance_stat1()
scores_2cov_sn1 = sidekit.iv_scoring.two_covariance_scoring(enroll_iv_sn1, test_iv_
↪sn1, test_ndx, W1, B1)
```

Using Probabilistic Linear Discriminant Analysis

Normalize i-vector using Spherical Nuisance Normalization and compute scores using Probabilistic Linear Discriminant Analysis

```
meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')

plda_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
enroll_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
test_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])

plda_mean, plda_F, plda_G, plda_H, plda_Sigma = plda_iv.factor_analysis(rank_f=plda_
↪rk,
                                                                        rank_g=0,
                                                                        rank_h=None,
                                                                        re_estimate_
↪residual=True,
                                                                        it_nb=(10,0,
↪0),
                                                                        min_div=True,
                                                                        ubm=None,
                                                                        batch_
↪size=1000,
                                                                        num_
↪thread=nbThread)

sidekit.sidekit_io.write_plda_hdf5((plda_mean, plda_F, plda_G, plda_Sigma), "data/
↪plda_model_tel_m_{}.h5".format(distrib_nb))

scores_plda = sidekit.iv_scoring.PLDA_scoring(enroll_iv, test_iv, test_ndx, plda_mean,
↪ plda_F, plda_G, plda_Sigma, full_model=False)
```

Plot the DET curves

In case you want to display the results of the experiments. First define the target prior, the parameters of the graphic window and the title of the plot.

```
# Set the prior following NIST-SRE 2010 settings
prior = sidekit.logit_effective_prior(0.001, 1, 1)
# Initialize the DET plot to 2010 settings
dp = sidekit.DetPlot(windowStyle='sre10', plotTitle='I-Vectors SRE 2010-ext male,␣
↪cond 5')
```

For each of the performed experiments, load the target and non-target scores for the condition 5 according to the key file.

```
dp.set_system_from_scores(scores_cos, keys, sys_name='Cosine')
dp.set_system_from_scores(scores_cos_wccn, keys, sys_name='Cosine WCCN')
dp.set_system_from_scores(scores_cos_lda, keys, sys_name='Cosine LDA')
dp.set_system_from_scores(scores_cos_wccn_lda, keys, sys_name='Cosine WCCN LDA')

dp.set_system_from_scores(scores_mah_efr1, keys, sys_name='Mahalanobis EFR')

dp.set_system_from_scores(scores_2cov, keys, sys_name='2 Covariance')
dp.set_system_from_scores(scores_2cov_sn1, keys, sys_name='2 Covariance Spherical Norm
↪')

dp.set_system_from_scores(scores_plda, keys, sys_name='PLDA')
```

Create the window and plot:

```
dp.create_figure()
dp.plot_rocch_det(0)
dp.plot_rocch_det(1)
dp.plot_rocch_det(2)
dp.plot_rocch_det(3)
dp.plot_rocch_det(4)
dp.plot_rocch_det(5)
dp.plot_rocch_det(6)
dp.plot_rocch_det(7)
dp.plot_DR30_both(idx=0)
dp.plot_mindcf_point(prior, idx=0)
```

Depending of the data available, the following plot could be obtained at the end of this tutorial: (For this example, data used include NIST-SRE 04, 05, 06, 08, the SwitchBoard Part 2 phase 2 and 3 and Cellular part 2) Those results are far from optimal as don't generalize on other conditions of NIST-SRE 2010. This system has been trained without any specific data selection and its purpose is only to give an idea of what you can obtain.

DNN-UBM I-vector

This script runs an experiment on the male NIST Speaker Recognition Evaluation 2010 extended core task. For more details about the protocol, refer to the NIST-SRE website.

In order to get this scirpt running on your machine, you will need to modify a limited number of options to indicate where your features are located and how many threads you want to run in parallel.

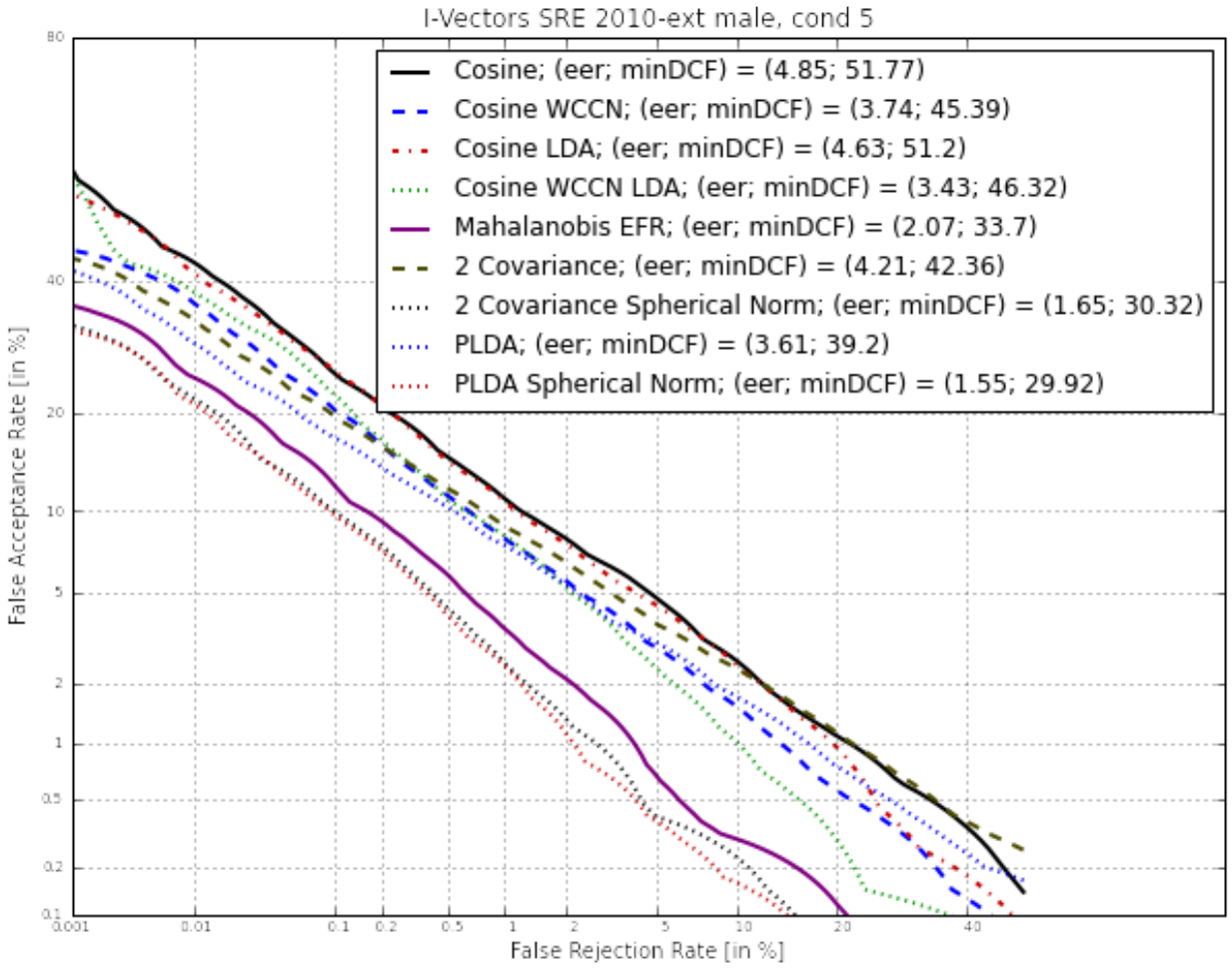
Getting ready

Load your favorite modules before going further.

```
import sidekit
```

Set parameters of your system:

```
distrib_nb = 2048  # number of Gaussian distributions for each GMM
rank_TV = 400  # Rank of the total variability matrix
tv_iteration = 10  # number of iterations to run
plda_rk = 400  # rank of the PLDA eigenvoice matrix
feature_dir = '/lium/spk1/larcher/mfcc_24/'  # directory where to find the features
feature_extension = 'h5'  # Extension of the feature files
nbThread = 10  # Number of parallel process to run
```


I-Vectors SRE 2010-ext male, cond 5
Cosine; (eer; minDCF) = (4.85; 51.77)
Cosine WCCN; (eer; minDCF) = (3.74; 45.39)
Cosine LDA; (eer; minDCF) = (4.63; 51.2)
Cosine WCCN LDA; (eer; minDCF) = (3.43; 46.32)
Mahalanobis EFR; (eer; minDCF) = (2.07; 33.7)
2 Covariance; (eer; minDCF) = (4.21; 42.36)
2 Covariance Spherical Norm; (eer; minDCF) = (1.65; 30.32)
PLDA; (eer; minDCF) = (3.61; 39.2)
PLDA Spherical Norm; (eer; minDCF) = (1.55; 29.92)
False Acceptance Rate [in %]
False Rejection Rate [in %]
80
40
20
10
5
2
1
0.5
0.2
0.1
0.001
0.01
0.1
0.2
0.5
1
2
5
10
20
40

Load list of files to process. All the files neede to run this tutorial are available at *Download lists for standard datasets*.

```
with open("task/ubm_list.txt", "r") as fh:
    ubm_list = np.array([line.rstrip() for line in fh])
tv_idmap = sidekit.IdMap("task/tv_idmap.h5")
plda_male_idmap = sidekit.IdMap("task/plda_male_idmap.h5")
enroll_idmap = sidekit.IdMap("task/core_male_sre10_trn.h5")
test_idmap = sidekit.IdMap("task/test_sre10_idmap.h5")
```

The lists needed are:

- the list of files to train a Neural Network
- the frame alignments provided by an ASR system
- the list of files to train the GMM-UBM
- an IdMap listing the files to train the total variability matrix
- an IdMap to train the PLDA, WCCN, Mahalanobis matrices
- the IdMap listing the enrolment segments and models
- the IdMap describing the test segments

Load Key and Ndx:

```
test_ndx = sidekit.Ndx("task/core_core_all_sre10_ndx.h5")
keys = sidekit.Key('task/core_core_all_sre10_cond5_key.h5')
```

Define one FeaturesServer to load the acoustic features used to train the Neural Network and then to compute the zero order statistics. Here we use Filter Bank coefficients. We use `global_cmvn` normalization to normalize segments by using the mean and variance computed on the entire file.

A second FeaturesServer is created to provide a second set of acoustic features used to compute first and second order statistics. In this tutorial, we use classic MFCC features.

```
feature_context = (7, 7)
fs_dnn = sidekit.FeaturesServer(feature_filename_structure=feature_dir+"{}.h5",
                                dataset_list = ["fb"],
                                context=feature_context,
                                feat_norm="cmvn",
                                global_cmvn=True)

fs = sidekit.FeaturesServer(feature_filename_structure="{dir}/{{}}.{ext}".
↪format(dir=feature_dir, ext=feature_extension),
                            dataset_list=["energy", "cep", "vad"],
                            mask="[0-12]",
                            feat_norm="cmvn",
                            keep_all_features=True,
                            delta=True,
                            double_delta=True,
                            rasta=True,
                            context=None)
```

Load the FeedForward Neural Network trained with **SIDEKIT** and Theano. (See tutorial on *Phonetically aware Neural Network for speaker recognition* training for more details).

```
FfNn = sidekit.FForwardNetwork.read("dnn/FFNN_1200sig-1200sig-80_1200sig_1200sig_final
↪")
```

Train your system

Train now the UBM with statistics computed from the Neural Network and one M-step form the EM algorithm. The fake-GMM UBM is then written to disk. Parameter `viterbi` is set to `False` to keep the entire output from the soft-max layer. Setting this parameter to `True`, the statistics will be turned to zero for all components except one.

```
ubm = FfNn.compute_ubm_dnn(ubm_list,
                           fs_dnn,
                           fs,
                           viterbi=False)
ubm.write("gmm/ubm_{}_{}.h5".format(distrib_nb, expe_id))
```

Create StatServers for the enrollment, test and background data and compute the statistics using the Neural Network:

```
# Compute enrollment data statistics
enroll_stat = sidekit.nnet.feed_forward.compute_stat_dnn(enroll_idmap,
                                                         "dnn/FFNN_1200sig-1200sig-80_
↪1200sig_1200sig_final",
                                                         fs_dnn,
                                                         fs,
                                                         num_thread=nbThread)
enroll_stat.write('data/stat_sre10_core-core_enroll_{}_{}.h5'.format(distrib_nb, expe_
↪id))


# Compute test data statistics
test_stat = sidekit.nnet.feed_forward.compute_stat_dnn(test_idmap,
                                                       "dnn/FFNN_1200sig-1200sig-80_
↪1200sig_1200sig_final",
                                                       fs_dnn,
                                                       fs,
                                                       num_thread=nbThread)
test_stat.write('data/stat_sre10_core-core_test_{}_{}.h5'.format(distrib_nb, expe_id))


# Merge TV and PLDA (remove duplicated sessions) and compute background data␣
↪statistics
back_idmap = plda_all_idmap.merge(tv_idmap)
back_stat = sidekit.nnet.feed_forward.compute_stat_dnn(back_idmap,
                                                       "dnn/FFNN_1200sig-1200sig-80_
↪1200sig_1200sig_final",
                                                       fs_dnn,
                                                       fs,
                                                       num_thread=nbThread)
back_stat.write('data/stat_back_{}_{}.h5'.format(distrib_nb, expe_id))
```

Note: that for UBM estimation and statistics computation, we keep using the version parallelized with multiprocessing as we haven't observe any issue with Numpy 1.11 so far. Please let us know in case you encounter any issue. A new version using MPI might be provided in future versions.

In order to train the Total Variability Matrix for i-vector extraction, report to the specific tutorial: *Train an i-vector extractor* for this part in order to chose between single processing, multiprocessing or MPI version.

Once this step has been completed, you will have a `FactorAnalyser` saved in HDF5 format and you can now extract your i-vectors for target models, training and test segments. To extract i-vectors, you can refer to the dedicated tutorial on *Extract your I-Vectors* using single or multiple nodes.

Run the tests

```
keys = []
for cond in range(9):
    keys.append(sidekit.Key('/lium/buster1/larcher/nist/sre10/core_core_{}_sre10_cond
↪{}_key.h5'.format("all", cond + 1)))

enroll_iv = sidekit.StatServer('data/iv_sre10_core-core_enroll_{}.h5'.format(distrib_
↪nb))
test_iv = sidekit.StatServer('data/iv_sre10_core-core_test_{}.h5'.format(distrib_nb))
plda_iv = sidekit.StatServer.read_subset('data/iv_plda_{}.h5'.format(distrib_nb),␣
↪plda_male_idmap)
```

Using Cosine similarity

A simple cosine scoring without any normalization of the i-vectors.

```
scores_cos = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv, test_ndx, wccn =␣
↪None)
```

A version where *i*-vectors are normalized using Within Class Covariance normalization (WCCN).

```
wccn = plda_iv.get_wccn_choleski_stat1()
scores_cos_wccn = sidekit.iv_scoring.cosine_scoring(enroll_iv, test_iv, test_ndx,␣
↪wccn=wccn)
```

The same with a Linear Discriminant Analysis performed first to reduce the dimension of *i*-vectors to 150 dimensions.

```
LDA = plda_iv.get_lda_matrix_stat1(150)

plda_iv_lda = copy.deepcopy(plda_iv)
enroll_iv_lda = copy.deepcopy(enroll_iv)
test_iv_lda = copy.deepcopy(test_iv)

plda_iv_lda.rotate_stat1(LDA)
enroll_iv_lda.rotate_stat1(LDA)
test_iv_lda.rotate_stat1(LDA)

scores_cos_lda = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda, test_iv_lda, test_
↪ndx, wccn=None)
```

And now combine LDA and WCCN:

```
wccn = plda_iv_lda.get_wccn_choleski_stat1()
scores_cos_lda_wcnn = sidekit.iv_scoring.cosine_scoring(enroll_iv_lda, test_iv_lda,␣
↪test_ndx, wccn=wccn)
```

Using Mahalanobis distance

If the scoring is 'mahalanobis', *i*-vectors are normalized using one iteration of the Eigen Factor Radial algorithm (equivalent to the so called length-normalization). Then scores are computed using a Mahalanobis distance.

```
meanEFR, CovEFR = plda_iv.estimate_spectral_norm_stat1(3)
```

```
plda_iv_efr1 = copy.deepcopy(plda_iv)
enroll_iv_efr1 = copy.deepcopy(enroll_iv)
test_iv_efr1 = copy.deepcopy(test_iv)

plda_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
enroll_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
test_iv_efr1.spectral_norm_stat1(meanEFR[:1], CovEFR[:1])
M1 = plda_iv_efr1.get_mahalanobis_matrix_stat1()
scores_mah_efr1 = sidekit.iv_scoring.mahalanobis_scoring(enroll_iv_efr1, test_iv_efr1,
↪ test_ndx, M1)
```

Using Two-covariance scoring

If the scoring is '2cov', two 2-covariance models are trained with and without *i*-vector normalization. The normalization applied consists of one iteration of Spherical Noramlization.

```
W = plda_iv.get_within_covariance_stat1()
B = plda_iv.get_between_covariance_stat1()
scores_2cov = sidekit.iv_scoring.two_covariance_scoring(enroll_iv, test_iv, test_ndx,␣
↪W, B)

meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')

plda_iv_sn1 = copy.deepcopy(plda_iv)
enroll_iv_sn1 = copy.deepcopy(enroll_iv)
test_iv_sn1 = copy.deepcopy(test_iv)

plda_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
enroll_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])
test_iv_sn1.spectral_norm_stat1(meanSN[:1], CovSN[:1])

W1 = plda_iv_sn1.get_within_covariance_stat1()
B1 = plda_iv_sn1.get_between_covariance_stat1()
scores_2cov_sn1 = sidekit.iv_scoring.two_covariance_scoring(enroll_iv_sn1, test_iv_
↪sn1, test_ndx, W1, B1)
```

Using Probabilistic Linear Discriminant Analysis

Normalize i-vector using Spherical Nuisance Normalization and compute scores using Probabilistic Linear Discriminant Analysis

```
meanSN, CovSN = plda_iv.estimate_spectral_norm_stat1(1, 'sphNorm')

plda_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
enroll_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])
test_iv.spectral_norm_stat1(meanSN[:1], CovSN[:1])

plda_mean, plda_F, plda_G, plda_H, plda_Sigma = plda_iv.factor_analysis(rank_f=plda_
↪rk,
                                                                        rank_g=0,
                                                                        rank_h=None,
                                                                        re_estimate_
↪residual=True,
                                                                        it_nb=(10,0,
↪0),
```

```
                                                                    min_div=True,
                                                                    ubm=None,
                                                                    batch_
↪size=1000,
                                                                    num_
↪thread=nbThread)

sidekit.sidekit_io.write_plda_hdf5((plda_mean, plda_F, plda_G, plda_Sigma), "data/
↪plda_model_tel_m_{}.h5".format(distrib_nb))

scores_plda = sidekit.iv_scoring.PLDA_scoring(enroll_iv, test_iv, test_ndx, plda_mean,
↪ plda_F, plda_G, plda_Sigma, full_model=False)
```

Plot the DET curves

In case you want to display the results of the experiments. First define the target prior, the parameters of the graphic window and the title of the plot.

```
# Set the prior following NIST-SRE 2010 settings
prior = sidekit.logit_effective_prior(0.001, 1, 1)
# Initialize the DET plot to 2010 settings
dp = sidekit.DetPlot(windowStyle='sre10', plotTitle='I-Vectors SRE 2010-ext male,␣
↪cond 5')
```

For each of the performed experiments, load the target and non-target scores for the condition 5 according to the key file.

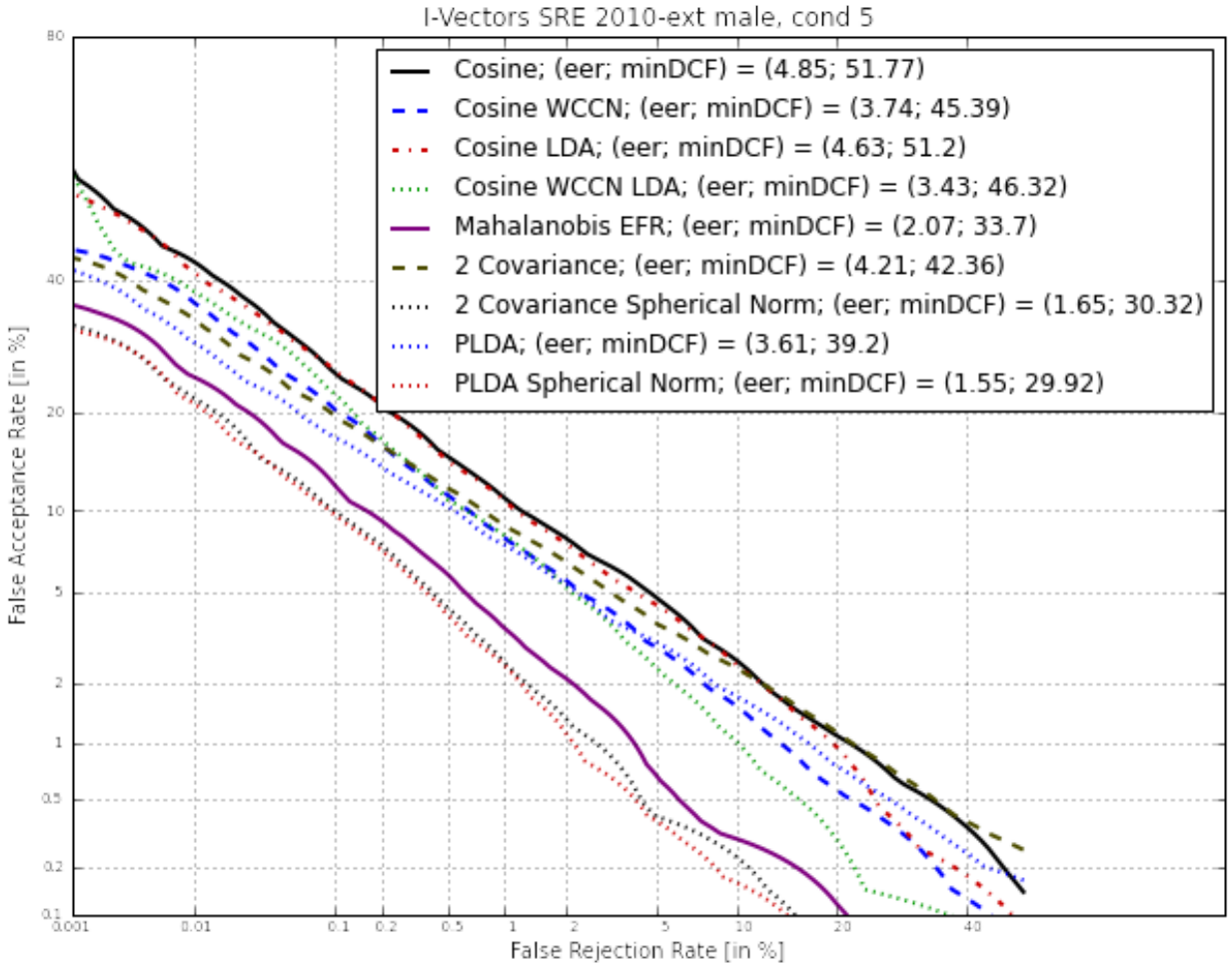
```
dp.set_system_from_scores(scores_cos, keys, sys_name='Cosine')
dp.set_system_from_scores(scores_cos_wccn, keys, sys_name='Cosine WCCN')
dp.set_system_from_scores(scores_cos_lda, keys, sys_name='Cosine LDA')
dp.set_system_from_scores(scores_cos_wccn_lda, keys, sys_name='Cosine WCCN LDA')

dp.set_system_from_scores(scores_mah_efr1, keys, sys_name='Mahalanobis EFR')

dp.set_system_from_scores(scores_2cov, keys, sys_name='2 Covariance')
dp.set_system_from_scores(scores_2cov_sn1, keys, sys_name='2 Covariance Spherical Norm
↪')

dp.set_system_from_scores(scores_plda, keys, sys_name='PLDA')
```

Create the window and plot:

```
dp.create_figure()
dp.plot_rocch_det(0)
dp.plot_rocch_det(1)
dp.plot_rocch_det(2)
dp.plot_rocch_det(3)
dp.plot_rocch_det(4)
dp.plot_rocch_det(5)
dp.plot_rocch_det(6)
dp.plot_rocch_det(7)
dp.plot_DR30_both(idx=0)
dp.plot_mindcf_point(prior, idx=0)
```

Depending of the data available, the following plot could be obtained at the end of this tutorial: (For this example, data used include NIST-SRE 04, 05, 06, 08, the SwitchBoard Part 2 phase 2 and 3 and Cellular part 2) Those results

are far from optimal as don't generalize on other conditions of NIST-SRE 2010. This system has been trained without any specific data selection and its purpose is only to give an idea of what you can obtain.



Bottleneck based i-vector system

Train an i-vector system using MPI

1.6 Additional materials

1.6.1 External links

A non-exhaustive list of links that are of interest to the authors.

- Tools for speaker recognition
  - ALIZE a speaker verification toolkit in C++
  - BOSARIS MATLAB code for calibration, fusion and evaluation of binary classifiers
  - Focal MATLAB code for evaluation, calibration and fusion of statistical pattern recognizers

  - HTK portable toolkit for building and manipulating hidden Markov models in C++
  - LIBSVM a library for Support Vector Machines in C
- Tools for Python programming
  - h5py HDF5 library for Python
  - Matplotlib 2D plotting library for Python
  - Numpy package for scientific computing in Python
  - Pandas library to manage and analyse data
  - Python official website
  - Scipy python-based-ecosystem of software for mathematics, science and engineering
  - Sphinx tool to create easy documentation (including this very one)
  - Spyder IDE for Python that allows line by line execution
  - VirtualEnv to create isolated Python environments
- Others programming tools
  - HDF5 a multi-platform library, and file format for storing and managing data
- Speech resources for speaker recognition
  - ELRA
  - LDC catalog
  - NIST-Speaker Recognition Evaluation main page
  - RSR2015 a database for text-dependent speaker verification *[Larcher14]*

More links can be find on the ISCA webpage.

1.6.2 References

1.6.3 Known errors and warnings

Warning due to the ctypes for multiprocessing...:

```
ctypeslib.py:408: RuntimeWarning: Item size computed from the PEP 3118 buffer format␣
↪string does not match the actual item size.
```

Pickle files created with version of Python below 2 and 3 might not be readable with Python 3 and 2.

It might happen that the PLDA does not converge, especially after *i*-vector normalization. This is due to the version of Lapack and Blas available on the machine and used by scipy. To solve this issue, install OpenBlas and Lapack and link to scipy. We srongly recommand not to use ALTAS.

1.6.4 Download lists for standard datasets

SORRY: This page will be hosted on another server and made available asap!

RSR2015

Dataset recorded in Singapore for text-dependent speaker recognition evaluation.

[Larcher14] Anthony Larcher, Kong Aik Lee, Bin Ma and Haizhou Li, "Text-dependent speaker verification: Classifiers, databases and RSR2015," in Speech Communication 60 (2014)

Condition 1:

- IdMap
- Ndx
- Keys

NIST-SRE 2010

core-core

- IdMap
- Ndx‘
- **Keys**
  - condition 1
  - condition 2
  - condition 3
  - condition 4
  - condition 5
  - condition 6
  - condition 7
  - condition 8
  - condition 9

coreX-coreX

- IdMap
- Ndx
- **Keys**
  - condition 1
  - condition 2
  - condition 3
  - condition 4
  - condition 5
  - condition 6
  - condition 7

- condition 8
- condition 9

RedDots

(To come soon)

1.7 Citation

When using **SIDEKIT** for research, please cite:

Anthony Larcher, Kong Aik Lee and Sylvain Meignier,
An extensible speaker identification SIDEKIT in Python,
in International Conference on Audio Speech and Signal Processing (ICASSP), 2016

1.8 Documentation

This documentation is available in PDF format `here`

1.9 Contacts and info

1.9.1 Contact

The SIDEKIT project aims at enabling exchanges, contacts and collaborations between academics and industrial actors in biometrics. You can get help, information and tips via two channels

- a developers / users mailing list
- the responsibles of the projects

Mailing List

By subscribing to the *Dev-sidekit@keaton.univ-lemans.fr* mailing list you will be able to post and receive information with othe users and developers of SIDEKIT.

To post to this list, send your email to:

dev-sidekit@keaton.univ-lemans.fr

General information about the mailing list is at:

http://keaton.univ-lemans.fr/cgi-bin/mailman/listinfo/dev-sidekit

If you ever want to unsubscribe or change your options (eg, switch to or from digest mode, change your password, etc.), visit your subscription page at:

http://keaton.univ-lemans.fr/cgi-bin/mailman/options/dev-sidekit/anthony.larcher%40univ-lemans.fr

You can also make such adjustments via email by sending a message to:

Dev-sidekit-request@keaton.univ-lemans.fr

with the word *help* in the subject or body (don't include the quotes), and you will get back a message with instructions.

You must know your password to change your options (including changing the password, itself) or to unsubscribe. It is:

Responsibles

Anthony Larcher

Associate Professor
LIUM, Universite du Mans (FRANCE)
anthony.larcher[at]univ-lemans.fr

Kong Aik Lee

Scientist
Human and Language Technology, Institute for Infocomm Research, A*Star (SINGAPORE)
kalee[at]i2r.a-star.edu.sg

Sylvain Meignier

Professor
LIUM, Universite du Mans (FRANCE)
sylvain.meignier[at]univ-lemans.fr

CHAPTER

TWO

SPONSORS

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Bimbot04] Frédéric Bimbot, Jean-François Bonastre, Corinne Fredouille, Guillaume Gravier, Ivan Magrin-Chagnolleau, Sylvain Meignier, Teva Merlin, Javier Ortega-García, Dijana Petrovska-Delacrétaz, and Douglas A. Reynolds. **A tutorial on text-independent speaker verification.** *EURASIP journal on applied signal processing* (2004): 430-451.

[Bousquet11] Pierre-Michel Bousquet, Anthony Larcher, Driss Matrouf, Jean-Francois Bonastre and Ma Bin, **Application of new i-vector conditioning algorithm and scoring method to NIST speaker recognition evaluation 2010**, *in NIST Speaker Recognition Evaluation Analysis Workshop*, 2011

[Chang11] Chih-Chung Chang and Chih-Jen Lin, **LIBSVM : a library for support vector machines**, *in ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[Cumani13] Sandro Cumani and Pietro Laface. **Fast and memory effective i-vector extraction using a factorized sub-space.**, INTERSPEECH, pages 1599-1603, 2013.

[Davis80] S.B. Davis and P. Mermelstein, **Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences**, *in IEEE Transaction on Acoustic,. Speech and Signal Processing*, TASSP-28 (4): 357-366, August 1980

[Garcia-Romero11] Daniel Garcia-Romero and C.Y. Espy-Wilson, **Analysis of i-vector length normalization in speaker recognition systems**, *in Annual Conference of the International Speech Communication Association (Interspeech)*, 2011, pp. 249-252

[Glembeck09] Ondrej Glembeck, Lukas Burget, Pavel Matejka, M. Karafiat and Patrick Kenny, **Simplification and optimization of I-Vector extraction**, *in IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, 2011, 4516-4519

[Larcher14] Anthony Larcher, Kong Aik Lee, Bin Ma and Haizhou Li, **Text-dependent speaker verification: Classifiers, databases and RSR2015.**, Speech Communication 60 (2014): 56-77.

[Lee13] Kong Aik Lee, Anthony Larcher, Chang Huai You, Bin Ma and Haizhou Li, **Multi-session PLDA scoring of i-vector for partially open-set speaker detection**, In INTERSPEECH, 3651-3655, 2013

PYTHON MODULE INDEX

INDEX

G

H

I

K

L

M

N

P

R

S

T

V

W