GONDOLA

Release v1.0 (r709)

Kayhan N. Batmanghelich **Andreas Schuh** **Aristeidis Sotiras**

September 07, 2012

Contents

This software implements Generative-Discriminative Basis Learning (GONDOLA), which is explained in detail in [TMI2012] and extended later on in [MICCAI2011]. Theoretical ideas are explained in these papers, but as a brief explanation, GONDOLA provides a generative method to reduce the dimensionality of medical images while using class labels. It produces basis vectors that are useful for classification and also clinically interpretable.

When provided with two sets of labeled images as input, the software outputs features saved in the Weka Attribute-Relation File Format (ARFF) [1] and a MATLAB data file. The program can also save basis vectors as NIfTI-1 images. Scripts are provided to find and build an optimal classifier using Weka [2]. The software can also be used for semi-supervised cases in which a number of subjects do not have class labels (for an example, please see [TMI2012]).

1 Download

The GONDOLA software is freely available under a BSD-style open source license that is compatible with the Open Source Definition by The Open Source Initiative and contains no restrictions on use of the software.

The full license text is included with the distribution package and available online.

Please fill this online form with the appropriate information to receive an email with the download links.

2 Changes

2.1 Release 1.0.0 (08/24/2012)

- First public release of the GONDOLA software.

[1] http://www.cs.waikato.ac.nz/ml/weka/arff.html
[2] http://www.cs.waikato.ac.nz/ml/weka/

3 Installation

An exhaustive list of minimum build dependencies, including the build tools along detailed step-by-step build, test, and installation instructions can be found in the BASIS how-to guide on software installation.

Please refer to this guide first if you are uncertain about the steps summarized here or if you have problems to build, test, or install the software on your system. If this guide does not help you to resolve the issue, please contact us at `sbia-software at uphs.upenn.edu`.

3.1 Prerequisites

See the *Required Packages* and *Optional Packages* tables for a summary of the software packages required by this software and those which are optionally made use of if available.

For instructions on how to build or install these prerequisites, please refer to the documentation of the respective software package and the linked how-to guides.

Required Packages

The following software has to be installed in order to build and run GONDOLA.

Package	Version	Description
BASIS	2.0	A meta-project developed at SBIA to standardize and simplify the software development.
CMake	2.8.4	The build of the software packages developed at SBIA is based on CMake, a cross-platform, open-source build tool. This tool can be used to configure the build system for various build tools which perform the actual build.
ITK	3.18	The ITK has to be build with the same compiler as the one used to build the MEX-files and MATLAB executable. Moreover, `CMAKE_CXX_FLAGS` and `CMAKE_C_FLAGS` has to include the `-fPIC` option for GCC. Although there have been major changes in ITK 4.0, this software should also work well with this version of the ITK as only the image IO classes are being used.
Jython	2.2	Jython is a Java implementation of the Python language interpreter. It enables the use of Java classes in scripts using the Python language and is in particular required by the command-line tools which determine the best parameters for each classifier and train them as these make use of the Weka Java API.
LIBLINEAR	1.8	LIBLINEAR is a fast linear SVM solver used in the GONDOLA optimization procedure. Make sure that you compile both LIBLINEAR and its MEX-files.
MATLAB	R2009b	An important part of this software has been written in MATLAB. Note that the Image Processing Toolbox is in particular required.
Python	2.6	The scripts for the cross-validation experiments are implemented in Python.
SBIAUtilities	1.0	The `sbiautilities.pyxel` Python module is used to read and write spreadsheets with the results of the cross-validation experiments in the CSV format.
Weka	3.7.3	Weka is a collection of machine learning algorithms for data mining tasks. It is used for the classification task.

Optional Packages

Package	Version	Description
GNU Compiler Collection	4.3	See the Supported and Compatible Compilers page of MATLAB and the MATLAB Compiler which versions of the GNU Compiler Collection (GCC) will work. It is important to use a version which is compatible. See the *How to build the GNU Compiler Collection* guide for help on the build of the GNU Compiler Collection from sources.
MOSEK	6.0	MOSEK is a large scale optimization software that is used to optimize one of the blocks of the objective function. Its use is, however, only recommended for those continue the development of GONDOLA or methods based on it. Otherwise, the default Spectral Projected Gradient (SPG) optimization algorithm (`csolver`) should be used which results further in a shorter execution time. Moreover, MOSEK is not freely available.
Oracle Grid Engine	any	Oracle Grid Engine software is a distributed resource management system that manages the distribution of users' workloads to available compute resources. It can be used to run several instances of the software in parallel. See the `gondola-sbia` script which can be used in place of `gondola` as command for the learning step for information on how a batch-queing system can be used to run the cross-validation experiments in parallel.

How to build the GNU Compiler Collection

The following instructions shall guide you through the build of the GNU Compiler Collection (GCC).

Note: These instructions were tested on **Ubuntu 12.04** and may differ from those required to build GCC for your operating system. Please refer also to the official GCC installation instructions.

For previous versions of the Ubuntu OS system, it is highly probable that the required version of the compiler, i.e., version 4.3 if you use MATLAB R2009b, is available in their repository. In this case, the installation may be performed in the straightforward manner through the `Synaptic Package Manager`. Make sure that the `libstdc++` and `g++` packages corresponding to the desired version are also installed.

In the most recent versions, among them *Ubuntu 12.04*, there is no package for version 4.3, however. Thus, GCC has to be build sources.

1. Install the build dependencies:

   ```
   sudo apt-get build-dep gcc-4.5
   ```

   The following may be required to avoid linking problems:

   ```
   export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu
   ```

   Alternatively, create a symbolic link such as:

   ```
   sudo ln -s /usr/lib/x86_64-linux-gnu /usr/lib64
   ```

2. Obtain the GCC source distribution package and extract the files:

   ```
   cd
   MIRROR=ftp://ftp.mirrorservice.org/sites/sourceware.org
   wget ${MIRROR}/pub/gcc/releases/gcc-4.3.4/gcc-4.3.4.tar.bz2
   tar -xjvf gcc-4.3.4.tar.bz2
   rm -f gcc-4.3.4.tar.bz2 (optional)
   ```

3. Make a separate build directory and change to it:

   ```
   mkdir gcc-4.3.4-build
   cd gcc-4.3.4-build
   ```

4. Configure the build and set the installation prefix to a different location:

   ```
   ../gcc-4.3.4/configure --prefix=/opt/gcc-4.3.4/
   ```

5. Once configured, build GCC using GNU Make:

   ```
   make [-j<#cores>]
   ```

6. To install the built executables, run:

   ```
   make install
   ```

7. Once the installation is done, you may remove the source and build files again:

   ```
   cd ..
   rm -rf gcc-4.3.4
   rm -rf gcc-4.3.4-build
   ```

Note: On Mac OS, use `curl -O` instead of `wget` in step 1 or simply download the sources using the browser of your choice instead.

Now, there should be two different versions of the GNU Compiler Collection being installed. When you build any of the other build dependencies of GONDOLA as well as GONDOLA itself, make sure that the correct version of the compiler is used. In CMake, toggle therefore to the advanced view and check the value of the `CMAKE_C_COMPILER` and `CMAKE_CXX_COMPILER` variables. Correct the paths to point to the right version if necessary. Note that after this change, CMake must reconfigure everything from scratch and all other settings will be lost. Therefore, set these values first. To avoid having to set these values after the first initialization of CMake, call `ccmake` as follows:

```
ccmake -DCMAKE_C_COMPILER:FILEPATH=/opt/gcc/4.3.4/gcc \
    -DCMAKE_CXX_COMPILER:FILEPATH=/opt/gcc-4.3.4/g++ \
    [options] <source dir>
```

Note: If you encounter the following error during the compilation of GONDOLA using GCC version 4.3.4, replace the file */opt/gcc-4.3.4/libjava/prims.cc* by the one available here. For more information, see the bug report.

```
./.libs/libgcj.so: undefined reference to '__cxa_call_unexpected'
```

3.2 Configure

This software is based on BASIS, a meta-project developed at SBIA to ease and standardize the software development, build, testing, and packaging. In particular, CMake, a cross-platform, open-source build system, is used for the configuration of the software build. For detailed instructions on how to build a software based on CMake, please refer to the BASIS how-to guide on software installation.

In summary, the steps to configure the so-called build tree are:

1. Extract source files:

   ```
   tar -xzf gondola-1.0.0-source.tar.gz
   ```

2. Create build directory:

   ```
   mkdir gondola-1.0.0-build
   ```

3. Change to build directory:

   ```
   cd gondola-1.0.0-build
   ```

4. Run CMake to configure the build tree:

   ```
   ccmake -DBASIS_DIR:PATH=/path/to/basis ../gondola-1.0.0-source
   ```

   - Press `c` to configure the build system and `e` to ignore warnings.
   - Set `CMAKE_INSTALL_PREFIX` and other CMake variables and options.
   - Continue pressing `c` until the option `g` is available.
   - Then press `g` to generate the configuration files for the selected build tool.

See the BASIS how-to guide on software installation for a documentation of the default configuration options provided by BASIS. The CMake options which are in particular of interest for the build of GONDOLA are documented *here*.

CMake Options

The following CMake options/variables can be configured:

`ITK_DIR <dir>`
Installation directory of ITK or the directory where the `ITKConfig.cmake` file can be found, respectively.

`LIBLINEAR_DIR <dir>`
Installation directory of LIBLINEAR.

`MATLAB_DIR <dir>`
Installation directory of MATLAB.

`MOSEK_DIR <dir>`
Installation directory of MOSEK.

`PythonModules_DIR <dir>`
Paths to NumPy and SciPy Python packages. If more these are not installed in the same directory, separate the two directories by `;`.

`SBIAUtilities_DIR <dir>`
Installation directory of SBIAUtilities or the directory where the `SBIAUtilitiesConfig.cmake` file can be found, respectively.

`Weka_DIR <dir>`
Installation directory of Weka.

Weka_PACKAGES_DIR <dir>

Since Weka 3.7.2, the functionality of Weka can be extended by additional packages, the installation of which is managed by the Weka Package Manager. Set this variable to the directory where the Weka packages are installed. By default, the directory is derived from the WEKA_HOME environment variable if set.

Advanced CMake Options

Advanced users may further be interested in the settings of the following options which in most cases are automatically derived from the non-advanced CMake options summarized above. To view these options in the CMake GUI, press the `t` key in `ccmake` (Unix) or check the `Show Advanced Values` box (Windows).

BASIS_COMPILE_MATLAB ON|OFF

Whether to compile the MATLAB sources using the MATLAB Compiler (mcc) if available. If set to `OFF`, the MATLAB source files are copied as part of the installation and a Bash script for the execution of `matlab` with the `-c` option is generated on Unix or a Windows NT Command script on Windows, respectively. This allows the convenient execution of the `gondola` MATLAB function from the command-line even without having a license for the build of a binary executable using the MATLAB Compiler. Each instance of the built executable will take up one MATLAB license, however. Moreover, the startup of the executable is longer every time, not only the first time it is launched as is the case for MCC compiled executables. It is therefore recommended to enable this option and to obtain a MATLAB Compiler license if possible. By default, this option is `ON`.

JYTHON_EXECUTABLE <file>

File path of the Jython interpreter executable (`jython`).

LIBLINEAR_libsvmread_MEX <file>

File path of the `libsvmread` MEX-file of the LIBLINEAR package.

LIBLINEAR_libsvmwrite_MEX <file>

File path of the `libsvmwrite` MEX-file of the LIBLINEAR package.

LIBLINEAR_predict_MEX <file>

File path of the `predict` MEX-file of the LIBLINEAR package.

LIBLINEAR_train_MEX <file>

File path of the `train` MEX-file of the LIBLINEAR package.

MOSEK_mosekopt_MEX <file>

File path of the `mosekopt` MEX-file of the LIBLINEAR package.

PYTHON_EXECUTABLE <file>

File path of the Python interpreter executable (`python`).

PythonModules_numpy_PATH <dir>

Directory which contains the NumPy Python package (`numpy` subdirectory).

PythonModules_scipy_PATH <dir>

Directory which contains the SciPy Python package (`scipy` subdirectory).

3.3 Build

After the configuration of the build tree, GONDOLA can be build using GNU Make:

```
make
```

3.4 Test

After the build of the software, optionally run the tests using the command:

```
make test
```

In case of failing tests, re-run the tests, but this time by executing CTest directly with the `-V` option to enable verbose output and redirect the output to a text file:

```
ctest -V >& gondola-test.log
```

And send the file `gondola-test.log` as attachment of the issue report to `sbia-software at uphs.upenn.edu`.

3.5 Install

The final installation copies the built files and additional data and documentation files to the installation directory specified using the `CMAKE_INSTALL_PREFIX` option during the configuration of the build tree:

```
make install
```

After the successful installation, the build directory can be removed again.

4 Documentation

A `PDF Version` of this documentation is available online and included with the distribution package.

4.1 Command-line Tools

The GONDOLA software package provides the following main command-line programs:

Program	Description
`gondola`	A MATLAB executable which implements the Generative-Discriminative Basis Learning and feature extraction.
`gondola-crossval`	This Python script is used to perform the different steps of the Cross-validation Experiments. It therefore either calls `gondola` with the appropriate command-line arguments or one of the other auxiliary executables for the search of the best parameters for each classifier and the training of these classifier.
`gondola-config`	This utility Python script sets up the cross-validation experiments and generates the default configuration files for both `gondola` and `gondola-crossval`.

Note: See the help output of each of these programs for further usage information. To display the help, simply run the commands either with no arguments or using the `-h` (`--help`) switch.

4.2 Generative-Discriminative Basis Learning

The Generative-Discriminative Basis Learning [MICCAI2011] [TMI2012] method is implemented by the `gondola` MATLAB package, where the main function is named `gondola.main`. For convenience, the MATLAB function `gondola` located at the same level as the MATLAB package is provided, which calls the `gondola.main` function. For the convenient execution of this MATLAB function as stand-alone program, different options are provided thanks to BASIS during the configuration of the build of the GONDOLA software. No matter which option was chosen, the command-line use of the `gondola` executable is always as in the following, where `gondola` refers to the installed `gondola` executable instead of the MATLAB function, but both can be called with the same arguments with the only difference being the syntax required by the shell versus the one required by MATLAB.

Note: The `gondola` executable is either a Bash script, a Windows NT Command, or a binary file created by the MATLAB Compiler. You can further use the `gondola` MATLAB function directly from within MATLAB. See the *Frequently Asked Questions* for more information on how to do this.

The `gondola` command implements the following subfunctions:

Function	Description
`learn`	Learn the basis vectors from training samples.
`extract`	Extract features given the learned basis vectors.
`show`	Save the learned basis vectors as NIfTI-1 images.
`help`	Show the help and exit/return.

Data Preparation

The input to `gondola` is a text file listing the names of the image files from which the basis vectors are learned, or from which the learned features shall be extracted from. Class labels for each sample have to be provided for the learning process. To maintain a common file format, class labes are also required for the feature extraction even if those are extracted from the test set of images. If the correct labels are unknown, use the value 0 as in case of the feature extraction the class labels are ignored anyways. Another input required for the feature extraction step is a text file with the subject IDs corresponding to each row of images in the image list file.

Note: All input images must be registered to a common template. The file format of the images can be any that is supported by the ITK.

The file format of the image list file is as follows:

```
<#subjects> <#images per subjects>
<dimx> <dimy> <dimz>
<root>

<subject 1, image 1> [<subject 1, image 2>...] <label>
<subject 2, image 1> [<subject 2, image 2>...] <label>
```

where `<#subjects>` is the number of subjects, `<#images per subjects>` is the number of images, e.g., modalities, tissue scans,..., per subject, `<dimx> <dimy> <dimz>` is the size of each image, and `<root>` is the common root directory of the image files used to make relative paths absolute. If the root directory itself is a relative path, it is considered to be relative to the location of the image list file. Thus, a root directory value of `.` corresponds to the directory where the image list file is located. The last column of each row after the file header specifies the class `<label>` which must be a positive number or 0 for unlabeled data. This class label is only considered during the learning of the basis vectors, but ignored otherwise. Yet, it has to be specified.

Note: Each image file path can be either absolute or relative. If it is an absolute path it is used as specified. Otherwise, the `<root>` directory is used to make the relative image file path absolute. While in most cases all images will be located within the same root directory, it is possible to have the image files stored in different directories by using absolute file paths.

The format of the corresponding subject ID list file is simply:

```
<subject ID 1>
<subject ID 2>
```

The paths to these input text files are specified as follows on the command line:

```
gondola [learn|extract] [options] --imagelistfile images.lst --idlistfile ids.lst
```

Configuration

The parameters of the Generative-Discriminative Basis Learning have to be specified in a configuration file such as:

```
algo:                    MultiViewXY
csolver:                 SPG
downsampleratio:         2
maxitr:                  300
numbasisvectors:         30
saveaftereachiteration: True
```

By default, this configuration file has to be named `gondola.cfg` and be located in the current working directory. The `--config` option can be used, however, to specify a different file path for the configuration file.

See the help of the `gondola` command for details on the parameters that can be set in the configuration file. If a parameter is not set in the configuration file, its default value is used. The `algo` and `numbasisvectors` settings are, however, required.

Hint: The configuration file is generated by `gondola-config` (see *Configure the Experiments*).

Output Files

The output file of the learning process is a MATLAB file (by default named `gondola.mat`) which stores the parameters that were used by the method in the `ConstVars` structure, the optimized variables `B`, `C`, and `w`, and the optimization history. This output file is input to the feature extraction which is performed by `gondola extract`. The extracted features are stored in the Attribute-Relation File Format (ARFF) of Weka. The learned basis vectors can be saved as NIfTI-1 images of data type `DT_FLOAT32`.

4.3 Cross-validation Experiments

To find out how well the results generalize, the learning of the basis vectors, extraction of the features, and the classification must typically be repeated several times. For the convenient execution of such cross-validation experiments using given training and testing data sets, the `gondola-crossval` command is provided by this package. The exact commands to execute as well as the names and paths of the input and output files for each step of the cross-validation experiments are specified by a configuration file which is input to this `gondola-crossval` command.

In the following, the commands that are executed to run a cross-validation experiment with 5 folds using the 20 example images included with the GONDOLA software are described for illustration.

Configure the Experiments

The first step is to create training and testing list files from the text file listing all available samples as well as the generation of the configuration files for both `gondola` as well as `gondola-crossval`:

```
mkdir gondola-cv-example && cd gondola-cv-example
gondola-config --algo MultiViewXY --csolver SPG --numbasisvectors 30 \
        --numfolds 5 ${EXAMPLE_DIR}/images.lst ${EXAMPLE_DIR}/ids.lst
```

where `${EXAMPLE_DIR}` refers to the `example/` directory of the GONDOLA distribution package.

This command will generate the configuration file for `gondola` (*ref*), named `gondola.cfg`, the configuration file for `gondola-crossval`, named `crossval.cfg`, and a separate directory for each fold, named after the ID of the fold, i.e., `1` for the first cross-validation fold, `2` for the second, and so on. Each fold directory contains the files `training.lst`, `trainids.lst`, `testing.lst`, and `testids.lst` for the corresponding disjoint training and testing sets.

See the generated `crossval.cfg` file and the help of `gondola-crossval` for details on the content and format of this configuration file. For most cases, the default settings should be used and only the parameters of `gondola` itself be modified, i.e., those specified by the `gondola.cfg` file.

The parameters used for the basis learning can be either changed by editing the generated `gondola.cfg` file or by supplying the desired parameter value as argument to `gondola-config`:

```
gondola-config [...] --lambda_gen 10 --maxitr 200
```

Run Experiment for each Fold

The first step of each experiment is the learning of the basis. Therefore, run the command:

```
gondola-crossval learn
```

in the directory where the `crossval.cfg` file is located or alternatively specify the configuration file using the `--config` option as in:

```
gondola-crossval learn --config /path/to/mycrossval.cfg
```

Tip: If the learning process was interrupted, the `--continue` option can be used to continue the learning beginning from the last saved iteration instead of starting all over from the scratch.

The output MATLAB files will be written to the directories specified by the configuration file.

All other steps are executed similarly, where only the name of the step to execute differs. Hence, to save the learned basis as NIfTI-1 images, to extract the features and train the classifiers, and to classify the training and testing sets, run the following commands in the given order:

```
gondola-crossval show
gondola-crossval extract
gondola-crossval search
gondola-crossval classify
```

A summary of the cross-validation experiments which is both printed to screen and further written to a text file (see `summaryfile` setting in the `crossval.cfg` configuration file) can be generated by running the:

```
gondola-crossval summarize
```

command.

Note: Using the `-v` option, the exact command that is executed by each of these steps is printed to screen. Moreover, the option `--simulate` can be used to only print those commands to screen without actually executing them. This can be used to copy only single commands in order to rerun these only.

Tip: Instead of running each of the named steps manually, you can run the command `gondola-crossval all`.

Warning: Note, however, that as convenient the `all` step is, it requires each step to be executed serially (the default). If you modified the commands specified in the `crossval.cfg` configuration that are executed by each step, for example, to execute one or more of the steps in parallel by submitting separate jobs to a batch-queuing system such as the Oracle Grid Engine, it is recommended to execute the steps one after another and not to use the `gondola-crossval all` command.

4.4 General Questions

If you have any questions regarding GONDOLA, please contact us at `sbia-software at uphs.upenn.edu`. Emails sent to this address will be forwarded to the right people at SBIA who will then reply to you and try to answer your questions.

Have also a look at the list of *Frequently Asked Questions*.

5 Frequently Asked Questions

5.1 How can I rank basis vectors according to importances in the classification?

The GONDOLA software by default does not rank the basis vectors. However, a Jython script named `wekaRankBasis` is included which can be found in the `lib/` directory of the installation (or `lib/gondola` depending on the selected installation scheme). Note that this script requires Weka >= 3.7 with additional packages installed. Check the *Installation* instructions for details. For usage information of this script, run it with the `-h` (`--help`) option.

5.2 How can I use GONDOLA inside of MATLAB?

You can use the `gondola` MATLAB package inside of MATLAB. Since there are some MEX-files to compile, first build the software to make sure that these are available. By setting the advanced CMake option `BASIS_COMPILE_MATLAB` to `OFF`, you can further request that the MATLAB sources are not being compiled using the MATLAB Compiler even if available, but copied to the installation tree. Then, add the paths to the MATLAB source files as well as the built (and installed) MEX-files to the MATLAB search path. Note that you have to add the parent directory containing the `+gondola` subdirectory to the path, not the `+gondola` package directory itself. In addition, you have to make sure that the LIBLINEAR MEX-files and those of MOSEK (if used) are in the search path as well. When the compilation of the MATLAB source files was skipped, i.e., the `BASIS_COMPILE_MATLAB` option was set to `OFF` during the configuration of the build system or if the MATLAB Compiler was not found, the installed `gondola` package contains a `startup.m` file. This script calls the `addpath` function of MATLAB to add the paths to the found MEX-files to the search path. Therefore, after adding the directory to the `gondola` package to the search path, run the command `gondola.startup` in MATLAB in order to set up the MATLAB search path. Now you can use the `gondola` function within MATLAB similar as the command-line executable.

5.3 How can I get probability of the classes instead of just class labels?

The Jython scripts coming with GONDOLA are capable of producing probability instead of class labels. However the default configuration files are generated in a way to produce just class labels. If you are interested to probabilities, you need to replace the `wekaClassifier` as the command used for the `classify` step in the *configuration file of the cross-validation experiments* by the `wekaClassifierWithProbability` command:

```
classify:  wekaClassifierWithProbability
                  --trainArff         "%(training.featuresfile)s"
                  --testArff          "%(testing.featuresfile)s"
                  --bestClassifier    "%(classifier)s"
                  --bestParam         "%(bestparams)s"
                  --extraParam        "%(extraparams)s"
                  --trainCSV          "%(training.resultfile)s"
                  --testCSV           "%(testing.resultfile)s"
                  --hdrTrain          "%(training.resultheader)s"
                  --hdrTest           "%(testing.resultheader)s"
```

5.4 How can I improve the running time?

There are a few ways to make GONDOLA run faster:

1. When compiling `gondola` using the MATLAB Compiler (MCC), remove the `-R -singleCompThread` from the advanced `BASIS_MCC_FLAGS` CMake variable (press ‘t’ in `ccmake`) before building GONDOLA. This will let the MATLAB Compiler Runtime (MCR) use multiple threads instead of only one.

2. Make sure that there are as few useless values (e.g. air which is usually zero background in brain images) in the image by appropriately cropping the images using the bounding box of the foreground regions.
3. Use a `downsamplingratio` of 2 as specified by the *configuration file for gondola*. It is not recommended to use higher values unless you are dealing with very high-resolution images or if the resolution of the basis vectors does not matter to you.
4. To avoid the startup time of launching MATLAB, ensure that the `gondola` source files are compiled using MCC. The caching done by MCC results in shorter startup times after the first execution.

5.5 How can I get better interpretability?

The basis vectors (i.e. columns of B) are between zero and one but because of the scaling of the data, they barely reach the upper bound of the feasible set (they barely have maximum value of 1). When you overlay the basis images written by the `gondola show` command on top of an image, set a threshold and make the values below that threshold transparent. You can easily do this using AFNI or MIPAV for medical images.

5.6 How can I choose the parameters?

The default parameters should work for most cases. However, if you want to have a good insight about how to choose the parameters, see the [TMI2012] paper.

5.7 Is multi-class classification possible with GONDOLA?

At the moment, GONDOLA does not fully support multi-class classification problems.

6 People

6.1 Advisor

- Christos Davatzikos
- Ben Taskat

6.2 Software Development

- Kayhan N. Batmanghelich,
- Andreas Schuh,
- Aristeidis Sotiras

6.3 Algorithm Development

- Kayhan N. Batmanghelich,
- Christos Davatzikos

References

[TMI2012] K. N. Batmanghelich, B. Taskar, and C. Davatzikos; Generative-Discriminative Basis Learning for Medical Imaging; IEEE Trans Med Imaging. 2012 Jan, 31(1):51-69.

[MICCAI2011] K. N. Batmanghelich, B. Taskar, D. Ye, and C. Davatzikos; Regularized Tensor Factorization for Multi-modality Medical Image Classification; MICCAI 2011, LNCS 6893, p17.