

Generative Discriminative Basis Learning for Medical Imaging: Software Manual

Nematollah Kayhan Batmanghelich, Andreas Schuh

June 12, 2012

Contents

1	Introduction	2
2	Package Contents	2
3	A Sample Case Study	3
4	Output Files	7
5	Important Notes	7
6	API Description	8

1 Introduction

This software implements Generative-Discriminative Basis Learning (**GONDOLA**), which is explained in detail in paper [3] and extended later on in paper [2]. 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 in **.arff** format (**Weka** format [4]) and a **.mat** file (**MATLAB** file). The program can also save basis vectors in **.nii** format (**NIFTI** format). Scripts are provided to find and build an optimal classifier using the **Weka** APIs. Prior knowledge about basis vectors can be provided via "group-wise" regularization that divides voxels of an image into groups - or, in practice, a segmentation image. Currently, only non-overlapping groups are supported. 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 [3]).

To install and use this software, you must first meet the requirements explained in the **INSTALL.txt** file. Note that all input images must first be registered to a common template, and that any image format supported by **ITK** [1] can be used. If you find this piece of software useful, please cite [3] and [2].

2 Package Contents

This package contains several binary and library files. Below, we have introduced the bare minimum binaries or scripts that you need to be familiar with in order to run the algorithm:

- **bin/gondola-config** : This script accepts a list of files, a list of corresponding IDs and a set of experiment parameters, then generates an appropriate folder structure and the required config files. Read the help file for more detail.
- **bin/gondola-ExperimentLauncher** : To find out how well how well we can generalize our results, the experiment must typically be repeated several times (i.e. learning the basis, extracting the features and performing classification). This script manages entire experiments (e.g. all folds of 10-fold cross validation) and provides correct arguments to **GONDOLA** in. It requires as input another **.config** file describing the experiment; please read the documentation of **launchExperiment** for more detail.

- `bin/sgeExecLearning_withMOSEK` : This script is used to run experiments in parallel on servers equipped with **Sun Grid Engine (SGE)**. It must be changed slightly to define the allocated memory, the number of CPU's required, and the e-mail address to use when reporting the experiment's progress.
- `bin/gondola` : This is the main executable, and provides the following functions:
 1. **Learning**: Perform optimized learning of the basis vectors (**B**) and other parameters.
 2. **Feature Extraction**: Project the images onto the basis vectors
 3. **Show**: Output each basis as an independent image file.

Please see the documentation of each file for more detail.

For function `bin/gondola`, the program requires a `.config` file specifying the algorithm parameters and outputs a `MATLAB` file (`.mat` file) containing the optimization history and the values of **B**, **C**, and **w**.

NOTE: It is not typically necessary to run `GONDOLA` directly. A more intuitive wrapper script (`bin/gondola-ExperimentLauncher`) is included that calls `GONDOLA` and provides the proper arguments.

3 A Sample Case Study

To start `GONDOLA`, first prepare an input ID list and an input image list. ID list must only have the a short snippet of naming that is unique to an image, eg. a prefix.

Example ID list:

```
$$> cat IDS.lst
R0186
R0178
R0179
R0139
R0014
R0009
R0037
R0185
```

The corresponding image list must include a list of images with the unique ID incorporated in the filename. The image list is then organized in the **COMPARE** list fashion.

COMPARE list format:

```

#[Subjects] \tab #[Modalities]
[x-dimension] \tab [y-dimension] \tab [z-dimension]
[Root path]
[Filepath relative to root SUBJECT 1 MODALITY 1] \tab [Filepath relative to root
SUBJECT 1 MODALITY 2] \tab [LABEL 1]
[Filepath relative to root SUBJECT 2 MODALITY 1] \tab [Filepath relative to root
SUBJECT 2 MODALITY 2] \tab [LABEL 2]
...

```

The reason for use of COMPARE list format is for backwards compatability of input with many other SBIA tools.

Example image list:

```

$$> cat FILES.lst
8 2
96 113 94
/sbia/sbiaprj/autism/TobaccoCAR/RAVENSProcessed/
R0186_RAVENSmap.WM.nii.gz      R0186_RAVENSmap.GM.nii.gz 1
R0178_RAVENSmap.WM.nii.gz      R0178_RAVENSmap.GM.nii.gz 1
R0179_RAVENSmap.WM.nii.gz      R0179_RAVENSmap.GM.nii.gz 1
R0139_RAVENSmap.WM.nii.gz      R0139_RAVENSmap.GM.nii.gz 1
R0014_RAVENSmap.WM.nii.gz      R0014_RAVENSmap.GM.nii.gz 2
R0009_RAVENSmap.WM.nii.gz      R0009_RAVENSmap.GM.nii.gz 2
R0037_RAVENSmap.WM.nii.gz      R0037_RAVENSmap.GM.nii.gz 0
R0185_RAVENSmap.WM.nii.gz      R0185_RAVENSmap.GM.nii.gz 0

```

Note that class labels are positive integers for known class labels, and 0 for unlabelled examples.

HINT: Class labels should be 1,2,3,... or 0.

These two lists then can be fed into `gondola-config` to generate an experiment configuration file.

To build the experiment configuration file, enter the parameters and the input ID and file list in the following fashion to `gondola-config`:

Example `gondola-config`:

```

$$> ./gondola-config -n 1708 -a MultiChannel_MultiView_BoxedSparsity_spg_Mosek
"lambda_gen:,lambda_disc:,lambda_const:,numBasisVectors: 10,0.1,0.2,30" -r
/sbia/comp_space/batmangn/Projects/gondola/results/ -m 5 -i ./IDS.lst -l
./FILES.lst

```

- `-n` :an experiment ID for the user to later identify the results related to this given experiement

- `-a` :algorithm type
- `-f` :parameters. These must be entered in the following fashion “`param1: ,param2: ,param3: value1,value2,value3`”
- `-r` :directory to put experiment files
- `-m` :number of folds for cross validation
- `-i` :ID list
- `-l` :File list

This command will create an experiment folder called `exp1708_5foldCV` in

`/sbia/comp_space/batmangn/Projects/gondola/results`

(specified in `-r`). Inside this folder, there will be 5 folders `1,2,3,4,5` and 2 files: `exp1708.config` and `exp1708_5foldCV.config`

The main config file `exp1708.config` will store the related parameters for this experiment. `lambda_gen` is the generative term constraint, `lambda_disc` is the discriminative term constraint, `lambda_const` is the sparsity constraint. `numBasisVectors` is the number of basis vectors that will be attempted to be found.

The cross-validation config file, `exp1708_5foldCV.config` will store parameters related to the partitioned cross validation folds: training and testing sub-lists paths, intermediate step result paths for the cross-validation, etc.

In a given cross-validation fold, subfolder, eg. `1` we will see:

```

$$> cd 1
$$> ls -l
ids_testing.lst
ids_training.lst
testing.lst
training.lst

```

To launch experiments, use `.gondola-ExperimentLauncher` with the generated configuration file, specifying the step to be performed.

In the following, we provide an example which runs `gondola-ExperimentLauncher` step-by-step (`-s STEP` to be performed):

STEP 1: Learn the best basis vectors

```

$$> ./gondola-ExperimentLauncher -c
/sbia/comp_space/batmangn/Projects/gondola/results/exp1708_5foldCV/
exp1708_5foldCV.config -s 1

```

The learned basis vectors will be stored in each cross validation folder: `exp1708_CV1_5.mat`. Also, the log file `exp1708_CV1_5.log` will be updated.

STEP 2: Extract features and save into an `.arff` file

```
$$> ./gondola-ExperimentLauncher -c  
/sbia/comp_space/batmangn/Projects/gondola/results/exp1708_5foldCV/  
exp1708_5foldCV.config -s 2
```

Extracted features using the previously learned basis elements will be saved in each cross validation folder: `training.arff` and `testing.arff`.

STEP 3: Find the best parameters and save into `.csv` files

```
$$> ./gondola-ExperimentLauncher -c  
/sbia/comp_space/batmangn/Projects/gondola/results/exp1708_5foldCV/  
exp1708_5foldCV.config -s 3
```

Training cross validation will be employed to do a parameter search for classifiers, (Random Forest, Simple logistic classifier, SMO, Logistic, and Optimal Bayes). The parameters will be saved in each cross validation folder: `bestParam.csv`

STEP 4: Train the classifier with best parameters and apply to the training and testing data

```
$$> ./gondola-ExperimentLauncher -c  
/sbia/comp_space/batmangn/Projects/gondola/results/exp1708_5foldCV/  
exp1708_5foldCV.config -s 4
```

The classifiers will be trained using the best parameters, and then the test data will be classified. Results are saved in each cross validation folder:

- `train-Logistic-Results.csv`
- `train-SMO-Results.csv`
- `train-Simple Logistic-Results.csv`
- `train-Bayesian-Results.csv`
- `train-Random Forest-Results.csv`
- `test-Logistic-Results.csv`
- `test-SMO-Results.csv`
- `test-Simple Logistic-Results.csv`

- test-Bayesian-Results.csv
- test-Random Forest-Results.csv

STEP 5: Summarize and make a report

```
$$> ./gondola-ExperimentLauncher -c
/sbia/comp_space/batmangn/Projects/gondola/results/exp1708_5foldCV/
exp1708_5foldCV.config -s 5
```

This will generate summaries of all the cross-validation folds in the main experiment folder. The summary of the experiment for each classifier is created in a text file corresponding to each classifier:

- 1708-summary_Bayesian.txt
- 1708-summary_Logistic.txt
- 1708-summary_Random Forest.txt
- 1708-summary_SMO.txt
- 1708-summary_Simple Logistic.txt

4 Output Files

- .mat file : The learned basis elements, **B**, for the input dataset is saved here along with other parameters used in the algorithm.
- .arff files : The extracted features corresponding to the learned basis elements are stored here.
- bestParam.csv file : Best classifier parameters for, Bayesian, Logistic, Random Forest, SMO, and Simple Logistic classifiers are learned here using cross validation within training set.
- classifier results.csv files : For the above classifiers, training and testing classification results are displayed.

5 Important Notes

6 API Description

References

- [1] ITK., <http://www.itk.org>
- [2] Batmanghelich, N., Dong, A., Taskar, B., Davatzikos, C.: Regularized Tensor Factorization for Multi-Modality Medical Image Classification. In: Fichtinger, G., Martel, A.L., Peters, T.M. (eds.) MICCAI (3). Lecture Notes in Computer Science, vol. 6893, pp. 17–24. Springer (2011)
- [3] Batmanghelich, N.K., Taskar, B., Davatzikos, C.: Generative-discriminative basis learning for medical imaging. IEEE Trans Med Imaging 31(1), 51–69 (Jan 2012), <http://www.ncbi.nlm.nih.gov/pubmed/21791408>
- [4] Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, Amsterdam, 3. edn. (2011), <http://www.sciencedirect.com/science/book/9780123748560>