Leaspy

Release 1.1.2

unknown

Apr 13, 2021

GETTING STARTED

1 Installation & testing **3**
- 1.1 Dependencies . . . 3
- 1.2 User installation . . . 3
- 1.3 Testing . . . 4

2 Leaspy in a nutshell **5**
- 2.1 Comprehensive example . . . 5
- 2.2 Using my own data . . . 7
- 2.3 Going further . . . 8

3 API Documentation **9**
- 3.1 `leaspy.api`: **Main API** . . . 9
- 3.2 `leaspy.algo`: **Algorithms** . . . 9
- 3.3 `leaspy.dataset`: **Datasets** . . . 10
- 3.4 `leaspy.io`: **Inputs / Outputs** . . . 11
- 3.5 `leaspy.models`: **Models** . . . 11

4 User guide **13**
- 4.1 Mathematical aspects . . . 13
- 4.2 Leaspy's tutorial . . . 13

5 LEArning Spatiotemporal Patterns in Python **15**
- 5.1 Description . . . 15
- 5.2 Getting started . . . 16
- 5.3 User Guide . . . 16
- 5.4 API Documentation . . . 16
- 5.5 License . . . 16
- 5.6 Further information . . . 16

Index **17**


leaspy

CHAPTER

ONE

INSTALLATION & TESTING

1.1 Dependencies

leaspy requires:

- Python (>=3.6)
- numpy (>=1.16.6)
- scipy (>=1.5.4)
- scikit-learn (>=0.21.3, <0.24)
- pandas (>=1.0.5)
- torch (>=1.2.0, <1.7)
- joblib (>=0.13.2)
- matplotlib>=3.0.3
- statsmodels (>=0.12.1)

1.2 User installation

1. (Optional) Create a dedicated *conda environment*:

```
conda create --name leaspy python=3.7
conda activate leaspy
```

2. Download *leaspy* with *pip*:

```
pip install leaspy
```

1.3 Testing

After installation, you can run the examples in *Leaspy in a nutshell* and in *the Leaspy API*. To do so, in your *leaspy environment*, you can download `ipykernel` to use *leaspy* with *jupyter*:

```
conda install -c anaconda ipykernel
python -m ipykernel install --user --name=leaspy
```

Now, you can open *jupyter lab* or *jupyter notebook* and select the *leaspy kernel*.

CHAPTER

TWO

LEASPY IN A NUTSHELL

2.1 Comprehensive example

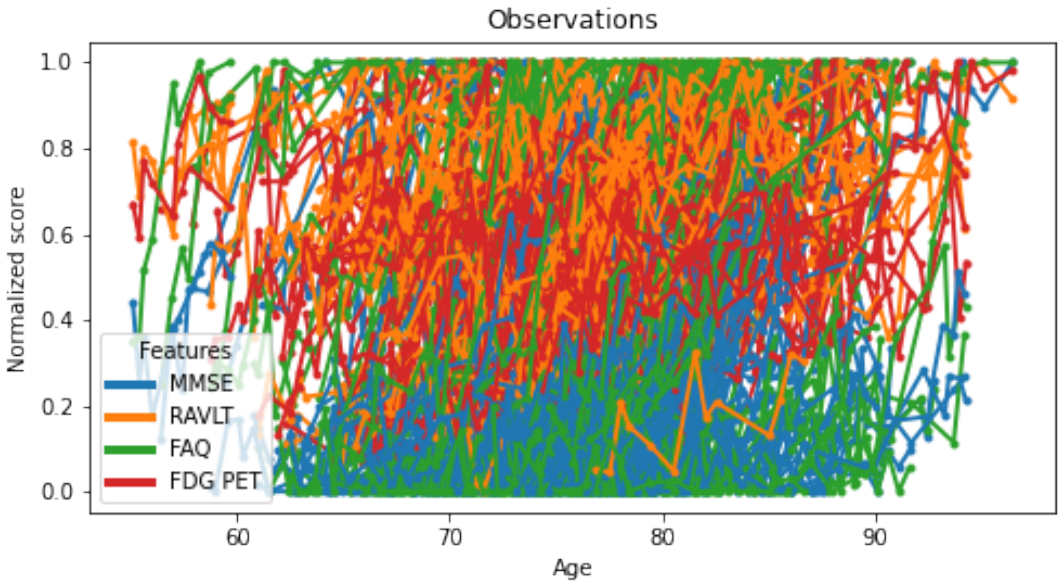
We load some synthetic data from the *leaspy.datasets* module, encapsulate them in the main *leaspy Data container*, then we plot them with the main API *Leaspy*.

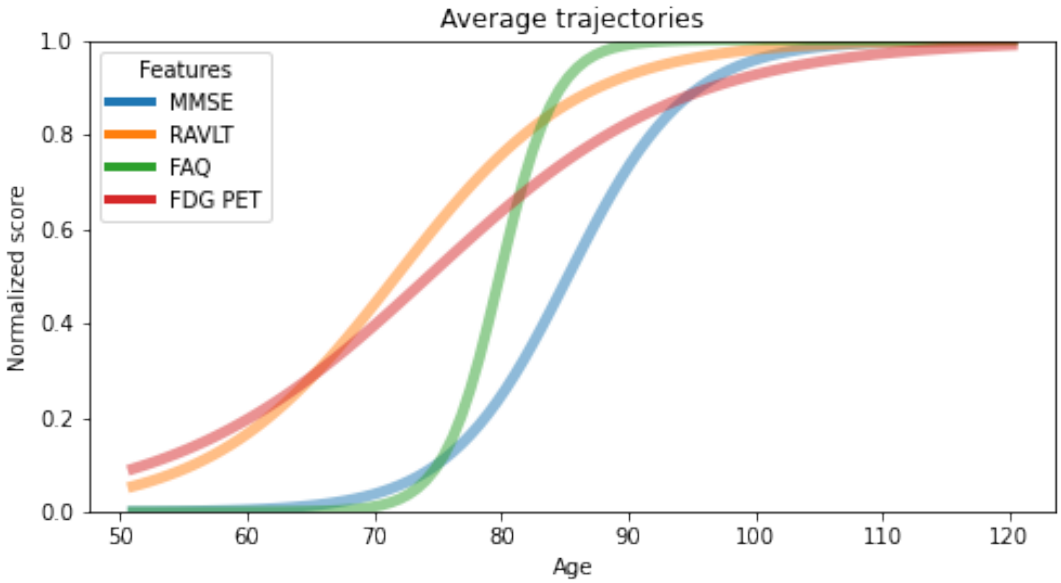
```
>>> from leaspy import AlgorithmSettings, Data, Leaspy
>>> from leaspy.datasets import Loader
>>> alzheimer_df = Loader.load_dataset('alzheimer-multivariate')
>>> print(alzheimer_df.columns)
Index(['E-Cog Subject', 'E-Cog Study-partner', 'MMSE', 'RAVLT', 'FAQ',
   'FDG PET', 'Hippocampus volume ratio'],
  dtype='object')
>>> alzheimer_df = alzheimer_df[['MMSE', 'RAVLT', 'FAQ', 'FDG PET']]
>>> print(alzheimer_df.head())
                     MMSE     RAVLT       FAQ   FDG PET
ID     TIME
GS-001 73.973183  0.111998  0.510524  0.178827  0.454605
       74.573181  0.029991  0.749223  0.181327  0.450064
       75.173180  0.121922  0.779680  0.026179  0.662006
       75.773186  0.092102  0.649391  0.156153  0.585949
       75.973183  0.203874  0.612311  0.320484  0.634809
>>> data = Data.from_dataframe(alzheimer_df)
>>> leaspy_logistic = Leaspy('logistic')
>>> ax = leaspy_logistic.plotting.patient_observations(data)
```

Not so engaging, right? With *leaspy*, we can derive the group average trajectory of this population. We use the *Leaspy.fit* method by providing it the settings for the MCMC-SAEM algorithm. Then, we plot the group average trajectory:

```
>>> model_settings = AlgorithmSettings('mcmc_saem', seed=0, progress_
↪bar=True)
>>> leaspy_logistic.fit(data, model_settings)
==> Setting seed to 0
|##################################################|   10000/10000 iterations
The standard deviation of the noise at the end of the calibration is:
0.0718
Calibration took: 5min 55s
>>> ax2 = leaspy_logistic.plotting.average_trajectory()
```

We can also derive the individual trajectory of each subject. To do this, we use the *Leaspy.personalize* method, again by providing the proper settings. Then we plot both, the first subjects observations and trajectories:


Observations
Normalized score
Age
Features
MMSE
RAVLT
FAQ
FDG PET
0.0
0.2
0.4
0.6
0.8
1.0
60
70
80
90


Average trajectories
Normalized score
Age
Features
MMSE
RAVLT
FAQ
FDG PET
0.0
0.2
0.4
0.6
0.8
1.0
50
60
70
80
90
100
110
120

```
>>> personalize_settings = AlgorithmSettings('scipy_minimize', progress_
↪bar=True, \
use_jacobian=True, seed=0)
>>> individual_parameters = leaspy_logistic.personalize(data, personalize_
↪settings)
 ==> Setting seed to 0
|################################################|   200/200 subjects
The standard deviation of the noise at the end of the personalization is:
0.0686
Personalization scipy_minimize took: 11s
>>> ax = leaspy_logistic.plotting.patient_trajectories(data, individual_
↪parameters, 'GS-001')
>>> leaspy_logistic.plotting.patient_observations(data, 'GS-001', ax=ax)
>>> import matplotlib.pyplot as plt
>>> plt.legend(loc='upper left', title='Features', bbox_to_anchor=(1.05, 1))
>>> plt.title('Subject GS-001 - observations & trajectories')
```

2.2 Using my own data

2.2.1 Data format

Leaspy use its own data container. To use it properly, you need to provide a *.csv* file or a *pandas.DataFrame* in the right format. Let's have a look on the data used in the previous example:

```
>>> print(alzheimer_df.head())
                     MMSE     RAVLT       FAQ   FDG PET
ID     TIME
GS-001 73.973183  0.111998  0.510524  0.178827  0.454605
       74.573181  0.029991  0.749223  0.181327  0.450064
       75.173180  0.121922  0.779680  0.026179  0.662006
       75.773186  0.092102  0.649391  0.156153  0.585949
       75.973183  0.203874  0.612311  0.320484  0.634809
```

You **MUST** have *ID* and *TIME*, either in index or in the columns. The other columns must be the observation variables, or *features*. In this fashion, you have one column per *feature* and one line per *visit*.

2.2.2 Data scale & constraints

Leaspy use *linear* and *logisitic* models. The features **MUST** be increasing with time. For the *logisitic* model, you need to rescale your data between 0 and 1.

2.2.3 Missing data

Leaspy automatically handle missing data. However, they **MUST** be encoded as `numpy.nan` in your *pandas.DataFrame*.

2.3 Going further

You can check the *User guide* and the *full API documentation*.

CHAPTER

THREE

API DOCUMENTATION

Full API documentation of the *Leaspy* Python package.

3.1 `leaspy.api`: Main API

The main class, from which you can instantiate and calibrate a model, personalize it to a given set a subjects, estimate trajectories and simulate synthetic data.

`Leaspy`(model_name, **kwargs)	Main API used to fit models, run algorithms and simulations.

3.2 `leaspy.algo`: Algorithms

Contains all algorithms used in the package.

`abstract_algo.AbstractAlgo`()	Abstract class containing common methods for all algorithm classes.
`algo_factory.AlgoFactory`()	Return the wanted algorithm given its name.

3.2.1 `leaspy.algo.fit`: Fit algorithms

Algorithms used to calibrate a model.

`abstract_fit_algo.AbstractFitAlgo`()	Abstract class containing common method for all *fit* algorithm classes.
`abstract_mcmc.AbstractFitMCMC`(settings)	Abstract class containing common method for all *fit* algorithm classes based on *Monte-Carlo Markov Chains* (MCMC).
`tensor_mcmcsaem.TensorMCMCSAEM`(settings)	Main algorithm for MCMC-SAEM.

3.2.2 `leaspy.algo.personalize`: Personalization algorithms

Algorithms used to personalize a model to a given set of subjects.

`abstract_personalize_algo.AbstractPersonalizeAlgo(...)`	Abstract class for *personalize* algorithm.
`scipy_minimize.ScipyMinimize`(settings)	Gradient descent based algorithm to compute individual parameters, *i.e.* personalize a model to a given set of subjects.

3.2.3 `leaspy.algo.samplers`: Samplers

Samplers used by the algorithms.

`abstract_sampler.AbstractSampler`(info, ...)	Abstract sampler class.
`gibbs_sampler.GibbsSampler`(info, n_patients)	Gibbs sampler class.
`hmc_sampler.HMCSampler`(info, n_patients, eps)	Hamiltonian Monte Carlo sampler class.

3.2.4 `leaspy.algo.simulate`: Simulation algorithms

Algorithm to simulate synthetic observations and individual parameters.

`simulate.SimulationAlgorithm`(settings)	To simulate new data given existing one by learning the individual parameters joined distribution.

3.2.5 `leaspy.algo.others`: Other algorithms

Reference algorithms to use with reference models (for benchmarks).

`constant_prediction_algo.ConstantPredictionAlgorithm(...)`	Personalization algorithm associated to `ConstantModel`
`lme_fit.LMEFitAlgorithm`(settings)	Calibration algorithm associated to `LMEModel`
`lme_personalize.LMEPersonalizeAlgorithm`(settings)	Personalization algorithm associated to `LMEModel`

3.3 `leaspy.dataset`: Datasets

Give access to some synthetic longitudinal observations mimicking cohort of subjects with neurodegenerative disorders, as well as calibrated models and computed individual parameters.

`loader.Loader`()	Contains static methods to load synthetic longitudinal dataset, calibrated `Leaspy` instances & `IndividualParameters`.

3.4 leaspy.io: Inputs / Outputs

Containers class objects used as input / ouputs in the *Leaspy* package.

3.4.1 leaspy.io.data: Data containers

data.Data()	Main data container, initialized from a *csv file* or a pandas.DataFrame.
dataset.Dataset(data[, model, algo])	Data container based on torch.Tensor, used to run algorithms.

3.4.2 leaspy.io.outputs: Outputs class objects

individual_parameters.IndividualParameters()	Data container for individual parameters, contains IDs, timepoints and observations values.

3.4.3 leaspy.io.realizations: Realizations class objects

realization.Realization(name, shape, ...)	Contains the realization of a given parameter.
collection_realization.CollectionRealization()	Realizations of population and individual parameters.

3.4.4 leaspy.io.settings: Settings class objects

model_settings.ModelSettings(...)	Used in Leaspy.load() to create a Leaspy class object from a *json* file.
algorithm_settings.AlgorithmSettings(name, ...)	Used to set the algorithms' settings.
outputs_settings.OutputsSettings(settings)	Used to create the *logs* folder to monitor the convergence of the calibration algorithm.

3.5 leaspy.models: Models

Available models in *Leaspy*.

model_factory.ModelFactory()	Return the wanted model given its name.
abstract_model.AbstractModel(name)	Contains the common attributes & methods of the different models.
abstract_multivariate_model.AbstractMultivariateModel(...)	Contains the common attributes & methods of the multivariate models.
multivariate_model.MultivariateModel(name, ...)	Manifold model for multiple variables of interest (logistic or linear formulation).

continues on next page

Table 13 – continued from previous page

multivariate_parallel_model.MultivariateParallelModel(...)	Logistic model for multiple variables of interest, imposing same average evolution pace for all variables (logistic curves are only time-shifted).
univariate_model.UnivariateModel(name, **kwargs)	Univariate (logistic or linear) model for a single variable of interest.
lme_model.LMEModel(name)	LMEModel is a benchmark model that fits and personalize a linear mixed-effects model
constant_model.ConstantModel(name)	*ConstantModel* is a benchmark model that predicts constant values no matter of the patient's ages.

3.5.1 leaspy.models.utils.attributes: Models' attributes

Attributes used by the models.

attributes_factory.AttributesFactory()	Return an *Attributes* class object based on the given parameters.
abstract_attributes.AbstractAttributes(name)	Abstract base class for attributes of models.
abstract_manifold_model_attributes.AbstractManifoldModelAttributes(...)	Abstract base class for attributes of leaspy manifold models.
linear_attributes.LinearAttributes(name, ...)	Attributes of leaspy linear models.
logistic_attributes.LogisticAttributes(name, ...)	Attributes of leaspy logistic models.
logistic_parallel_attributes.LogisticParallelAttributes(...)	Attributes of leaspy logistic parallel models.

3.5.2 leaspy.models.utils.initialization: Initialization methods

model_initialization.initialize_parameters(...)	Initialize the model's group parameters given its name & the scores of all subjects.

CHAPTER

FOUR

USER GUIDE

TODO

4.1 Mathematical aspects

4.1.1 Introduction

TODO

4.1.2 Mathematical formulation

TODO

4.1.3 Riemanian framework

TODO

4.1.4 Missing data

4.2 Leaspy's tutorial

4.2.1 What do I need?

TODO

4.2.2 Derive the population parameters

TODO

4.2.3 Derive the individual parameters

TODO

4.2.4 Cofactor analysis

TODO

4.2.5 What about missing values?

TODO

4.2.6 Predictions

TODO

4.2.7 Simulations

TODO

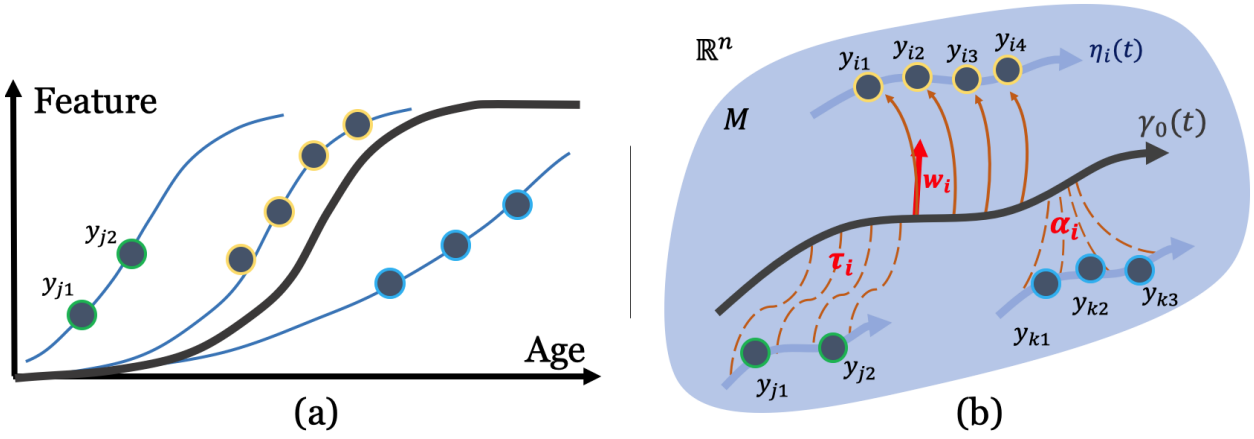
CHAPTER

FIVE

LEARNING SPATIOTEMPORAL PATTERNS IN PYTHON

5.1 Description

Leaspy is a software package for the statistical analysis of **longitudinal data**, particularly **medical** data that comes in a form of **repeated observations** of patients at different time-points.



Considering these series of short-term data, the software aims at :

- Recombining them to reconstruct the long-term spatio-temporal trajectory of evolution
- Positioning each patient observations relatively to the group-average timeline, in term of both temporal differences (time shift and acceleration factor) and spatial differences (diffent sequences of events, spatial pattern of progression, ...)
- Quantifying impact of cofactors (gender, genetic mutation, environmental factors, ...) on the evolution of the signal
- Imputing missing values
- Predicting future observations
- Simulating virtual patients to unbias the initial cohort or mimic its characteristics

The software package can be used with scalar multivariate data whose progression can be modeled by a logistic shape, an exponential decay or a linear progression. The simplest type of data handled by the software are scalar data: they correspond to one (univariate) or multiple (multivariate) measurement(s) per patient observation. This includes, for instance, clinical scores, cognitive assessments, physiological measurements (e.g. blood markers, radioactive markers) but also imaging-derived data that are rescaled, for instance, between 0 and 1 to describe a logistic progression.

5.2 Getting started

Information to install, test, and contribute to the package.

5.3 User Guide

The main documentation. This contains an in-depth description of all algorithms and how to apply them.

5.4 API Documentation

The exact API of all functions and classes, as given in the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

5.5 License

5.6 Further information

More detailed explanations about the models themselves and about the estimation procedure can be found in the following articles :

- **Mathematical framework**: *A Bayesian mixed-effects model to learn trajectories of changes from repeated manifold-valued observations.* Jean-Baptiste Schiratti, Stéphanie Allassonnière, Olivier Colliot, and Stanley Durrleman. The Journal of Machine Learning Research, 18:1–33, December 2017. Open Access
- **Application to imaging data**: *Statistical learning of spatiotemporal patterns from longitudinal manifold-valued networks.* I. Koval, J.-B. Schiratti, A. Routier, M. Bacci, O. Colliot, S. Allassonnière and S. Durrleman. MICCAI, September 2017. Open Access
- **Application to imaging data**: *Spatiotemporal Propagation of the Cortical Atrophy: Population and Individual Patterns.* Igor Koval, Jean-Baptiste Schiratti, Alexandre Routier, Michael Bacci, Olivier Colliot, Stéphanie Allassonnière, and Stanley Durrleman. Front Neurol. 2018 May 4;9:235. Open Access
- **Application to data with missing values**: *Learning disease progression models with longitudinal data and missing values*. R. Couronne, M. Vidailhet, JC. Corvol, S. Lehéricy, S. Durrleman. ISBI, April 2019. Open Access
- **Intensive application for Alzheimer's Disease progression**: *AD Course Map charts Alzheimer's disease progression*, I. Koval, A. Bone, M. Louis, S. Bottani, A. Marcoux, J. Samper-Gonzalez, N. Burgos, B. Charlier, A. Bertrand, S. Epelbaum, O. Colliot, S. Allassonniere & S. Durrleman, Under review Open Access
- www.digital-brain.org : Website related to the application of the model for Alzheimer's disease.

INDEX

L

leaspy.algo
    module, 9
leaspy.api
    module, 9

M

module
    leaspy.algo, 9
    leaspy.api, 9