GPTwoSample Documentation

Release 0.1.7a

Max Zwießele, Oliver Stegle

April 07, 2013

CONTENTS

1 Command line Tool **3**
- 1.1 Installing the package . . . 3
- 1.2 Example usage . . . 3
- 1.3 Further Details . . . 4

2 Developer **9**
- 2.1 Package for using GPTwoSample . . . 9
- 2.2 GPTwoSample plot . . . 14
- 2.3 Package for data handling . . . 15

Bibliography **19**

Python Module Index **21**

gptwosample.py is a tool to run two-sample tests on time series differential gene expression experiments. It can either be called form the command line or using the interactive Python Modules. Here, we will explore usage from the command line, for detailed description, refers to the other sections.

CHAPTER

ONE

COMMAND LINE TOOL

`gptwosample` is designed to compare two gene expression time series experiments, including the possibility for several replicates in each experiment. The test fits latent functions to both time series, comparing the assumption that all data originated from a single latent process (*common model fit*) or a two distinct separate time series (*individual model fit*). See [Stegle2010] for details. The Raw gene expression data can be supplied via simple CSV files, one for each experiments. The data format is flexible, permits missing values and non-synchronized time points. Fur full details please see *Data format*.

1.1 Installing the package

To install `gptwosample` run:

```
python setup install
```

from `gptwosample` directory.

Try printing the full help of the script using:

```
python gptwosample --help
```

To run optional package tests before installing run:

```
python setup test
```

1.2 Example usage

Once the data has been prepared, GPTwoSample can be executed from the unix command line.

General command line parameters of interest include:

```
--help
-v
```

Also, to create plots of the fitted functions, which creates verbose plots illustrating the fit for every tested gene:

```
-p
```

For example, to run the basic `gptwosample` model on the tutorial datasets provided alongside the package including verbose output and plots, run:

```
python gptwosample -v -p -t -o ./examplerun/ examples/ToyCondition1.csv examples/ToyCondition2.csv
```

This stores results in `./examplerun/`. Quantitative readouts summarizing the differential expression stores ares provided in "results.csv" (see *Result structure* for format). Plots in will be saved in a subfolder `./examplerun/plots/`.

1.3 Further Details

1.3.1 Parameter options

Calling signature:

```
gptwosample [-h] [-o DIR] [-t] [-c N] [-p] [-v] [--version] [--backend [PDF,...]]  FILE FILE
```

where:

```
FILE                  treatment/control files to compare against each other
-h, --help            show this help message and exit
-o DIR, --out DIR     set output dir [default: ./twosample_out/]
-t, --timeshift       account for timeshifts in data [default: False]
-c N, --confounder N  account for N confounders in data [default: 0]
-p, --plot            plot data into outdir/plots? [default: False]
-v, --verbose         set verbosity level [default: 0]
--version             show program's version number and exit
--backend [PDF,...]   matplotlib backend - see matplotlib.use(backend)
```

1.3.2 Data format

The format of the two `.csv` files (`FILE FILE` in usage) is as follows:

arbitrary	**x1**	**...**	**xl**
Gene ID 1	y1 replicate 1	...	yl replicate 1
...	...	...	...
Gene ID 1	y1 replicate k1	...	yl replicate k1
...			
Gene ID n	y1 replicate 1	...	yl replicate 1
...	...	...	...
Gene ID n	y1 replicate kn	...	yl replicate kn

where all values, which cannot be translated by `float()` will be treated as missing values in the model.

1.3.3 Accounting for confounding factors

We detect common confounding factors using probabilistic principal component analysis modeled by gaussian process latent variable models (GPLVM) [Lawrence2004]. This probabilistic approach to detect low dimensional significant features can be interpreted as detecting common confounding factors in time series experiments by applying GPLVM in advance to two-sample tests of [Stegle2010] on the whole dataset. Two-sample tests on Gaussian Processes decide differential expression based on the bayes factor of marginal probabilities for control and treatment being modeled by one common or two separate underlying function(s). As GPLVM is based on Gaussian Processes it provides a covariance structure of confounders in the dataset. We take this covariance structure between features to build up a two-sample Gaussian Process model taking confounding factors throughout the dataset into account.

To account for confounding factors in `gptwosample` simply at the option `-c N` to the run call, where `N` is the number of confounding factors to learn.

1.3.4 Timeshift detection between replicates

A novel covariance function detecting timehifts between time series accounts for temporal mismatches between time series, (of replicates and samples) which share similar patterns, shifted in time. This allows for additional correction of confounding variation in time, as treatment might slow down reaction time of cell-cycle genes, leading to a bunch of falsely positive predicted non differential expressed genes downstream.

To enable timeshift detection add the flag `-t` to the run script. Timeshifts for all replicates will be reported in `results.csv`, where the order of replicates is the same order as in the input files `FILE FILE` (see *Parameter options*).

1.3.5 Result structure

The results are given in form of a `results.csv`. Each line corresponds to the results for one gene. The results file is structured as follows:

```
| Gene ID | Bayes Factor | [Learnt covariance function parameters] |
```

The `Gene ID` is the ID given in the input files. The `Bayes Factor` is a log-score for model comparison of the individual model against the common model. The individual model assumes both samples (treatment and control) to be modelled individually by one Gaussian process each. In contrast the common model assumes both samples to be modelled by one Gaussian Process. Both likelihoods are computed and the score is created by contrasting the both likelihoods:

$$\mathcal{BF} = \ln \frac{p(\text{Individual model})}{p(\text{Common model})}$$

All plots are saved in a subfolder `<outdir>/plots/`

1.3.6 Step by step tutorial & examples

Once the data has been prepared, `gptwosample` can be executed from the unix command line. See the full usage information in *Parameter options*.

See format for input data `.csv` files in *Data format*.

Make sure you either install gptwosample (*Installing the package*) or `cd` into the extracted gptwosample folder before running this tutorial.

Try printing the full help of the script using:

```
python gptwosample --help
```

If an error occurs, you probably `cd` one level too deep and you can `cd ..` up one level.

In this tutorial we will build up a full usage call of `gptwosample`. First, we want to run gptwosample verbosly, thus the call so far looks like:

```
gptwosample -v
```

To enable plotting we provide the switch `-p` to the script:

```
python gptwosample -v -p
```

We want to correct for timeshifts (more on *Timeshift detection between replicates*), thus we enable the timeshift switch `-t`:

```
python gptwosample -v -p -t
```

Next we could additionally learn x confounding factors (see *Accounting for confounding factors* for details on confounding factors) and account for them while two-sampling:

```
python gptwosample -v -p -t -c x
```

but we do not want to account for confounders in this tutorial.

The output of the script shall be in the subfolder `./tutorial/`, so we add the output flag `-o ./tutorial/`:

```
python gptwosample -v -p -t -o ./tutorial/
```

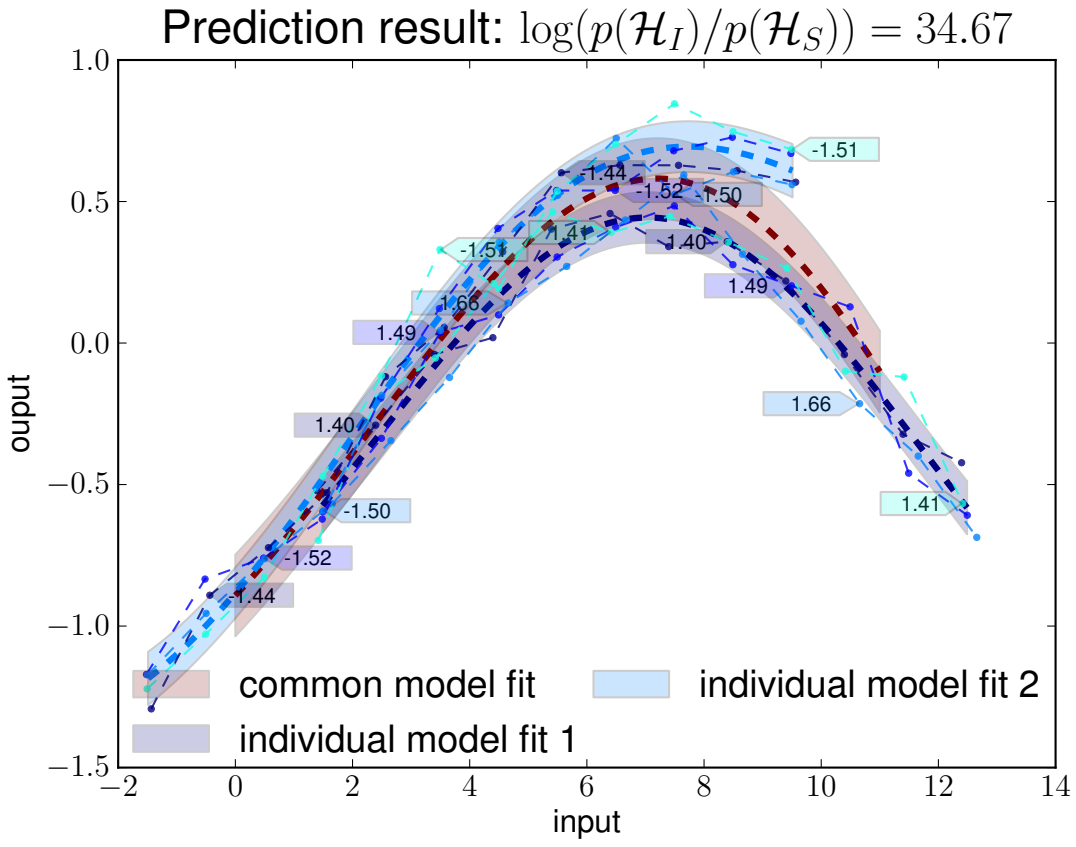
The script shall be run on the two toy condition files `ToyCondition{1,2}.csv` given in `examples/ToyCondition{1,2}.csv`. These files are non optional as this package is only for comparing two timeseries experiments to each other:

```
python gptwosample -v -p -t -o ./tutorial/ examples/ToyCondition1.csv examples/ToyCondition2.csv
```

Note that the optional parameters could be collected together to give rise to a more compact call signature:

```
python gptwosample -vpto tutorial examples/ToyCondition1.csv
examples/ToyCondition2.csv
```

After hitting return the script runs gptwosample on every gene given in the ToyCondition files and plots each gene into `tutorial/plots/`. One example plot will look like:



The results are saved in the `results.csv`, which contains all predicted Bayes Factors and learnt covariance function parameters for all genes (*Result structure*).

For more tutorials and example files on how to use this package see `gptwosample/examples`.

CHAPTER

TWO

DEVELOPER

2.1 Package for using GPTwoSample

This module allows the user to compare two timelines with respect to diffferential expression.

It compares two timeseries against each other, depicting whether these two timeseries were more likely drawn from the same function, or from different ones. This prediction is defined by which covariance function `pygp.covar` you use.

Created on Jun 15, 2011

@author: Max Zwiessele, Oliver Stegle

class `gptwosample.twosample.twosample.`**`TwoSample`**(*T*, *Y*, *covar_common=None*, *covar_individual_1=None*, *covar_individual_2=None*)

Bases: `object`

Run GPTwoSample on given data.

Parameters:

- T : TimePoints [n x r x t] [Samples x Replicates x Timepoints]
- Y : ExpressionMatrix [n x r x t x d] [Samples x Replicates x Timepoints x Genes]

Fields:

- T: Time Points [n x r x t] [Samples x Replicates x Timepoints]
- Y: Expression [n x r x t x d] [Samples x Replicates x Timepoints x Genes]
- X: Confounders [nrt x 1+q] [SamplesReplicatesTimepoints x T+q]
- lvm_covariance: GPLVM covaraince function used for confounder learning
- n: Samples
- r: Replicates
- t: Timepoints
- d: Genes
- q: Confounder Components

`bayes_factors`(*likelihoods=None*)

get list of bayes_factors for all genes.

returns: bayes_factor for each gene in Y

plot(*xlabel='input'*, *ylabel='ouput'*, *title=None*, *interval_indices=None*, *alpha=None*, *legend=True*, *replicate_indices=None*, *shift=None*, *timeshift=False*, **args*, ***kwargs*)
iterate through all genes and plot

predict_likelihoods(*T*, *Y*, *message='Predicting Likelihoods: '*, *messages=False*, *priors=None*, ***kwargs*)
Predict all likelihoods for all genes, given in Y

parameters:

indices [[int]] list (or array-like) for gene indices to predict, if None all genes will be predicted

message: str printing message

kwargs: {...} kwargs for `gptwosample.twosample.GPTwoSampleBase.predict_model_likelihoods`

predict_means_variances(*interpolation_interval*, *indices=None*, *message='Predicting means and variances: '*, **args*, ***kwargs*)
Predicts means and variances for all genes given in Y for given interpolation_interval

set_data(*T*, *Y*)
Set data by time T and expression matrix Y:

Parameters:

T [real [n x r x t]] All Timepoints with shape [Samples x Replicates x Timepoints]

Y [real [n x r x t x d]] All expression values given in the form: [Samples x Replicates x Timepoints x Genes]

class `gptwosample.confounder.confounder.`**TwoSampleConfounder**(*T*, *Y*, *q=4*, *lvm_covariance=None*, *init='random'*, *covar_common=None*, *covar_individual_1=None*, *covar_individual_2=None*)
Bases: `gptwosample.twosample.twosample.TwoSample`

Run GPTwoSample on given Data

Parameters:

- T : TimePoints [n x r x t] [Samples x Replicates x Timepoints]
- Y : ExpressionMatrix [n x r x t x d] [Samples x Replicates x Timepoints x Genes]
- q : Number of Confounders to use
- lvm_covariance : optional - set covariance to use in confounder learning
- init : [random, pca]

Fields:

- T: Time Points [n x r x t] [Samples x Replicates x Timepoints]
- Y: Expression [n x r x t x d] [Samples x Replicates x Timepoints x Genes]
- X: Confounders [nrt x 1+q] [SamplesReplicatesTimepoints x T+q]
- lvm_covariance: GPLVM covaraince function used for confounder learning
- n: Samples
- r: Replicates
- t: Timepoints

- d: Genes
- q: Confounder Components

initialize_twosample_covariance(*covar_common=<function <lambda> at 0x1149060c8>*, *covar_individual_1=<function <lambda> at 0x1149061b8>*, *covar_individual_2=<function <lambda> at 0x114906938>*)

initialize twosample covariance with function covariance(XX), where XX is a FixedCF with the learned confounder matrix.

default is SumCF([SqexpCFARD(1), FixedCF(self.K_conf.copy()), BiasCF()])

learn_confounder_matrix(*ard_indices=None*, *x=None*, *messages=True*, *gradient_tolerance=1e-12*, *lvm_dimension_indices=None*, *gradcheck=False*, *maxiter=10000*)

Learn confounder matrix with this model.

Parameters:

x [array-like] If you provided an own lvm_covariance you have to specify the X to use within GPLVM

lvm_dimension_indices [[int]] If you specified an own lvm_covariance you have to specify the dimension indices for GPLVM

ard_indices [[indices]] If you provided an own lvm_covariance, give the ard indices of the covariance here, to be able to use the correct hyperparameters for calculating the confounder covariance matrix.

class gptwosample.twosample.twosample_base.**TwoSampleShare**(*covar*, **args*, ***kwargs*)

Bases: gptwosample.twosample.twosample_base.TwoSampleBase

This class provides comparison of two Timeline Groups to each other.

see gptwosample.twosample.twosample_base.TwoSampleBase for detailed description of provided methods.

class gptwosample.twosample.twosample_base.**TwoSampleSeparate**(*covar_individual_1*, *covar_individual_2*, *covar_common*, ***kwargs*)

Bases: gptwosample.twosample.twosample_base.TwoSampleBase

This class provides comparison of two Timeline Groups to one another, inlcuding timeshifts in replicates, respectively.

see gptwosample.twosample.twosample_base.TwoSampleBase for detailed description of provided methods.

Note that this model will need one covariance function for each model, respectively!

class gptwosample.twosample.twosample_base.**TwoSampleBase**(*learn_hyperparameters=True*, *priors=None*, *initial_hyperparameters=None*, ***kwargs*)

Bases: object

TwoSampleBase object with the given covariance function covar.

bayes_factor(*model_likelihoods=None*)

Return the Bayes Factor for the given log marginal likelihoods model_likelihoods

Parameters:

model_likelihoods [{'individual': *the individual likelihoods*, 'common': *the common likelihoods*}]
The likelihoods calculated by predict_model_likelihoods(training_data) for given training data training_data.

get_data(*model='common model fit'*, *index=None*)
get inputs of model *model* with group index *index*. If index is None, the whole model group will be returned.

get_learned_hyperparameters()
Returns learned hyperparameters in model structure, if already learned.

get_model_likelihoods()
Returns all calculated likelihoods in model structure. If not calculated returns None in model structure.

get_predicted_mean_variance()
Get the predicted mean and variance as:

```
{'individual':{'mean':[pointwise mean], 'var':[pointwise variance]},
     'common':{'mean':[pointwise mean], 'var':[pointwise variance]}}
```

If not yet predicted it will return 'individual' and 'common' empty.

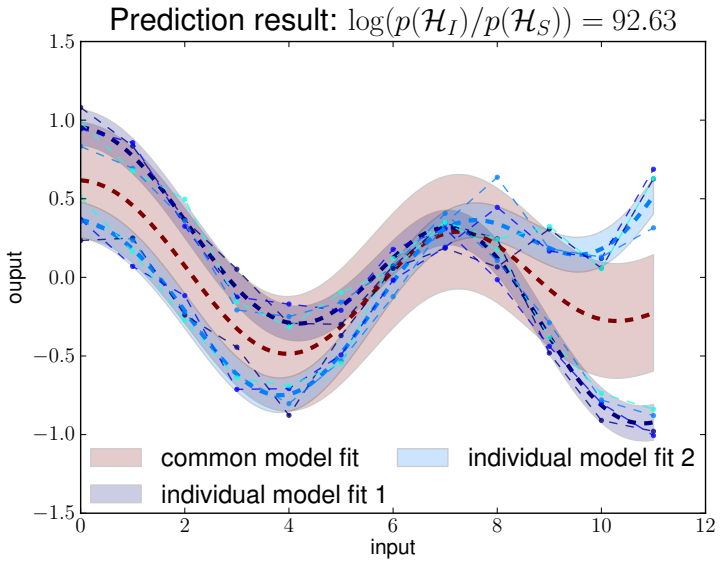
plot(*xlabel='input'*, *ylabel='ouput'*, *title=None*, *interval_indices=None*, *alpha=None*, *legend=True*, *replicate_indices=None*, *shift=None*, **args*, ***kwargs*)
Plot the results given by last prediction.

Two Instance Plots of comparing two groups to each other:

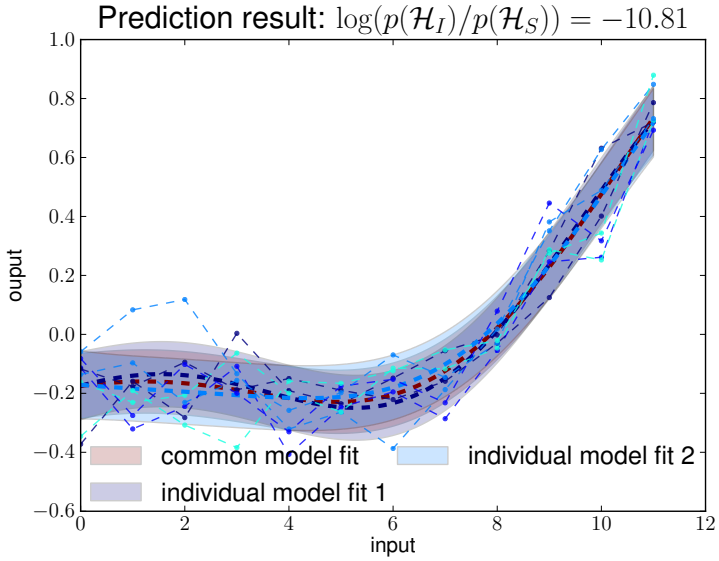
Parameters:

twosample_object [gptwosample.twosample] GPTwoSample object, on which already 'predict' was called.

Differential Groups:



Non-Differential Groups:



Returns: Proper rectangles for use in pylab.legend().

predict_mean_variance (*interpolation_interval*, *hyperparams=None*, *interval_indices={'common model fit': None, 'individual model fit': None}*, **args*, ***kwargs*)

Predicts the mean and variance of both models. Returns:

```
{'individual':{'mean':[pointwise mean], 'var':[pointwise variance]},
     'common':{'mean':[pointwise mean], 'var':[pointwise variance]}}
```

Parameters:

interpolation_interval [[double]] The interval of inputs, which shall be predicted

hyperparams [{'covar':logtheta, ...}] Default: learned hyperparameters. Hyperparams for the covariance function's prediction.

interval_indices [{'common':[boolean],'individual':[boolean]}] Indices in which to predict, for each group, respectively.

predict_model_likelihoods (*training_data=None*, *interval_indices={'common model fit': None, 'individual model fit': None}*, **args*, ***kwargs*)

Predict the probabilities of the models (individual and common) to describe the data. It will optimize hyperparameters respectively.

Parameters:

training_data [dict traning_data] The training data to learn from. Input are time-values and output are expression-values of e.g. a timeseries. If not given, training data must be given previously by `gptwosample.twosample.basic.set_data`.

interval_indices: **`gptwosample.data.data_base.get_model_structure()`** interval indices, which assign data to individual or common model, respectively.

args [[..]] see `pygp.gpr.gp_base.GP`

kwargs [{..}] see `pygp.gpr.gp_base.GP`

set_data (*training_data*)

Set the data of prediction.

Parameters:

training_data [dict traning_data] The training data to learn from. Input are time-values and output are expression-values of e.g. a timeseries.

Training data training_data has following structure:

```
{'input' : {'group 1':[double] ... 'group n':[double]},
 'output' : {'group 1':[double] ... 'group n':[double]}}
```

2.2 GPTwoSample plot

The easiest way to plot your results in an easy and convenient way.

2.2.1 Plot GPTwoSample predictions

Module for easy plotting of GPTwoSample results.

`gptwosample.plot.plot_basic.plot_results` plots training data, as well as sausage_plots for a GPTwoSample experiment. You can give interval indices for plotting, if u chose

Created on Feb 10, 2011

@author: Max Zwiessele, Oliver Stegle

`gptwosample.plot.plot_basic.`**`plot_results`**(*twosample_object*, *xlabel='input'*, *ylabel='ouput'*, *title=None*, *interval_indices=None*, *alpha=None*, *legend=True*, *replicate_indices=None*, *shift=None*, **args*, ***kwargs*)

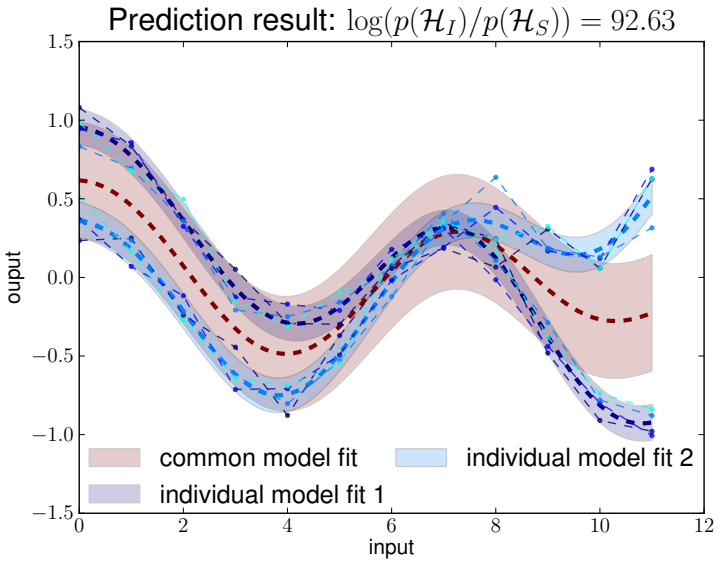
Plot the results given by last prediction.

Two Instance Plots of comparing two groups to each other:

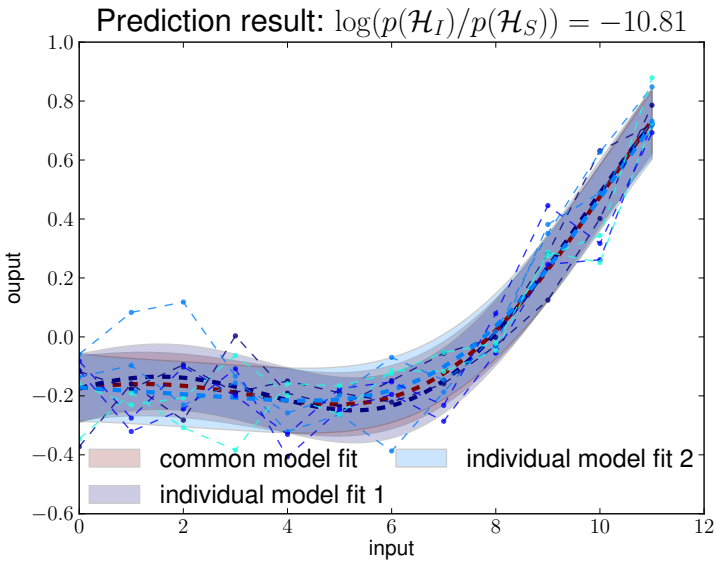
Parameters:

twosample_object [`gptwosample.twosample`] GPTwoSample object, on which already 'predict' was called.

Differential Groups:



Non-Differential Groups:



Returns: Proper rectangles for use in pylab.legend().

.. automodule:: gptwosample.plot.interval # :members:

2.3 Package for data handling

Use this Package for easiest way to handle the data for GPTwoSample.

2.3.1 Data Structure Module

This Module is for easy access to data structures gptwosample works with.

Created on Mar 18, 2011

@author: Max Zwiessele

exception gptwosample.data.data_base.**DataStructureError**(**args*, ***kwargs*)
Bases: exceptions.TypeError

Thrown, if DataStructure given does not fit. Training data training_data has following structure:

```
{input_id : {'group 1':[double] ... 'group n':[double]},
 output_id : {'group 1':[double] ... 'group n':[double]}}
```

gptwosample.data.data_base.**get_model_structure**(*individual=None*, *common=None*)
Returns the valid structure for model dictionaries, used in gptwosample. Make sure to use this method if you want to use the model structure in this package!

gptwosample.data.data_base.**get_training_data_structure**(*x1*, *x2*, *y1*, *y2*)
Get the structure for training data, given two inputs x1 and x2 with corresponding outputs y1 and y2. Make sure, that replicates have to be tiled one after the other for proper resampling of data!

gptwosample.data.data_base.**has_model_structure**(*structure*)
Returns the valid structure for model dictionaries, used in gptwosample. Make sure to use this method if you want to use the model structure in this package!

2.3.2 Data IO tool

For convienent usage this module provides IO operations for data

Created on Jun 9, 2011

@author: Max Zwiessele, Oliver Stegle

gptwosample.data.dataIO.**get_data_from_csv**(*path_to_file*, *delimiter=', '*, *count=-1*, *verbose=True*, *message='Reading File'*, *fil=None*)
Return data from csv file with delimiter delimiter in form of a dictionary. Missing Values are all values x which cannot be converted float(x)

The file format has to fullfill following formation:

arbitrary	**x1**	**...**	**xl**
Gene Name 1	y1 replicate 1	...	yl replicate 1
...	...	...	...
Gene Name 1	y1 replicate k1	...	yl replicate k1
...			
Gene Name n	y1 replicate 1	...	yl replicate 1
...	...	...	...
Gene Name n	y1 replicate kn	...	yl replicate kn

Returns: {"input":[x1,...,xl], "Gene Name 1":[[y1 replicate 1, ... yl replicate 1], ... ,[y1 replicate k, ..., yl replikate k]]}

gptwosample.data.dataIO.**write_data_to_csv**(*data*, *path_to_file*, *header='GPTwoSample'*, *delimiter=', '*)
Write given data in training_data_structure (see gptwosample.data.data_base for details) into file for path_to_file.

Parameters:

data [dict] data to write in training_data_structure

path_to_file [String] The path to the file to write to

header [String] Name of the table

delimiter [character] delimiter for the csv file

BIBLIOGRAPHY

[Lawrence2004] Neil Lawrence, *Gaussian process latent variable models for visualisation of high dimensional data*, Advances in neural information processing systems, *2004*

[Stegle2010] Stegle, Oliver and Denby, Katherine J and Cooke, Emma J and Wild, David L and Ghahramani, Zoubin and Borgwardt, Karsten M, *A robust Bayesian two-sample test for detecting intervals of differential gene expression in microarray time series*, Journal of Computational Biology, *2010*

PYTHON MODULE INDEX

g

gptwosample, 9
gptwosample.data, 15
gptwosample.data.data_base, 15
gptwosample.data.dataIO, 16
gptwosample.plot, 14
gptwosample.plot.plot_basic, 14