GPTwoSample Documentation

Release 0.1.7a

Oliver Stegle, Max Zwießele

April 07, 2013

Contents

This package provides Gaussian Process based differential gene expression detection.

Contents:

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 0x112fd77d0>*, *covar_individual_1=<function <lambda> at 0x112fd76e0>*, *covar_individual_2=<function <lambda> at 0x112fd7f50>*)

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'*, *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.

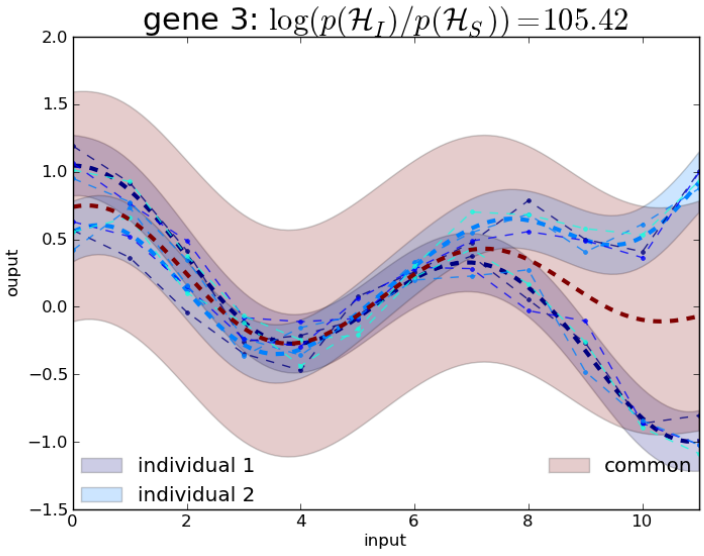
plot(*xlabel='input'*, *ylabel='ouput'*, *title=None*, *interval_indices=None*, *alpha=None*, *legend=True*, *replicate_indices=None*, *shift=None*, **args*, ***kwargs*)
Plot the results given by last prediction.

Two Instance Plots of comparing two groups to each other:

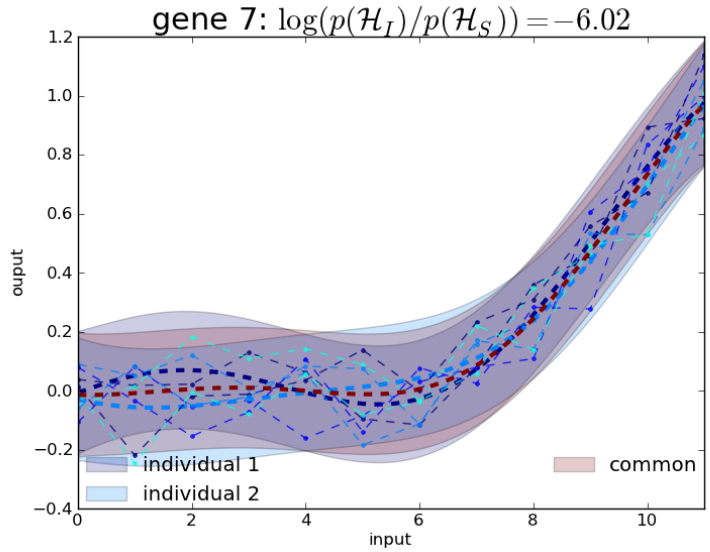
Parameters:

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={'individual': None, 'common': 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={'individual': None, 'common': 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]}}
```

CHAPTER 2

GPTwoSample plot

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

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*)

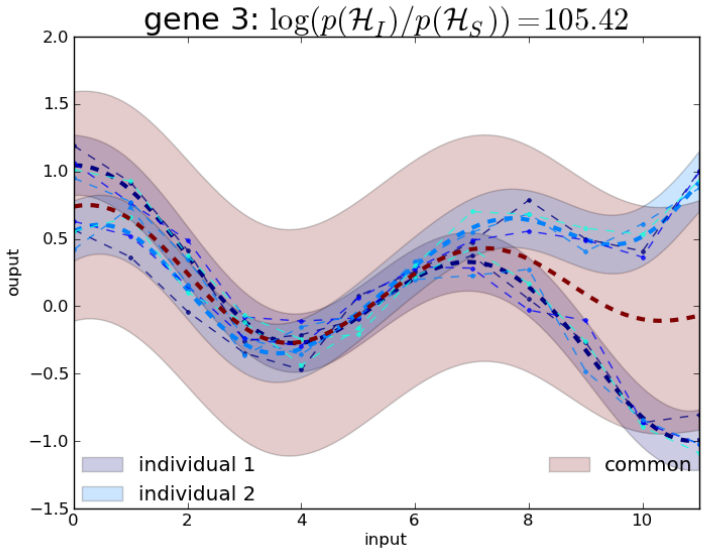
Plot the results given by last prediction.

Two Instance Plots of comparing two groups to each other:

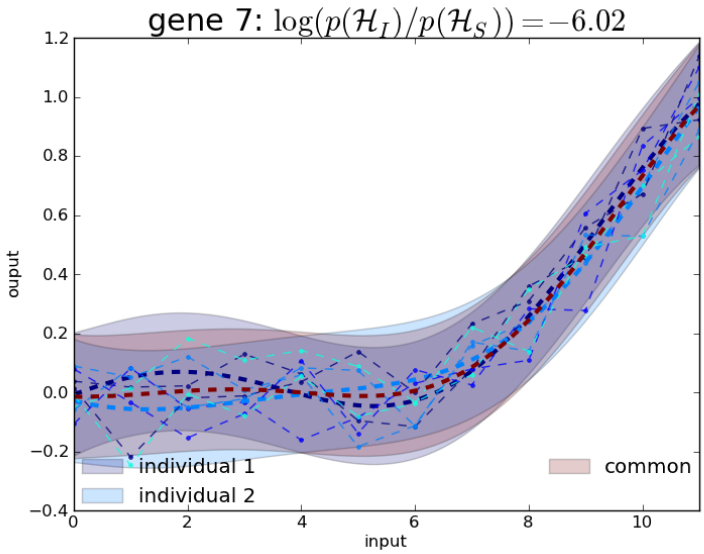
Parameters:

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:

CHAPTER 3

Package for data handling

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

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!

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'*)

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

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

g