congas

Release 0.1

Oct 25, 2021

Contents

1 Copy number genotyping from scRNA sequencing **1**
1.1 Anneal Core 2
1.2 Models 4

2 Indices and tables **9**

Python Module Index **11**

Index **13**

CHAPTER 1

Copy number genotyping from scRNA sequencing

A set of Pyro models and functions to infer CNA from scRNA-seq data. It comes with a companion R package (**in progress**) that works as an interface and provides preprocessing, simulation and visualization routines.

Currently providing:

- A mixture model on segments where CNV are modelled as LogNormal random variable (MixtureGaussian)
- Same as above but the number of cluster is learned (MixtureGaussianDMP)
- A model where CNVs are modelled as outcome from Categorical distributions, clusters share the same parameters (MixtureDirichlet)
- A simple Hmm where CNVs are again categorical, but there is no clustering (SimpleHmm)
- The version of MixtureDirichlet but with temporal dependency (HmmMixtureRNA)

Coming soon: - NUTS support

To install:

```
$ pip install congas
```

To run a simple analysis on the example data

```
import congas as cn
from congas.models import MixtureGaussian
data_dict = cn.simulation_data
params, loss = cn.run_analysis(data_dict,MixtureGaussian, steps=200, lr=0.05)
```

Warning: package is still under developmet

1.1 Anneal Core

1.1.1 Package contents

ANNEAL: rnA cNvs iNferEnce And cLustering (with Pyro)

A set of Pyro models and an interface for simultaneous CNV clustering and inference

1.1.2 anneal.Interface module

Standardize models execution.

The core of the package just provides class with functions to make every model behave in the same way

class congas.Interface.**Interface**(*model=None*, *optimizer=None*, *loss=None*, *inf_type=<class 'pyro.infer.svi.SVI'>*)

Bases: object

The interface class for all the congas models.

Basically it takes a model, am optimizer and a loss function and provides a functions to run the inference and get the parameters back masking the differences between models.

initialize_model(*data*)

learned_parameters()

Return all the estimated parameter values

Calls the right set of function for retrieving learned parameters according to the model type If posterior=True all the other parameters are just passed to `inference_categorical_posterior()`

Parameters **posterior** – learn posterior assignement (if false estimate MAP via Viterbi-like MAP inference)

Returns parameter:value dictionary

Return type dict

run(*steps*, *param_optimizer={'lr': 0.05}*, *param_loss=None*, *seed=3*, *MAP=False*)

This function runs the inference of non-categorical parameters

This function performs a complete inference cycle for the given tuple(model, optimizer, loss, inference modality). For more info about the parameters for the loss and the optimizer look at Optimization. and Loss.

Not all the the combinations Optimize-parameters and Loss-parameters have been tested, so something may not work (please open an issue on the GitHub page).

Parameters

- **steps** (*int*) – Number of steps
- **param_optimizer** (*dict*) – A dictionary of paramaters:value for the optimizer
- **param_loss** (*dict*) – A dictionary of paramaters:value for the loss function
- **seed** (*int*) – seed to be passed to pyro.set_rng_seed
- **MAP** (*bool*) – Perform learn a Delta distribution over the outer layer of the model graph
- **verbose** (*bool*) – show loss for each step, if false the functions just prints a progress bar

- **BAF** (*torch.tensor*) – if provided use BAF penalization in the loss

Returns loss (divided by sample size) for each step

Return type list

set_loss(*loss*)

set_model(*model*)

set_model_params(*param_dict*)

set_optimizer(*optimizer*)

1.1.3 anneal.utils module

Utils class

A set of utils function to run automatically an enetire inference cycle, plotting and saving results.

congas.utils.**dict_to_tensor**(*dict*)

congas.utils.**load_simulation_seg**(*dir*, *prefix*)

Read data from companion R package simulation

A function to read the

Parameters

- **dir** – directory where the simulation files are stored
- **prefix** –

Returns:

congas.utils.**log_sum_exp**(*args*)

congas.utils.**plot_loss**(*loss*, *save=False*, *output='run1'*)

congas.utils.**run_analysis**(*data_dict*, *model*, *optim=<function ClippedAdam>*, *elbo=<class 'pyro.infer.traceenum_elbo.TraceEnum_ELBO'>*, *inf_type=<class 'pyro.infer.svi.SVI'>*, *steps=500*, *lr=0.01*, *param_dict={}*, *MAP=True*, *seed=3*)

Run an entire analysis with the minimum amount of parameters

Simple function to run an entire step of inference and get the learned parameters back, less customizable than using directly the `Interface` , but still should satisfy most of hte user. Look at the R interface for even a easier

Parameters

- **data_dict** – dictionary with parameters
- **model** – a model from one in congas.models
- **optim** – an optimizer from pyro.optim
- **elbo** – a loss function from pyro.infer
- **inf_type** – SVI or NUTS (Hemiltonian MCMC)
- **steps** – number of inference steps
- **lr** – learning rate
- **param_dict** – parameters for the model, look at the model documentation if you want to change them

- **MAP** – perform MAP over the last layer of random variable in the model or learn the parameters of the distribution
- **seed** – seed for pyro.set_rng_seed
- **step_post** – steps if learning also posterior probabilities

Returns dictionary of parameters:value list: loss (divided by sample size) for every time step (not the one for posteriors)

Return type dict

congas.utils.**write_results**(*params*, *prefix*, *new_dir=False*, *dir_pref=None*)

Write parameters

This function writes the parameters appending a prefix and optionally in a new directory

Parameters

- **params** – parameters dictionary
- **prefix** – prefix to append to the filenames
- **new_dir** – create a new directory or use an exsisting ones
- **dir_pref** – name of the directory

1.2 Models

1.2.1 Models

anneal.models package

Submodules

anneal.models.HmmMIxtureRNA module

anneal.models.HmmSimple module

class congas.models.HmmSimple.**HmmSimple**(*data_dict*)

Bases: *congas.models.Model.Model*

Simple Hmm, models the CNV event as a Categorical variable. It does not cluster the data

Model parameters: T = max number of clusters (default = 6) init_probs = prior probs for initial state CNV probabilities (default=torch.tensor([0.1,0.1,0.2,0.3,0.2,0.1])) hidden_dim = hidden dimensions (should be len(probs)) theta_scale = scale for the normalization factor variable (default = 3) theta_rate = rate for the normalization factor variable (default = 1) batch_size = batch size (default = None) t = probability of remaining in the same state (default=0.1)

data_name = {'data', 'mu', 'pld', 'segments'}

guide(*MAP=False*, **args*, ***kwargs*)

init_fn()

model(**args*, ***kwargs*)

params = {'batch_size': None, 'hidden_dim': 6, 'init_probs': tensor([0.1000, 0.1000

anneal.models.MixtureCategorical module

anneal.models.MixtureDirichlet module

class congas.models.MixtureDirichlet.**MixtureDirichlet**(*data_dict*)
Bases: *congas.models.Model.Model*

create_dirichlet_init_values()

data_name = {'data', 'mu', 'pld', 'segments'}

full_guide(*MAP=False*, **args*, ***kwargs*)

guide(*MAP=False*, **args*, ***kwargs*)

init_fn()

model(**args*, ***kwargs*)

params = {'K': 2, 'batch_size': None, 'cnv_mean': 2, 'gamma_multiplier': 5, 'hidden

anneal.models.MixtureGaussian module

class congas.models.MixtureGaussian.**MixtureGaussian**(*data_dict*)
Bases: *congas.models.Model.Model*

A simple mixture model for CNV inference, it assumes independence among the different segments, needs to be used after calling CNV regions with bulk DNA or RNA. CNVs events are modelled as LogNormal distributions.

Model parameters:

- **K = number** of clusters (default = 2)
- **cnv_var = var of the LogNorm prior (default = 0.6)
- **theta_scale =** scale for the normalization factor variable (default = 3)
- **theta_rate =** rate for the normalization factor variable (default = 1)
- **batch_size =** batch size (default = None)
- **mixture =** prior for the mixture weights (default = 1/torch.ones(K))

calculate_cluster_assignements(*inf_params*)

create_gaussian_init_values()

data_name = {'data', 'mu', 'pld'}

full_guide(*MAP=False*, **args*, ***kwargs*)

guide(*MAP=False*, **args*, ***kwargs*)

init_fn()

likelihood(*inf_params*)

model(**args*, ***kwargs*)

params = {'K': 2, 'assignments': None, 'batch_size': None, 'cnv_locs': None, 'cnv_s

anneal.models.MixtureGaussianDMP module

class congas.models.MixtureGaussianDMP.**MixtureGaussianDMP**(*data_dict*)
Bases: *congas.models.Model.Model*

create_gaussian_init_values()

data_name = {'data', 'mu', 'pld', 'segments'}

full_guide(*MAP=False*, **args*, ***kwargs*)

guide(*MAP=False*, **args*, ***kwargs*)

init_fn()

mix_weights(*beta*)

model(**args*, ***kwargs*)

params = {'T': 6, 'alpha': 0.0001, 'batch_size': None, 'cnv_mean': 2, 'cnv_var': 0

anneal.models.MixtureGaussianEXP module

class congas.models.MixtureGaussianEXP.**MixtureGaussianEXP**(*data_dict*)
Bases: *congas.models.Model.Model*

A simple mixture model for CNV inference, it assumes independence among the different segments, needs to be used after calling CNV regions with bulk DNA or RNA. CNVs events are modelled as LogNormal distributions.

Model parameters:

- **K = number** of clusters (default = 2)
- **cnv_var = var of the LogNorm prior (default = 0.6)
- **theta_scale =** scale for the normalization factor variable (default = 3)
- **theta_rate =** rate for the normalization factor variable (default = 1)
- **batch_size =** batch size (default = None)
- **mixture =** prior for the mixture weights (default = 1/torch.ones(K))

create_gaussian_init_values()

data_name = {'data', 'pld'}

full_guide(*MAP=False*, **args*, ***kwargs*)

guide(*MAP=False*, **args*, ***kwargs*)

init_fn()

model(**args*, ***kwargs*)

params = {'K': 2, 'batch_size': None, 'cnv_sd': 0.6, 'kmeans': True, 'mixture': No

anneal.models.MixtureGaussianGenes module

anneal.models.Model module

class congas.models.Model.**Model**(*data_dict*, *data_name*)
Bases: abc.ABC

All the models in the package have more or less the same structure. Cells are assumed to come from different population based on their copy-number profiles. Each model treats the CNV in a different way, but the common idea is to have an explicit formula for the counts of a gene or a genomic segment and treat them as if they depends only on the CNV and a cell specific factor (for segments, for genes we also have a gene dependent effect).

Moreover gene/segment counts are independent over the genome (or at least they hava a temporal dependency of first order). Given that we can write a mixture model factorizing the CNV for every segment.

BIC()

guide()

model()

set_params(*params_dict*)

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

congas, 2
congas.Interface, 2
congas.models.HmmSimple, 4
congas.models.MixtureDirichlet, 5
congas.models.MixtureGaussian, 5
congas.models.MixtureGaussianDMP, 6
congas.models.MixtureGaussianEXP, 6
congas.models.Model, 6
congas.utils, 3

Index

B

BIC() (*congas.models.Model.Model method*), 7

C

calculate_cluster_assignements() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
congas (*module*), 2
congas.Interface (*module*), 2
congas.models.HmmSimple (*module*), 4
congas.models.MixtureDirichlet (*module*), 5
congas.models.MixtureGaussian (*module*), 5
congas.models.MixtureGaussianDMP (*module*), 6
congas.models.MixtureGaussianEXP (*module*), 6
congas.models.Model (*module*), 6
congas.utils (*module*), 3
create_dirichlet_init_values() (*congas.models.MixtureDirichlet.MixtureDirichlet method*), 5
create_gaussian_init_values() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
create_gaussian_init_values() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
create_gaussian_init_values() (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP method*), 6

D

data_name (*congas.models.HmmSimple.HmmSimple attribute*), 4
data_name (*congas.models.MixtureDirichlet.MixtureDirichlet attribute*), 5
data_name (*congas.models.MixtureGaussian.MixtureGaussian attribute*), 5
data_name (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP attribute*), 6
data_name (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP attribute*), 6
dict_to_tensor() (*in module congas.utils*), 3

F

full_guide() (*congas.models.MixtureDirichlet.MixtureDirichlet method*), 5
full_guide() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
full_guide() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
full_guide() (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP method*), 6

G

guide() (*congas.models.HmmSimple.HmmSimple method*), 4
guide() (*congas.models.MixtureDirichlet.MixtureDirichlet method*), 5
guide() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
guide() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
guide() (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP method*), 6
guide() (*congas.models.Model.Model method*), 7

H

HmmSimple (*class in congas.models.HmmSimple*), 4

I

init_fn() (*congas.models.HmmSimple.HmmSimple method*), 4

init_fn() (*congas.models.MixtureDirichlet.MixtureDirichlet method*), 5
init_fn() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
init_fn() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
init_fn() (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP method*), 6
initialize_model() (*congas.Interface.Interface method*), 2
Interface (*class in congas.Interface*), 2

L

learned_parameters() (*congas.Interface.Interface method*), 2
likelihood() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
load_simulation_seg() (*in module congas.utils*), 3
log_sum_exp() (*in module congas.utils*), 3

M

mix_weights() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
MixtureDirichlet (*class in congas.models.MixtureDirichlet*), 5
MixtureGaussian (*class in congas.models.MixtureGaussian*), 5
MixtureGaussianDMP (*class in congas.models.MixtureGaussianDMP*), 6
MixtureGaussianEXP (*class in congas.models.MixtureGaussianEXP*), 6
Model (*class in congas.models.Model*), 6
model() (*congas.models.HmmSimple.HmmSimple method*), 4
model() (*congas.models.MixtureDirichlet.MixtureDirichlet method*), 5
model() (*congas.models.MixtureGaussian.MixtureGaussian method*), 5
model() (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP method*), 6
model() (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP method*), 6
model() (*congas.models.Model.Model method*), 7

P

params (*congas.models.HmmSimple.HmmSimple attribute*), 4
params (*congas.models.MixtureDirichlet.MixtureDirichlet attribute*), 5
params (*congas.models.MixtureGaussian.MixtureGaussian attribute*), 5
params (*congas.models.MixtureGaussianDMP.MixtureGaussianDMP attribute*), 6
params (*congas.models.MixtureGaussianEXP.MixtureGaussianEXP attribute*), 6
plot_simulation() (*in module congas.utils*), 3

R

run() (*congas.Interface.Interface method*), 2
run_analysis() (*in module congas.utils*), 3

S

set_loss() (*congas.Interface.Interface method*), 3
set_model() (*congas.Interface.Interface method*), 3
set_model_params() (*congas.Interface.Interface method*), 3
set_optimizer() (*congas.Interface.Interface method*), 3
set_params() (*congas.models.Model.Model method*), 7

W

write_results() (*in module congas.utils*), 4