Vireo Documentation

Release 0.5.8

Yuanhua Huang

Feb 22, 2023

Main

CHAPTER 1

About Vireo

This documentation gives an introduction and usage manual of Vireo (Variational inference for reconstructing ensemble origins), a Bayesian method to demultiplex pooled scRNA-seq data with or without genotype reference.

Vireo is primarily designed for demultiplexing cells into donors by modelling of expressed alleles. It supports a variety of settings of donor genotype (from entirely missing, to partially missing, to fully observed). See more details in manual section.

As a general cell clustering methods by allelic ratio (equivalent to genotyping), Vireo is applicable for more settings besides donor demultiplexing, including reconstruction of somatic clones, see vireoSNP_clones.ipynb for example on mitochondral mutations.

CHAPTER 2

Notebooks for interactive analysis

Here are some notebooks for interactive analysis. Usually, you only need to use the command line to perform donor deconvolution, but you may refer to some of these notebooks for additional analysis.

donors: vireoSNP_donors.ipynb gives example on donor deconvolution manually. The *vireo* command line does this job automatically.

donors: donor_match.ipynb gives example on aligning donors to other omics data or other batches

clones: vireoSNP_clones.ipynb gives example on clone reconstruction on mitochondral mutations

CHAPTER 3

Quick Resources

Latest version on GitHub https://github.com/single-cell-genetics/vireo

Scripts for simulation https://github.com/single-cell-genetics/vireo/tree/master/simulate

All releases https://pypi.org/project/vireoSNP/#history

CHAPTER 4

Issue reports

If you find any error or suspicious bug, we will appreciate your report. Please write them in the github issues: https://github.com/single-cell-genetics/vireo/issues

CHAPTER 5

References

Yuanhua Huang, Davis J. McCarthy, and Oliver Stegle. Vireo: Bayesian demultiplexing of pooled single-cell RNA-seq data without genotype reference. **Genome Biology** 20, 273 (2019)

5.1 Installation

Required packages in python: `numpy>=1.9.0`, `scipy>=1.0`, `matplotlib`

Environment: we only tested Vireo in `Python 3` environment, so if it fails in Python 2, please try it in Python 3 before reporting the issue.

We recommend using Anaconda distribute to set up the environment. It not only includes all dependent packages, but also provides a user controlled environment, namely, you will have the root permission for this distribution, including installation of any package.

5.1.1 Easy install from PyPI

You can install *Vireo* simply via PyPI in terminal (**suggested**), or upgrade by adding `--upgrade` as follows:

```
pip install vireoSNP

pip install --upgrade --no-deps vireoSNP
```

5.1.2 Install from source code

Alternatively, you can download the source code from GitHub (for the latest version) and run python setup in terminal:

```
wget https://github.com/huangyh09/vireo/archive/master.zip
unzip master.zip
cd vireo-master
```

(continues on next page)

(continued from previous page)

```
python setup.py install
```

You can also use the following shortcut

```
pip install -U git+https://github.com/single-cell-genetics/vireo
```

In any case, if had the permission error for installation as you are not root, add `--user`.

5.1.3 Quick check

In order to test the installation, you could type `vireo` in terminal. If successfully installed, you will see the following output.

```
Welcome to vireoSNP v0.1.1!

use -h or --help for help on argument.
```

If installation is sucessful, but can't run it (e.g., message below), then check whether the directory which contains the executable binary file is added to PATH environment.

```
vireo: command not found
```

If using Anaconda, the executable `vireo` is located in `$anaconda3/bin/vireo`. If not using Anaconda, it is usually located in directory `~/.local/bin`. You could add the path into PATH environment variable, by write the following line into `.profile` or `.bashrc` file.

```
export PATH="~/.local/bin:$PATH"
```

5.2 Manual

Demultiplexing requires two count matrices (variant-by-cell) of reads or UMIs for each variant in each cell: `AD` for alternative allele and `DP` depth (i.e., summary of alternative and reference alleles). These two matrices can be obtained by genotyping a list of variants in each cell. We provide a guideline for cellular genotyping with a recommendation of cellSNP-lite that is also developed by us.

Once the genotypes for each cell have been obtained, e.g., in VCF format, or two sparse matrices `AD` and `DP`, we can apply Vireo for demultiplexing.

5.2.1 Demultiplexing for donors

By default, Vireo works without any known genotype information for pooled samples. However, if any of genotype of these samples are known or can be obtained, e.g., by bulk RNA-seq, exome-seq, it is still useful to add them, not only allowing us to align the deconvoluted samples to its identity, but also can benefits the doublets identification, especially if the coverage or the loaded cells per sample is low.

Depending the availability of genotype information, we provide four strategies to demultiplex scRNA-seq data.

1) without any genotype:

```
vireo -c $CELL_DATA -N $n_donor -o $OUT_DIR
```

2) with genotype for all samples (genoTag: GT, GP, or PL; default is PL, please choose the existing one)

```
vireo -c $CELL_DATA -d $DONOR_GT_FILE -o $OUT_DIR
```

Optionally, *-N* can be provided if it is samller than that in DONOR_GT_FILE for finding the relevant subset of donors.

Note: For efficient loading of donor VCF file, we recommend subset it `bcftools view donor.vcf.gz -R cellSNP.cells.vcf.gz -Oz -o sub.vcf.gz`

You can also add `-s` or `-S` for subsetting samples.

Make sure you only keep informative SNPs, e.g., by filtering out SNPs with too much missing values or the gentoypes too similar across donors.

---

3) with genotype for part of the samples (n_donor is larger than that in DONOR_GT_FILE)

```
vireo -c $CELL_DATA -d $DONOR_GT_FILE -o $OUT_DIR -N $n_donor
```

4) with genotype but not confident (or only for subset of SNPs)

```
vireo -c $CELL_DATA -d $DONOR_GT_FILE -o $OUT_DIR --forceLearnGT
```

Formats of cell data

Viroe supports the cell data in three formats:

1) a cellSNP output folder containing VCF for variants info and sparse matrices `AD` and `DP`
2) Vartrix outputs with three or four files: alt.mtx,ref.mtx,barcodes.tsv[,SNP.vcf.gz]
3) standard VCF file with variants by cells

Vireo full arguments

Type `vireo -h` for details of all arguments:

```
Usage: vireo [options]

Options:
-h, --help            show this help message and exit
-c CELL_DATA, --cellData=CELL_DATA
                      The cell genotype file in VCF format or cellSNP folder
                      with sparse matrices.
-N N_DONOR, --nDonor=N_DONOR
                      Number of donors to demultiplex; can be larger than
                      provided in donor_file
-o OUT_DIR, --outDir=OUT_DIR
                      Dirtectory for output files [default:
                      $cellFilePath/vireo]

Optional input files:
   --vartrixData=VARTRIX_DATA
                      The cell genotype files in vartrix outputs (three/four
                      files, comma separated):
```

(continues on next page)

(continued from previous page)

```
                        alt.mtx,ref.mtx,barcodes.tsv,SNPs.vcf.gz. This will
                        suppress cellData argument.
   -d DONOR_FILE, --donorFile=DONOR_FILE
                        The donor genotype file in VCF format. Please filter
                        the sample and region with bcftools -s and -R first!
   -t GENO_TAG, --genoTag=GENO_TAG
                        The tag for donor genotype: GT, GP, PL [default: PL]

Optional arguments:
   --noDoublet          If use, not checking doublets.
   -M N_INIT, --nInit=N_INIT
                        Number of random initializations, when GT needs to
                        learn [default: 50]
   --extraDonor=N_EXTRA_DONOR
                        Number of extra donor in pre-cluster, when GT needs to
                        learn [default: 0]
   --extraDonorMode=EXTRA_DONOR_MODE
                        Method for searching from extra donors. size: n_cell
                        per donor; distance: GT distance between donors
                        [default: distance]
   --forceLearnGT       If use, treat donor GT as prior only.
   --ASEmode            If use, turn on SNP specific allelic ratio.
   --noPlot             If use, turn off plotting GT distance.
   --randSeed=RAND_SEED
                        Seed for random initialization [default: none]
   --cellRange=CELL_RANGE
                        Range of cells to process, eg. 0-10000 [default: all]
   --callAmbientRNAs    If use, detect ambient RNAs in each cell (under
                        development)
   -p NPROC, --nproc=NPROC
                        Number of subprocesses for computing - this sacrifices
                        memory for speedups [default: 1]
```

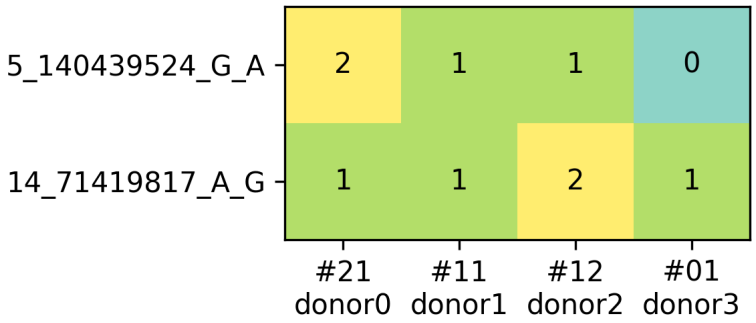
5.2.2 Discriminatory variants

Given a set of variants for which estimated genotypes are available, the Vireo software implements a heuristic to define a minimal and informative set of discriminatory variants. This set of variants can be used to perform qPCR-based genotyping or for other targeted genoytping methods. Briefly, this algorithm prioritises variants with largest information gain in splitting samples.

For any donor genotype file in VCF format, especially the output from Vireo, `GT_donors.vireo.vcf.gz`, the `GTbarcode` function can be used to generate the minimal set of discriminatory variants by the following command line:

```
GTbarcode -i $dir/GT_donors.vireo.vcf.gz -o $dir/GT_barcodes.tsv --randSeed 1
```

By default, this function filters out variants with <20 UMIs or >0.05 reads aligned other alleles except the annotated reference and alternative alleles. In case the variants with homozygous alternative alleles are not wanted, the arguments `--noHomoAlt` can be used. By default, this `GTbarcode` function will also generate a figure for the identified genotype barcode, as following (based on example data in the repo),



5.2.3 Example data

In order to test vireo and illustrate the usage, we provide a test data set, also some demo scripts.

This example data set contains 952 cells from 4 samples. The genotypes for these four samples are also provided.

5.3 Genotyping

Genotyping (or piling up) a list of common variants on each cell is a pre-step for demultiplexing them with Vireo. This step requires some bioinformatics efforts, but thanks to many developers in this community, there are a few good existing software to use.

Genotyping cells can be divided into the following two sub-steps, and in different situations, the strategy may need to be customised for ensuring high quality genotyping data to demultiplex cells.

Recommended strategies for genotyping cells:

- For human or genotyped species: variant list (given) + cellsnp-lite (typing).
- For species without known common variants: cellsnp-lite (calling with mode 2b & typing with mode 1a). freebayes is an alternative to cellsnp-lite mode 2b for calling of heterozygous SNPs.

Note: cellSNP was initially developed in Python based on pysam, which is convenient for a few thousand cells but becomes the computational bottleneck for large number of cells. Therefore, we have re-implemented it to cellsnp-lite in C/C++ with ~5x faster and ~50x less memory.

5.3.1 1. Identify candidate SNPs

There are multiple ways to identify candidate SNPs, each has unique properties and may suit situations or species differently. Here, we listed two common strategies.

Option 1): using a common variants

The best way is to genotype the each of the pooled samples either by genotyping array or exome/genome/RNA sequencing (or other omics data), and use this list of distinguish variants to as candidate to genotype each cell. However, this can be costly, and not necessary in most cases.

For human, a very comprehensive list of common variants have been identified by international efforts, e.g., 10000 genome project and gnomAD which gives a few millions common SNPs to genotype on each cell. The benefits include

the reduced confounders, e.g., caused by RNA editing. We normally recommend this if for human, and we provide some pre-processed SNP list.

Note: Here are some tips if you have genotypes for input donors:

1. Imputation can be helpfule if you obtained genotypes for each human individual from SNP-array or Whole exome-seq.
2. Selection of informative SNPs is useful for filtering out SNPs with identical or very similar genotypes in all mixed donors. You may consider *AC*, *AF* or similar tag in you donor VCF file. bcftools is a very useful tool for such preprocessing.

Option 2): Calling variants from scRNA-seq

Other than human, most species may not have a well-defined common variants, hence the best way is to call the variants from pooled scRNA-seq directly.

The package freebayes is an often choice, which designed to find small polymorphisms, specifically SNPs, indels, MNPs, and complex events smaller than the length of a short-read sequencing alignment. Importantly, freebayes has a set of options to filter reads and variants.

We also recommend an alternative method cellsnp-lite that is developed by us. Its mode 2b has a similar feature to pileup the whole genome and identify the heterozygous variants in the pooled samples. It has highly comparable accuracry to freebayes and bcftools mpileup and achieves 5-10x speedups.

5.3.2 2. Genotype each cell

Once a list of candidate variants are found, it is more straightforward to genotype each cell. We provide three common methods, with recommendation to our cellsnp-lite to seamlessly with Vireo.

- The famous mpileup from bcftools / samtools is often a good choice. However, this can be slow, as it doesn't allow parallel computing and it doesn't support the cell barcodes and UMI tag in the pooled BAM file for many cells.
- This limitation motivates us to develop cellSNP, a pysam wrap (now cellsnp-lite in C/C++) to pile up the variants in each cell. The benefits include parallel computing, taking cell barcoding tag and support UMIs.
- Alternatively, vartrix is also an option to genotype cells in 10x Genomics data.

Once the genotype (mainly pileup) has been achieved, it can be used to demultiplex the pooled cells, see the manual.

5.4 API

Import vireoSNP as:

```
import vireoSNP
```

5.4.1 Commands

- `vireo`: see manual
- `GTbarcode`: see manual

5.4.2 Read / Load

vireoSNP.**read_cellSNP**(*dir_name, layers=['AD', 'DP']*)
Read data from the cellSNP output directory

Parameters **dir_name** – directory full path name for cellSNP output

Returns

Return type A disctionary containing AD, DP, cells and variants

vireoSNP.**read_vartrix**(*alt_mtx*, *ref_mtx*, *cell_file*, *vcf_file=None*)
Read data from VarTrix

Parameters

- **alt_mtx** – sparse matrix file for alternative alleles
- **ref_mtx** – sparse matrix file for reference alleles
- **cell_file** – file for cell barcodes, each per line
- **vcf_file** – the vcf file used for fetch variants in VarTrix

Returns

Return type A disctionary containing AD, DP, cells and optionally variants

5.4.3 VCF processing

Load VCF to matrices

vireoSNP.vcf.**load_VCF**(*vcf_file*, *biallelic_only=False*, *load_sample=True*, *sparse=True*, *format_list=None*)
Initially designed to load VCF from cellSNP output, requiring

1) all variants have the same format list;

2) a line starting with "#CHROM", with sample ids.

If these two requirements are satisfied, this function also supports general VCF files, e.g., genotype for multiple samples.

Note, it may take a large memory, please filter the VCF with bcftools first.

Examples

- Load VCF file, e.g., from cellsnp-lite output:

```
>>> import vireoSNP
>>> import numpy as np
>>> vcf_dat = vireoSNP.vcf.load_VCF("cellSNP.cells.vcf.gz", sparse=False,
>>>     biallelic_only=False, format_list=['GT', 'AD', 'DP', 'ALL'])
>>> var_ids = np.array(vcf_dat['variants'])
>>> samples = np.array(vcf_dat['samples'])
>>> GT_mat = np.array(vcf_dat['GenoINFO']['GT'])
>>> AD_mat = np.array(vcf_dat['GenoINFO']['AD']).astype(float)
>>> DP_mat = np.array(vcf_dat['GenoINFO']['DP']).astype(float)
>>> ALL_bases_mat = np.array(vcf_dat['GenoINFO']['ALL'])
```

Parse genotype probablity to tenseor

vireoSNP.vcf.**parse_donor_GPb**(*GT_dat*, *tag='GT'*, *min_prob=0.0*)
Parse the donor genotype probability tag: GT, GP, or PL

Examples

```
>>> GProb_tensor = vireoSNP.vcf.parse_donor_GPb(vcf_dat['GenoINFO']['GT'], 'GT')
```

vireoSNP.vcf.**match_VCF_samples**(*VCF_file1*, *VCF_file2*, *GT_tag1*, *GT_tag2*)
Match donors in two VCF files. Please subset the VCF with bcftools first, as it is more computationally efficient:

bcftools view large_file.vcf.gz -R small_file.vcf.gz -Oz -o sub.vcf.gz

Parameters

- **VCF_file1** (*str*) – the full path of first VCF file, in plain text or gzip / bgzip
- **VCF_file2** (*str*) – the full path of second VCF file, in plain text or gzip / bgzip
- **GT_tag1** (*str*) – the tag for extracting the genotype probability in VCF1: GT, GP, PL
- **GT_tag2** (*str*) – the tag for extracting the genotype probability in VCF2: GT, GP, PL

5.4.4 Plotting

Heatmap plot

vireoSNP.plot.**heat_matrix**(*X*, *yticks=None*, *xticks=None*, *rotation=45*, *cmap='BuGn'*, *alpha=0.6*, *display_value=True*, *row_sort=False*, *aspect='auto'*, *interpolation='none'*, ***kwargs*)
Plot heatmap of distance matrix

Parameters

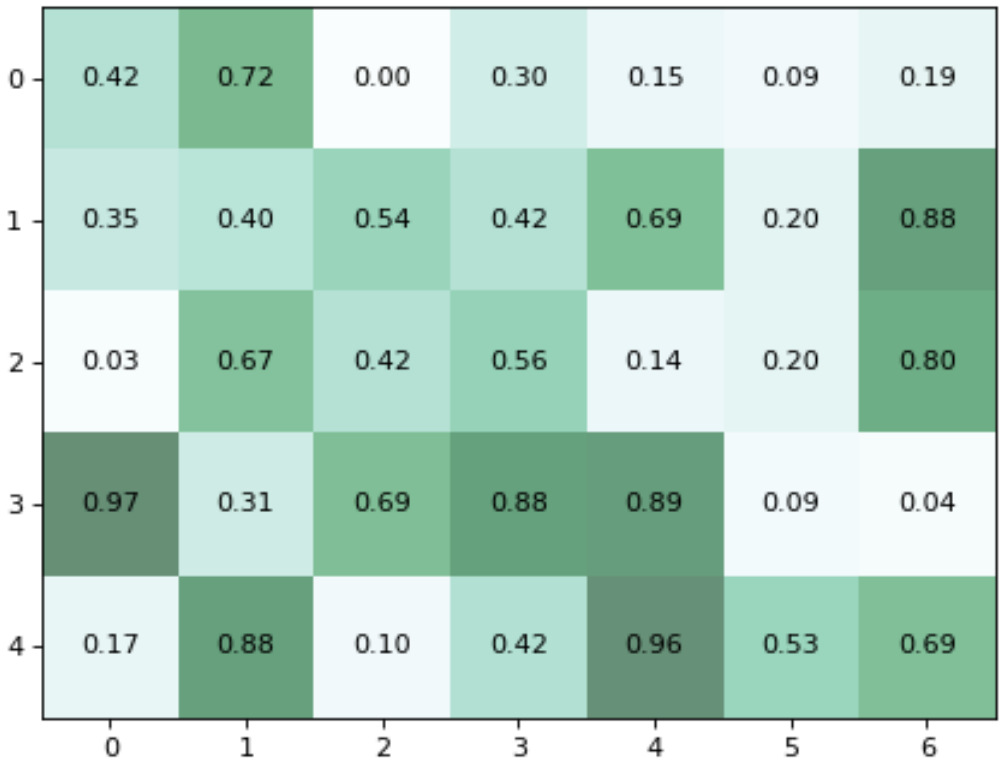
- **X** (*numpy.array or matrix*) – The matrix to plot in heatmap
- **yticks** (*list*) – The ticks ids for y axis
- **xticks** (*list*) – The ticks ids for x axis
- **ratation** (*scalar*) – The ratation angel for xticks
- **cmap** (*str*) – The colormap for the heatmap, more options: https://matplotlib.org/stable/tutorials/colors/colormaps.html
- **alpha** (*scalar*) – The transparency, value between 0 and 1
- **display_value** (*bool*) – If True, dispaly the values in the heatmap
- **raw_sort** (*bool*) – If True, sort the rows with row index as row_idx = np.argsort(np.dot(X, 2**np.arange(X.shape[1])))
- **aspect** (*str*) – *aspect* in *plt.imshow*
- **interpolation** (*str*) – *interpolation* in *plt.imshow*
- ****kwargs** (*keywords & values*) – ***kwargs* for *plt.imshow*

Returns

Return type The return from *plt.imshow*

Examples

```
>>> from vireoSNP.plot import heat_matrix
>>> import numpy as np
>>> np.random.seed(1)
>>> X = np.random.rand(5, 7)
>>> heat_matrix(X)
```



Annotated heatmap plot

vireoSNP.plot.**anno_heat**(*X*, *row_anno=None*, *col_anno=None*, *row_order_ids=None*, *col_order_ids=None*, *xticklabels=False*, *yticklabels=False*, *row_cluster=False*, *col_cluster=False*, ***kwargs*)

Heatmap with column or row annotations. Based on seaborn.clustermap() Row or column will be ordered by the annotation group.

Note, haven't tested if input both row_anno and col_anno.

5.4.5 Vireo Object

Objects of type *Vireo* allow clustering cells by allelic ratio

class vireoSNP.**Vireo**(*n_cell*, *n_var*, *n_donor*, *n_GT=3*, *learn_GT=True*, *learn_theta=True*, *ASE_mode=False*, *fix_beta_sum=False*, *beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*, *GT_prob_init=None*)

Viroe model: Variational Inference for reconstruction of ensemble origin

The prior can be set via set_prior() before fitting the model.

beta_mu: numpy array (1, n_GT) or (n_var, n_GT) Beta mean parameter of theta's posterior

beta_sum: numpy array (1, n_GT) or (n_var, n_GT), same as beta_mu Beta concetration parameter of theta's posterior

ID_prob: numpy array (n_cell, n_donor) Posterior cell assignment probability to each donor

GT_prob: numpy array (n_var, n_donor, n_GT) Posterior genotype probability per variant per donor

__init__(*n_cell*, *n_var*, *n_donor*, *n_GT=3*, *learn_GT=True*, *learn_theta=True*, *ASE_mode=False*, *fix_beta_sum=False*, *beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*, *GT_prob_init=None*)

Initialise Vireo model

Note, multiple initializations are highly recomended to avoid local optima.

Parameters

- **n_cell** (*int.*) – Number of cells
- **n_var** (*int.*) – Number of variants
- **n_donor** (*int.*) – Number of donors
- **n_GT** (*int.*) – Number of genotype categories
- **learn_GT** (*bool.*) – Whether updating *GT_prob*; otherwise using the initial
- **ASE_mode** (*bool.*) – Whether setting allelic ratio *theta* to be variant specific
- **fix_beta_sum** (*bool.*) – Whether fixing the concetration parameter of theta's posterior
- **beta_mu_init** (*numpy array (1, n_GT) or (n_var, n_GT)*) – Initial value of beta_mu, the mean parameter of theta
- **beta_sum_init** (*numpy array (1, n_GT) or (n_var, n_GT), same as beta_mu*) – Initial value of beta_sum, the concetration parameter of theta
- **ID_prob_init** (*numpy array (n_cell, n_donor)*) – Initial value of ID_prob, cell assignment probability to each donor
- **GT_prob_init** (*numpy array (n_var, n_donor, n_GT)*) – Initial value of GT_prob, genotype probability per variant and donor

fit(*AD*, *DP*, *max_iter=200*, *min_iter=5*, *epsilon_conv=0.01*, *delay_fit_theta=0*, *verbose=True*, *n_inits=50*, *nproc=1*)

Fit Vireo model with coordinate ascent

Parameters

- **AD** (*scipy.sparse.csc_matrix (n_var, n_cell)*) – Sparse count matrix for alternative allele
- **DP** (*scipy.sparse.csc_matrix (n_var, n_cell)*) – Sparse count matrix for depths, alternative + refeerence alleles
- **max_iter** (*int*) – Maximum number of iterations
- **min_iter** – Minimum number of iterations

- **epsilon_conv** (*float*) – Threshold for detecting convergence
- **delay_fit_theta** (*int*) – Number of steps to delay updating theta. This can be very useful for common genetics when there is good prior on allelic ratio.
- **verbose** (*bool*) – Whether print out log info

set_initial(*beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*, *GT_prob_init=None*)
Set initial values

set_prior(*GT_prior=None*, *ID_prior=None*, *beta_mu_prior=None*, *beta_sum_prior=None*, *min_GP=1e-05*)
Set prior for key variables: theta, GT_prob and ID_prob. The priors are in the same shape as its according variables.

min_GP: float. Minimun genotype probability in GT_prior.

5.4.6 BinomMixtureVB Object

Objects of type *BinomMixtureVB* for clustering with binomial mixture model

class vireoSNP.**BinomMixtureVB**(*n_cell*, *n_var*, *n_donor*, *fix_beta_sum=False*, *beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*)
Binomial mixture model with variational inference

The prior can be set via set_prior() before fitting the model.

beta_mu: numpy array (n_var, n_donor) Beta mean parameter of theta's posterior

beta_sum: numpy array (n_var, n_donor) Beta concetration parameter of theta's posterior

ID_prob: numpy array (n_cell, n_donor) Posterior cell assignment probability to each donor

__init__(*n_cell*, *n_var*, *n_donor*, *fix_beta_sum=False*, *beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*)
Initialise Vireo model

Note, multiple initializations are highly recomended to avoid local optima.

Parameters

- **n_cell** (*int.*) – Number of cells
- **n_var** (*int.*) – Number of variants
- **n_donor** (*int.*) – Number of donors
- **fix_beta_sum** (*bool.*) – Whether fixing the concetration parameter of theta's posterior
- **beta_mu_init** (*numpy array (n_var, n_donor)*) – Initial value of beta_mu, the mean parameter of theta
- **beta_sum_init** (*numpy array (n_var, n_donor)*) – Initial value of beta_sum, the concetration parameter of theta
- **ID_prob_init** (*numpy array (n_cell, n_donor)*) – Initial value of ID_prob, cell assignment probability to each donor

fit(*AD*, *DP*, *n_init=10*, *max_iter=200*, *max_iter_pre=100*, *random_seed=None*, ***kwargs*)
Fit VB with multiple initializations

Parameters

- **AD** (*scipy.sparse.csc_matrix (n_var, n_cell)*) – Sparse count matrix for alternative allele
- **DP** (*scipy.sparse.csc_matrix (n_var, n_cell)*) – Sparse count matrix for depths, alternative + refeerence alleles
- **n_inits** (*int*) – Number of random initialisations to use
- **max_iter** (*int*) – Maximum number of iterations for _fit_BV() in best initial
- **max_iter_pre** (*int*) – Maximum number of iterations for _fit_BV() in multiple initials
- **min_iter** – Minimum number of iterations for _fit_BV()
- **epsilon_conv** (*float*) – Threshold for detecting convergence for _fit_BV()
- **verbose** (*bool*) – Whether print out log info for _fit_BV()
- **random_seed** (*None or int*) – Random seed in numpy.random for multiple initializations

set_initial(*beta_mu_init=None*, *beta_sum_init=None*, *ID_prob_init=None*)
Random initialization

set_prior(*ID_prior=None*, *beta_mu_prior=None*, *beta_sum_prior=None*)
Set prior for key variables: theta and ID_prob. The priors are in the same shape as its according variables.

5.5 History

5.5.1 Development on GitHub

- fix bug with GT_tag2 in *match_VCF_samples()* function

5.5.2 Release v0.5.8 (18/02/2023)

- fix issue with None in match() and match_SNPs() when w/ chr w/o chr have partial match
- minor optimise the codes for vireoSNP.utils.vcf_utils.parse_donor_GPb
- add a snp_gene_match() function
- fix issue in write_VCF() when there is no samples

5.5.3 Release v0.5.7 (24/03/2022)

- fix the issue when output_dir is not given
- add doc for read VCF files

5.5.4 Release v0.5.6 (07/04/2021)

- fix a bug for detecting unsupported genotyep tag
- add a wrap function to compare samples in two VCF files
- add doublet_logLikRatio in *donor_ids.tsv* for extra indicators of doublets

- update documentation with supporting notebook *vireoSNP_clones.ipynb*
- update API

5.5.5 Release v0.5.5 (28/03/2021)

- update notebook *vireoSNP_clones.ipynb*
- update API

5.5.6 Release v0.5.4 (28/03/2021)

- introduce log likelihood ratio for detecting ambient RNAs
- support ambient RNAs from a mixture of all donors or only two donors
- introduce multiple processes for multiple initializations
- introduce ELBO_gain for selecting variants
- For donor_ids.tsv, the doublet_prob change from sum to max

5.5.7 Release v0.5.3 (28/03/2021)

- support detection of ambient RNAs, alternative way for doublet detection

5.5.8 Release v0.5.0 (09/02/2021)

- support support numpy.ndarray and automatically change to sparse matrix
- fix the sign of KL
- fix a minor bug on –noDoublet setting
- add –cellRange to subset the input cells for less memory
- update anno_heat() plotting
- add get_confusion() for results comparison and plotting

5.5.9 Release v0.4.2 (14/11/2020)

- fix the donor names when N < donors_in_GT
- change the suggestion from cellSNP python to C version (cellsnp-lite)

5.5.10 Release v0.4.1 (18/05/2020)

- add likelihood ratio test the differential donor abundance in bulk RNA-seq data
- set the interpolation='none' for plt.imshow

5.5.11 Release v0.4.0 (19/04/2020)

- add vireoBulk for demultiplexing in bulk RNA-seq data

5.5.12 Release v0.3.2 (10/04/2020)

- support donor variant match between with and without "chr" prefix

5.5.13 Release v0.3.1 (25/03/2020)

- replace greed_match to optimal_match for aligning donors via genotype

5.5.14 Release v0.3.0 (23/03/2020)

- Rewrite the Vireo in the object-oriented way for easier upgrading and adding new features
- Now support fix the dispersion of the theta posterior distribution
- Change *delay_fit_theta* as an augument. It's often useful for donor deconvolution, but may not ideal for clonal inference, where theta can be very different from our expectation due to ASE or copy numbers

5.5.15 Release v0.2.3 (22/03/2020)

- Fix a minor bug in donor_select()

5.5.16 Release v0.2.2 (21/03/2020)

- Change GP_prob's shape from (n_var, n_GT, n_donor) to (n_var, n_donor, n_GT)
- Restructure the codes for further upgrading
- Minor fix the GT_plot xlim and ylim

5.5.17 Release v0.2.1 (30/01/2020)

- Fix a bug when the donors in the input GT is smaller than donors in the pooled scRNA-seq. The sample id is now corrected.

5.5.18 Release v0.2.0 (28/01/2020)

- Support SNP specific allelic ratio, namely theta parameters. Note, SNP based ASE mode requires a much stronger prior on theta to avoid overfitting, as each variant has very low number of reads.
- Change the default extraDonor to 0.
- Provide examples/vireoSNP_usage.ipynb for using vireoSNP as a Python module for general cell clustering based on allelic ratio.

5.5.19 Release v0.1.8 (29/10/2019)

- Further fix the bug when variants in donor genotype are not in cell vcf file

5.5.20 Release v0.1.7 (05/10/2019)

- Support donor genotype vcf file with different FORMAT for different variants

5.5.21 Release v0.1.6 (05/10/2019)

- Fix a bug when variants in donor genotype are not in cell vcf file

5.5.22 Release v0.1.5 (28/09/2019)

- Support genotype barcode generation

5.5.23 Release v0.1.4 (22/09/2019)

- Support that the case that input GT is larger than wanted *n_donor*
- Clarify the structure in vireo_flock: 1) warm-up for multiple initials or extra donors; 2) pre-step to subset or fill up the genotype prior; 3) the main run.
- Provide more options in the warm-up step to search donors from extra clusters. Before, it only uses the size of the donor. Now, the genotype distance can be used to search the K donors with furthest genotype distance.

5.5.24 Release v0.1.3 (30/08/2019)

- Support vartrix sparse matrices as input
- Change –amplifyK to –extraDonor for extra donors in initial search
- Fixed the bug for –noDoublet
- Fixed a bug for unassigned
- Minor update of figure output
- Updated the submoduals for easier import

5.5.25 Release v0.1.2 (15/07/2019)

- Support sparse matrices as input (for cellSNP directory with *-O*)
- Plot the distance between genotype probability between estimated samples
- Upgrade the manual, including the usage of simulation (readme in the simulation folder of GitHub repo)

5.5.26 Release v0.1.1 (30/06/2019)

- A completed version for all planned features
- Donor deconvolution with supporting multiple modes: 1) without genotype 2) with genotype for all samples 3) with genotype for part of the samples 4) with genotype but not confident
- Manual for installation, usage, and preprocessing
- Release test data sets
- vireoSNP is available on PyPI, try it *pip install vireoSNP*

5.5.27 Release v0.1.0 (24/06/2019)

- reimplementation of vireo in Python (orignal in cardelino R package)
- Initial release with limited features

5.6 Mito Clones

Using vireo for clonal reconstruction - mitochondrial mutations

Date: 15/03/2022

The mitochondrial mutations data set is extracted from Ludwig et al, Cell, 2019, the 9 variants used here are from Supp Fig. 2F (and main Fig. 2F).

Generally, you can use cellSNP-lite to genotype mitochondrial genomes and call clonal informed mtDNA variants with MQuad.

With the filtered variants at hand, we can use the `vireoSNP.BinomMixtureVB` class to reconstruct the clonality.

[1]:
```
import vireoSNP
import numpy as np
from scipy import sparse
from scipy.io import mmread
import matplotlib.pyplot as plt

print(vireoSNP.__version__)
```

```
0.5.6
```

[2]:
```
# np.set_printoptions(formatter={'float': lambda x: format(x, '.5f')})
```

[3]:
```
AD = mmread("../data/mitoDNA/cellSNP.tag.AD.mtx").tocsc()
DP = mmread("../data/mitoDNA/cellSNP.tag.DP.mtx").tocsc()
# mtSNP_ids = np.genfromtxt('../data/mitoDNA/passed_variant_names.txt', dtype='str')
```

5.6.1 Identify clones

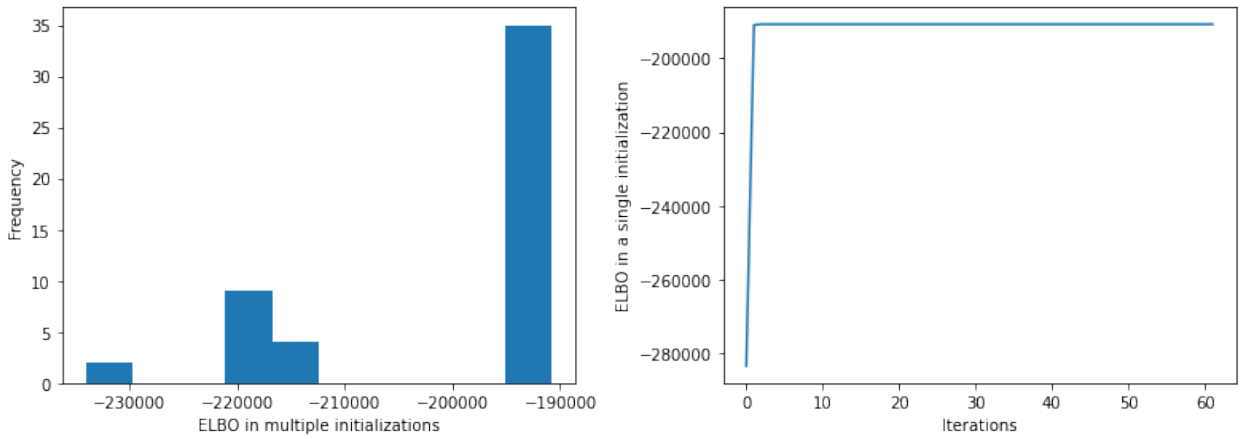
[4]:
```
from vireoSNP import BinomMixtureVB
```

[5]:
```
_model = BinomMixtureVB(n_var=AD.shape[0], n_cell=AD.shape[1], n_donor=3)
_model.fit(AD, DP, min_iter=30, n_init=50)
print(_model.ELBO_iters[-1])
```

```
-190779.74335041404
```

Check the model fitting

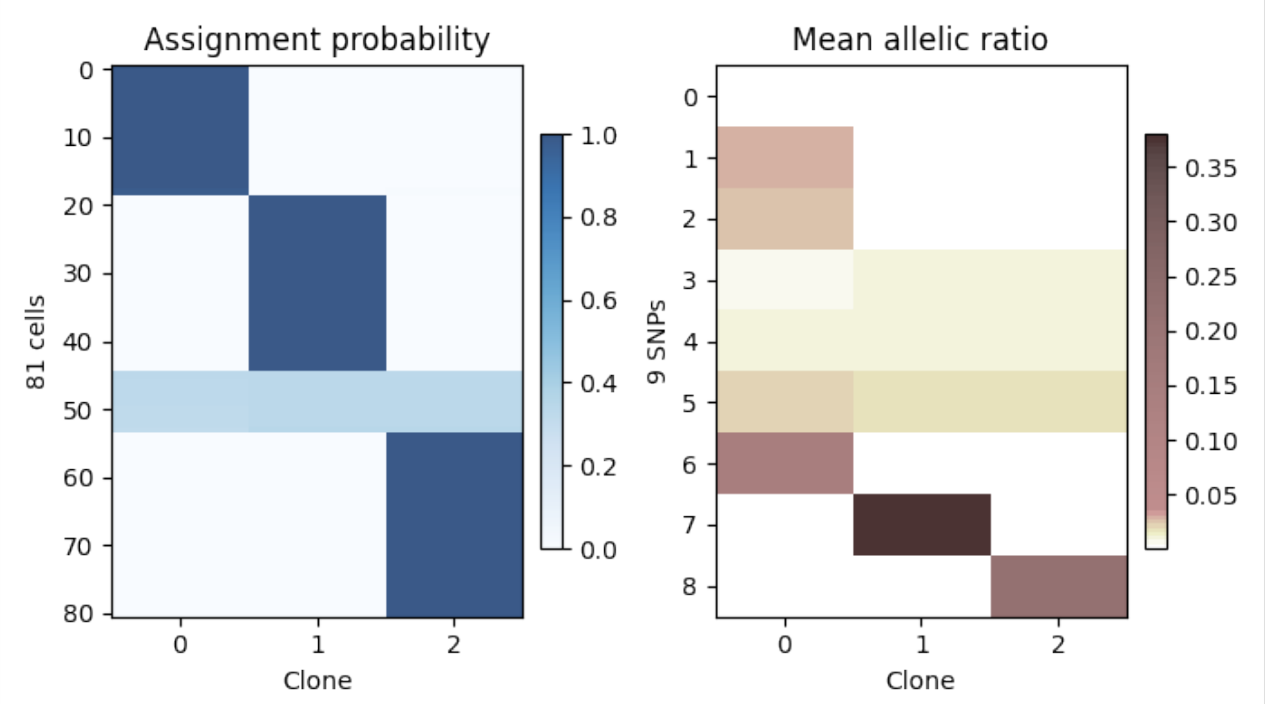
[6]:
```
fig = plt.figure(figsize=(11, 4))
plt.subplot(1, 2, 1)
plt.hist(_model.ELBO_inits)
plt.ylabel("Frequency")
plt.xlabel("ELBO in multiple initializations")

plt.subplot(1, 2, 2)
plt.plot(_model.ELBO_iters)
plt.xlabel("Iterations")
plt.ylabel("ELBO in a single initialization")

plt.tight_layout()
plt.show()
```



Visualize assignment probability and allele frequency

[7]:
```
# In mitochondrial, allele frequency is highly informative between 0.01 to 0.1,
# so we rescale the colour to give more spectrum for this region.
# You can design/choose your own colors from here:
# https://matplotlib.org/stable/tutorials/colors/colormaps.html

from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap

raw_col = cm.get_cmap('pink_r', 200)
new_col = np.vstack((raw_col(np.linspace(0, 0.7, 10)),
                     raw_col(np.linspace(0.7, 1, 90))))
segpink = ListedColormap(new_col, name='segpink')
```

[8]:
```
from vireoSNP.plot import heat_matrix

fig = plt.figure(figsize=(7, 4), dpi=100)
plt.subplot(1, 2, 1)
im = heat_matrix(_model.ID_prob, cmap="Blues", alpha=0.8,
                 display_value=False, row_sort=True)
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.title("Assignment probability")
plt.xlabel("Clone")
plt.ylabel("%d cells" %(_model.n_cell))
plt.xticks(range(_model.n_donor))


plt.subplot(1, 2, 2)
im = heat_matrix(_model.beta_mu, cmap=segpink, alpha=0.8,
                 display_value=False, row_sort=True)
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.title("Mean allelic ratio")
plt.xlabel("Clone")
plt.ylabel("%d SNPs" %(_model.n_var))
plt.xticks(range(_model.n_donor))

plt.tight_layout()
plt.show()
# plt.savefig("you_favorate_path with png or pdf")
```



5.6.2 Diagnosis

We are using multiple initializations via `n_init` and choose the one with highest ELBO. However, this doesn't guarantee to be the global optima. To double check it, we can run the same scripts multiple time (without fixed random seed), and check if the same (best) ELBO is found.

If yes, it is likely to be the global optima, otherwise, we need increase `n_init`, e.g., 300 for more search.

[9]:
```
n_init = 50
for i in range(3):
    _model = BinomMixtureVB(n_var=AD.shape[0], n_cell=AD.shape[1], n_donor=3)
    _model.fit(AD, DP, min_iter=30, n_init=n_init)
    print("rerun %d:" %i, _model.ELBO_iters[-1])
```

```
rerun 0: -190779.74335041404
rerun 1: -190779.74335041404
rerun 2: -190779.74335041404
```

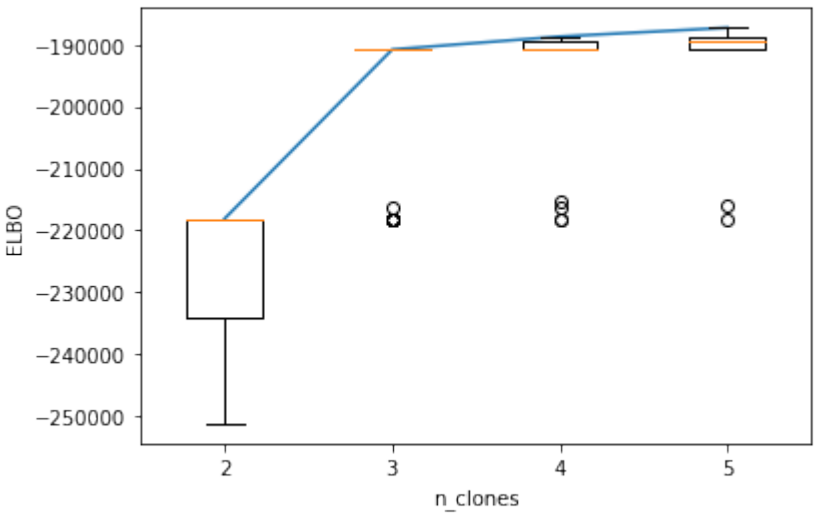
[ ]:
```
```

It is generally difficult to identify the number of clones, which is a balance between subclone resolution and analysis reliability. More clones maybe preferred, but there could be higher risk that the subclones are not genuine but rather technical noise.

Here, we could use ELBO for different number of clones as an indictor for model selection. However, this is still imperfect. One empirical suggestion is to choose the `n_clones` when ELBO stops increasing dramatically, for example in the case below, we will pick 3 clones.

[10]:
```
n_init = 50
n_clone_list = np.arange(2, 6)

_ELBO_mat = []
for k in n_clone_list:
    _model = BinomMixtureVB(n_var=AD.shape[0], n_cell=AD.shape[1], n_donor=k)
    _model.fit(AD, DP, min_iter=30, n_init=n_init)
    _ELBO_mat.append(_model.ELBO_inits)
```

[11]:
```
plt.plot(np.arange(1, len(n_clone_list)+1), np.max(_ELBO_mat, axis=1))
plt.boxplot(_ELBO_mat)
plt.xticks(np.arange(1, len(n_clone_list)+1), n_clone_list)
plt.ylabel("ELBO")
plt.xlabel("n_clones")
plt.show()
```



[ ]:

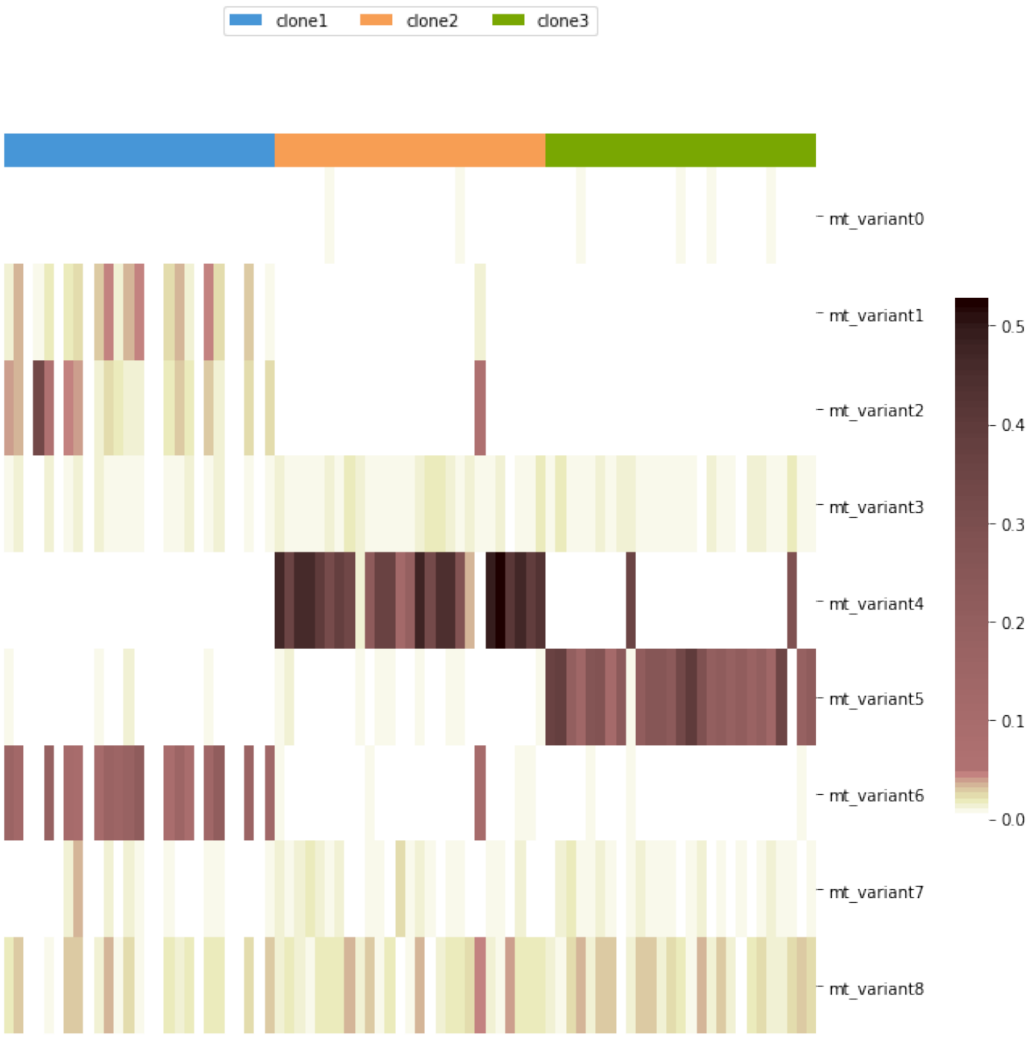
5.6.3 Visualization

If you want to visualise the raw allele frequency with annotation of cells, you may consider seaborn.clustermap. We also wrap this function here as `vireoSNP.plot.anno_heat` for quick use.

[12]:
```
mtSNP_ids = ['mt_variant%d' %x for x in range(AD.shape[0])]
cell_label = np.array(['clone1'] * 27 + ['clone2'] * 27 + ['clone3'] * 27)
id_uniq = ['clone1', 'clone2', 'clone3']
```

[13]:
```
vireoSNP.plot.anno_heat(AD/DP, col_anno=cell_label, col_order_ids=id_uniq,
                        cmap=segpink, yticklabels=mtSNP_ids)
```

[13]:
```
<seaborn.matrix.ClusterGrid at 0x7fd475067eb0>
```



[ ]:

Python Module Index

v

Index

Symbols

A

B

F

H

L

M

P

R

S

V