BRIE Documentation

Release 2.0.5

Yuanhua Huang

Jun 30, 2021

Main

CHAPTER 1

About BRIE

Welcome to the new BRIE2 (Bayesian regression for isoform estimate, v2), a scalable Bayesian method to robustly identify splicing phenotypes in single cells RNA-seq designs and accurately estimate isoform proportions and its uncertainty.

BRIE2 supports isoform quantification for different needs:

1. cell features: informative prior is learned from shared cell processes. It also allows to effectively detect splicing phenotypes by using Evidence Lower Bound gain, an approximate of Bayes factor.
2. gene features: informative prior is learned from shared gene regulatory features, e.g., sequences and RNA protein binding
3. no feature: use zero-mean logit-normal as uninformative prior, namely merely data deriven

Note, BRIE1 CLI is still available in this version but changed to *brie1* and *brie1-diff*.

CHAPTER 2

Questions or Bugs

If you find any error or suspicious bug, we will appreciate your report. Please write them in the github issues: https://github.com/huangyh09/brie/issues

If you have questions on using BRIE, feel free get in touch with us: yuanhua <at> hku.hk

CHAPTER 3

Quick Resources

Code: GitHub latest version https://github.com/huangyh09/brie

Data: splicing events annotations http://sourceforge.net/projects/brie-rna/files/annotation/

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

Issue reports https://github.com/huangyh09/brie/issues

CHAPTER 4

References

Yuanhua Huang and Guido Sanguinetti. BRIE: transcriptome-wide splicing quantification in single cells. **Genome Biology**, 2017; 18(1):123.

4.1 Installation

4.1.1 Environment setting

We recommend to create a separated conda environment for running BRIE2, which heavily depends on TensorFlow and TensorFlow-probability.

```
conda create -n TFProb python=3.7
```

replace `-n TFProb` with `-p ANY_PATH/TFProb` to specify the path for conda environment. Then activate the environment by `conda activate TFProb` or the full path, before install more packages.

4.1.2 Easy install

BRIE2 is available through pypi. To install, type the following command line, and add `-U` for upgrading:

```
pip install -U brie
```

Alternatively, you can install from this GitHub repository for latest (often development) version by following command line

```
pip install -U git+https://github.com/huangyh09/brie
```

In either case, if you don't have write permission for your current Python environment, add `--user`, but check the previous section on create your own conda environment.

4.1.3 GPU usage

With TensorFlow backend, BRIE2 can benefit from using GPUs. Here is one way to set up GPU configurations with NVIDIA GPU on Ubuntu:

```
pip install -U tensorflow-gpu
conda install -c anaconda cupti
conda install -c anaconda cudnn
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/extras/CUPTI/lib64
```

For more information on GPU configuration, please refer to the Tensorflow documentation, or anaconda GPU.

Note: At the moment, TensorFlow calls all available GPUs, which is not nessary. You can specify the card you want to use by add the following variable before you command line `CUDA_VISIBLE_DEVICES=3 brie-quant -i my_count.h5ad`

4.1.4 Test

In order to test the installation, you could type `brie-quant`. If successful, you will see the following output.

```
Welcome to brie-quant in BRIE v2.0.2!

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

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

```
brie-quant: command not found
```

Usually the directory is `~/.local/bin` if you don't use Anaconda. You could add the path into PATH environment variable, by write the following line into `.profile` or `.bashrc` file.

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

If you have any issue, please report it to the issue on brie issues.

4.2 Quick start

BRIE estimates the isoform proportion for two-isoform events across many single cells. For getting started quickly, there are two main steps to go. A demo file is available at brie2_demo.sh. Sepcifically, you can follow the steps below.

4.2.1 Step1: read counts

First, you need to count the isoform specific reads in each splicing events in each cell. For alternative splicing, e.g., exon-skipping event, you can download the splicing annotations generated by us or make your own, e.g., with briekit.

Then you can use the `brie-count` fetch the read count tensor, which will be stored in hdf5 format as AnnData. See more details on brie-count CLI, and you can use this example command line.

```
brie-count -a AS_events/SE.gold.gtf -S sam_and_cellID.tsv -o out_dir -p 15
```

Besides the SE event, other types of alternative splicing, e.g., intron retaining is also applicable with BRIE. Some pre-processing utilities will be available soon.

4.2.2 Step2: quantification

Step2.1: isoform quantification

Second, you can quantify the isoform with cell / gene or none features. Usually, we recommend to use aggregated imputation even if you don't have any feature, namely `mode 2` in brie-quant CLI as follows (please add `--interceptMode gene`),

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_aggr.h5ad --interceptMode␣
↪gene
```

Step2.2 phenotype detection

If you have cell level features, e.g., disease condition or cell type or continuous variable, you can use it in cell features to detect variable splicing events as phenotypes for further analysis. This is `mode 3` in `brie-quant`, so requires `-c` and `--LRTindex`

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_cell.h5ad \
    -c $DATA_DIR/cell_info.tsv --interceptMode gene --LRTindex=All
```

4.2.3 Step3: downstream analysis

The BRIE output AnnData files are compatible with Scanpy, hence you can easily use it for dimension reduction, clustering, and other visualization. Example analyses are coming soon.

4.3 brie-count CLI

BRIE (>=2.0.0) provids two CLI directly available in your Python path: `brie-count`, `brie-quant`.

4.3.1 Splicing annotations

As input, it requires a list of annotated splicing events. We have generated annotations for human and mouse. If you are using it, please align RNA-seq reads to the according version (or close version) of reference genome. Alternatively, you can use briekit package to generate.

4.3.2 Read counting

The `brie-count` CLI calculates a count tensor for the number of reads that are aligned to each splicing event and each cell, and stratifies four them into four different categories in :

1. key 1: Uniquely aligned to isoform1, e.g., in exon1-exon2 junction in SE event
2. key 2: Uniquely aligned to isoform2, e.g., in exon1-exon3 junction in SE event
3. key 3: Ambiguously aligned to isoform1 and isoform2, e.g., within exon1

4. key 0: Partially aligned in the region but not compatible with any of the two isoforms. We suggest ignoring these reads.

Then you fetch the counts on a list of bam files by the command line like this:

```
brie-count -a AS_events/SE.gold.gtf -S sam_and_cellID.tsv -o out_dir -p 15
```

By default, you will have four output files in the out_dir: `brie_count.h5ad`, `read_count.mtx.gz`, `cell_note.tsv.gz`, and `gene_note.tsv.gz`. The `brie_count.h5ad` contains all information for downstream analysis, e.g., for *brie-quant*.

4.3.3 Options

There are more parameters for setting (`brie-count -h` always give the version you are using):

```
Usage: brie-count [options]

Options:
  -h, --help            show this help message and exit
  -a GFF_FILE, --gffFile=GFF_FILE
                        GTF/GFF3 file for gene and transcript annotation
  -S SAMLIST_FILE, --samList=SAMLIST_FILE
                        A tsv file containing sorted and indexed bam/sam/cram
                        files. No header line; file path and cell id (optional)
  -o OUT_DIR, --out_dir=OUT_DIR
                        Full path of output directory [default: $samList/brieCOUNT]

  Optional arguments:
    -p NPROC, --nproc=NPROC
                        Number of subprocesses [default: 4]
```

4.4 brie-quant CLI

The `brie-quant` CLI (in brie>=2.0.0) uses the newly developed variational inference methods scalable to large data sets, which works both in CPU or GPU with the TensorFlow Backend. For using BRIE1 (<=0.2.4) with MCMC sampler, please refer to BRIE1.

This command allows to quantify the splicing isoform proportion Psi and detect variable splicing event along with cell level features, e.g., cell type, disease condition, development time.

As a Bayesian method, the key philosophy of BRIE is to combine likelihood (data driven) and prior (uninformative or informative). In BRIE2, a variety of prior settings are supported, as follows.

4.4.1 Mode 1: None imputation

In this mode, the prior is uninformative logit-normal distribution with mean=0, and learned variance. Therefore, if a splicing event in a gene doesn't have any read, it will return a posterior with Psi's mean=0.5 and 95% confidence interval around 0.95 (most case >0.9).

This setting is used if you have high covered data and you only want to calculate cells with sufficient reads for each interesting genes, e.g., by filtering out all genes with Psi_95CI > 0.3.

Otherwise, the 0.5 imputed genes will be confounded by the expression level, instead of the isoform proportion.

Example command line for mode 1:

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_pure.h5ad --interceptMode␣
↪None
```

4.4.2 Mode 2: Aggregated imputation

This mode requires argument `--interceptMode gene`. It aims to learn a prior shared by all cells on each gene. The benefit for this mode is that dimension reduction can be performed, e.g., PCA and UMAP on splicing. As there are many splicing events that are not well covered, it has a high variance in the estimation, and is often suggested filtered out, which will cause missing values. Based on the cell aggregated imputation, most dimension reduction methods can be used, even it doesn't support missing values.

Example command line for mode 2:

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_aggr.h5ad --interceptMode␣
↪gene
```

4.4.3 Mode 3: Variable splicing detection

This mode requires argument `-c` for cell features and `--LRTindex` for the index (zero-based) of cell features to perform likelihood ratio test. Again we suggest to keep the cell aggregation on each gene by `--interceptMode gene`.

Then this mode will learn a prior from the given cell level features and perform the second fit by leaving each feature out to calculate the EBLO gain, which can be further used as likelihood ratio test.

Example command line for mode 3:

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_cell.h5ad \
    -c $DATA_DIR/cell_info.tsv --interceptMode gene --LRTindex=All
```

4.4.4 Flexible settings

There could be more flexible settings, for example only use gene features as in BRIE1 by the following command:

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_gene.h5ad \
    -g $DATA_DIR/gene_seq_features.tsv --interceptMode cell --LRTindex=All
```

Or use both gene features and cell features

```
brie-quant -i out_dir/brie_count.h5ad -o out_dir/brie_quant_all.h5ad \
    -c $DATA_DIR/cell_info.tsv -g $DATA_DIR/gene_seq_features.tsv \
    --interceptMode gene --LRTindex=All
```

There are more parameters for setting (`brie-quant -h` always give the version you are using):

```
Usage: brie-quant [options]

Options:
  -h, --help            show this help message and exit
  -i IN_FILE, --inFile=IN_FILE
                        Input read count matrices in AnnData h5ad or brie npz
                        format.
```

(continues on next page)

(continued from previous page)

```
  -c CELL_FILE, --cellFile=CELL_FILE
                        File for cell features in tsv[.gz] with cell and
                        feature ids.
  -g GENE_FILE, --geneFile=GENE_FILE
                        File for gene features in tsv[.gz] with gene and
                        feature ids.
  -o OUT_FILE, --out_file=OUT_FILE
                        Full path of output file for annData in h5ad [default:
                        $inFile/brie_quant.h5ad]
  --LRTindex=LRT_INDEX  Index (0-based) of cell features to test with LRT:
                        All, None or comma separated integers [default: None]
  --interceptMode=INTERCEPT_MODE
                        Intercept mode: gene, cell or None [default: None]
  --layers=LAYERS       Comma separated layers two or three for estimating Psi
                        [default: isoform1,isoform2,ambiguous]

  Gene filtering:
    --minCount=MIN_COUNT
                        Minimum total counts for fitltering genes [default:
                        50]
    --minUniqCount=MIN_UNIQ_COUNT
                        Minimum unique counts for fitltering genes [default:
                        10]
    --minCell=MIN_CELL  Minimum number of cells with unique count for
                        fitltering genes [default: 30]
    --minMIF=MIN_MIF    Minimum minor isoform frequency in unique count
                        [default: 0.001]

  VI Optimization:
    --MCsize=MC_SIZE    Sample size for Monte Carlo Expectation [default: 3]
    --minIter=MIN_ITER  Minimum number of iterations [default: 5000]
    --maxIter=MAX_ITER  Maximum number of iterations [default: 20000]
    --batchSize=BATCH_SIZE
                        Element size per batch: n_gene * total cell [default:
                        500000]
```

4.5 Release notes

4.5.1 Release v2.0.5 (04/11/2020)

- Support saving detection table to tsv file
- Add the exon start and stop positions in brie-count

4.5.2 Release v2.0.4 (26/09/2020)

- Tune the learning rate with multiple values
- For test model, the fitted parameters will be used as initials
- Support base model with full features or null feature
- For gene feature only, switch sigma into per cell base
- Add noise term in simulator

- A few minor bug fix
- Implement a Inv-Gamma prior distribution for sigma (not in use)

4.5.3 Release v2.0.3 (26/08/2020)

- Support to use minimum minor isoform frequency to fileter genes (default=0.001)
- Add pseudo count (default=0.01) for none-empty element in both unique counts for more robust results
- Reduce the sample size for Monte Carlo Expectation (10 to 3) for computational efficiency
- Restructure the arguments in brie-quant
- Initialise the example notebook on multiple sclerosis data

4.5.4 Release v2.0.2 (24/08/2020)

- Fix minor bugs in brei-count and brie-quant cli for compatibility

4.5.5 Release v2.0.0 (23/08/2020)

- Change the whole BRIE model from MCMC sampler (v1) to variational inference (v2)
- Change the usage of each read to a summarised read counts for speedup
- Support splicing quantification without any feature or cell features or gene features or both types of features.
- Support detection of variable splicing event associated with cell features
- Support acceleration with Graphic card (Nvidia GPU)
- Compatible with Scanpy analysis suite with a variety of plotting functions
- Restructure the whole package
- BRIE earlier version is still avaible with *brie1*

4.5.6 Release v0.2.4 (21/10/2019)

- fix a bug that fragment length longer than transcript length

4.5.7 Release v0.2.2 (15/01/2019)

- move __version__ into a separate file, so import brie is not required before installation.
- support cram file as input aligned reads

4.5.8 Release v0.2.0 (03/06/2018)

- split the pre-processing functions in BRIE to another separate package BRIE-kit, as some functions in the pre-processing require Python2 environment. So, *BRIE* will keep two functions *brie* and *brie-diff*, while *BRIE-kit* will have *briekit-event*, *briekit-event-filter*, and *briekit-factor*. See BRIE-KIT: https://github.com/huangyh09/briekit/wiki

4.5.9 Release v0.1.5 (03/06/2018)

- support gff in gz format
- add an example with 130 cells
- *brie-diff* supporting ranking genes

4.5.10 Release v0.1.4 (02/06/2018)

- turn off pylab
- fix a bug for function *rev_seq*, reported in issue #10
- update documentation

4.5.11 Release v0.1.3 (16/04/2017)

- `brie-diff` takes multiple cells, which could handle pair-wise comparisons for 100 cells in ~10min with 30 CPUs; and 1000 cells within a few hours.
- Simulation wraps on Spanki are provided for simulating splicing events at different coverages or drop-out probability and drop-out rate for single cells: https://github.com/huangyh09/brie/tree/master/simulator

4.5.12 Release v0.1.2 (13/01/2017)

- support Python 3.x now
- do not depend on h5py anymore for hdf5 data storage.
- `brie-factor` returns xxx.csv.gz rather than xxx.h5
- `brie` returns sample.csv.gz rather than sample.h5
- `brie-diff` takes sample.csv.gz rather than sample.h5

4.5.13 Release v0.1.1 (02/01/2017)

- change licence to Apache License v2
- update `brie-event-filter`

4.5.14 Release v0.1.0 (29/12/2016)

- Initial release of BRIE

4.6 Multiple Sclerosis

This data set was generated by Falcão et al, 2018 containing 2,208 mouse cells probed by SMART-seq2 with half experimental autoimmune encephalomyelitis (EAE) cells (in mimicing multiple scleorsis) and half control cells.

Here, we will illustrate how BRIE2 can be used to effectively detect differential splicing events between EAE and control cells, and using splicing ratio to predict cell type and disease conditions.

We have pre-processed the data with these BRIE2 scripts with `brie-count` and `brie-quant`. You can download the pre-computed data, e.g., by using the following command line and unzip it into the `./data` folder for running this notebook.

```
wget http://ufpr.dl.sourceforge.net/project/brie-rna/examples/msEAE/brie2_msEAE.zip
unzip -j brie2_msEAE.zip -d ./data
```

4.6.1 Load Packages

[1]:
```
import umap
```

[2]:
```
import os
import brie
import numpy as np
import pandas as pd
import scanpy as sc
import matplotlib.pyplot as plt

print(brie.__version__)
```

```
2.0.5
```

sc.settings.verbosity = 3 verbosity: errors (0), warnings (1), info (2), hints (3) sc.logging.print$_versions()sc.settings.set_figure_params(dpi = 60)$

[3]:
```
# define the path you store the example data
# dat_dir = "./data"
dat_dir = "/storage/yhhuang/research/brie2/releaseDat/msEAE/"
```

4.6.2 BRIE2 option 1: differential splicing events

Detecting differential splicing event by regressing on the EAE state with considering the strain covariate.

The command line is available at run_brie2.sh. Here is an example of design matrix besides the intercept and `isEAE` as the only testing variable.

```
samID   isEAE   isCD1
SRR7102862      0       1
SRR7103631      1       0
SRR7104046      1       1
SRR7105069      0       0
```

Besides the big .h5ad file, you can use the detected splicing events directely from brie_quant_cell.brie_ident.tsv. However, if you want get the quantify `Psi` for downstream analysis, e.g., option2 below, you need load the `adata.layers['Psi']`, and `adata.layers['Psi_95CI']`.

[4]:
```
adata = sc.read_h5ad(dat_dir + "/brie_quant_cell.h5ad")
adata
```

[4]:
```
AnnData object with n_obs × n_vars = 2208 × 3780
    obs: 'samID', 'samCOUNT'
    var: 'GeneID', 'GeneName', 'TranLens', 'TranIDs', 'n_counts', 'n_counts_uniq',
↪'loss_gene'
    uns: 'Xc_ids', 'brie_losses', 'brie_param', 'brie_version'
```

(continues on next page)

(continued from previous page)

```
    obsm: 'Xc'
    varm: 'ELBO_gain', 'cell_coeff', 'effLen', 'fdr', 'intercept', 'p_ambiguous',
↪'pval', 'sigma'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'ambiguous', 'isoform1', 'isoform2', 'poorQual
↪'
```

[5]:
```
adata.uns['brie_param']
```

[5]:
```
{'LRT_index': array([0]),
 'base_mode': 'full',
 'intercept_mode': 'gene',
 'layer_keys': array(['isoform1', 'isoform2', 'ambiguous'], dtype=object),
 'pseudo_count': 0.01}
```

Change gene index from Ensemebl id to gene name

[6]:
```
adata.var['old_index'] = adata.var.index
new_index = [adata.var.GeneName[i] + adata.var.GeneID[i][18:] for i in range(adata.
↪shape[1])]
adata.var.index = new_index
```

Add the cell annotations from input covariates

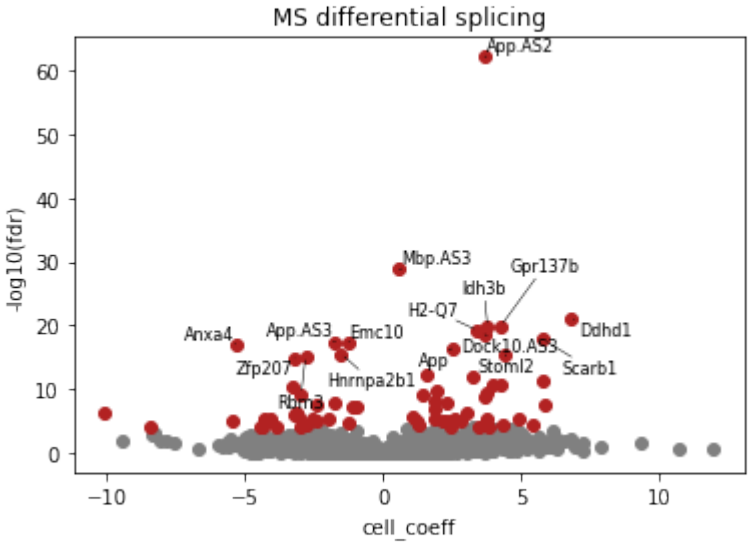
[7]:
```
adata.obs['MS'] = ['EAE' if x else 'Ctrl' for x in adata.obsm['Xc'][:, 0]]
adata.obs['isCD1'] = ['CD1_%d' %x for x in adata.obsm['Xc'][:, 1]]
```

Volcano plot

Here, BRIE2 detects the differential splicing events between EAE and control. Cell_coeff means the effect size on logit(Psi). Positive value means higher Psi in isEAE.

[8]:
```
## volcano plot for differential splicing events

brie.pl.volcano(adata, y='fdr', n_anno=16, adjust=True)
plt.title("MS differential splicing")
```

[8]:
```
Text(0.5, 1.0, 'MS differential splicing')
```



[9]:
```
DSEs = adata.var.index[adata.varm['fdr'][:, 0] < 0.05]
len(DSEs), len(np.unique([x.split('.')[0] for x in DSEs])), adata.shape
```

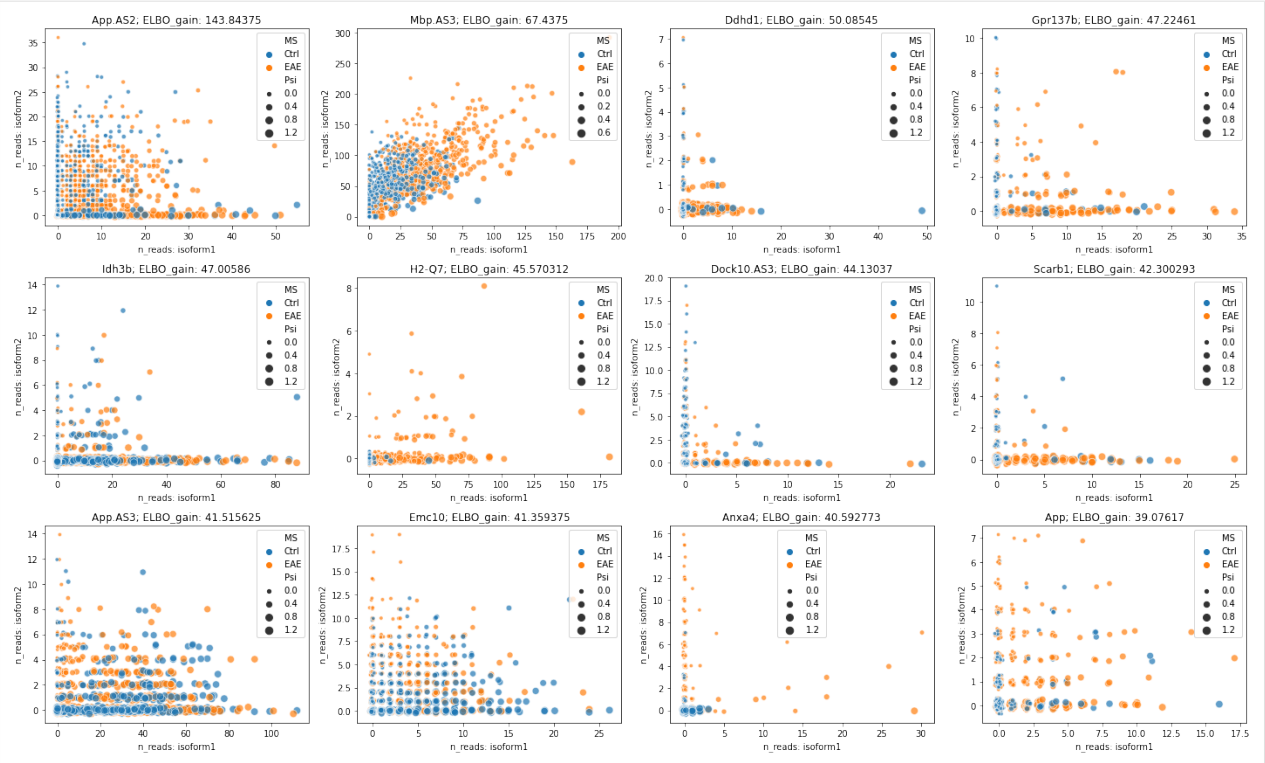
[9]:
```
(368, 348, (2208, 3780))
```

[10]:
```
adata.uns['Xc_ids']
```

[10]:
```
array(['isEAE', 'isCD1'], dtype=object)
```

Visualize raw counts for DSEs

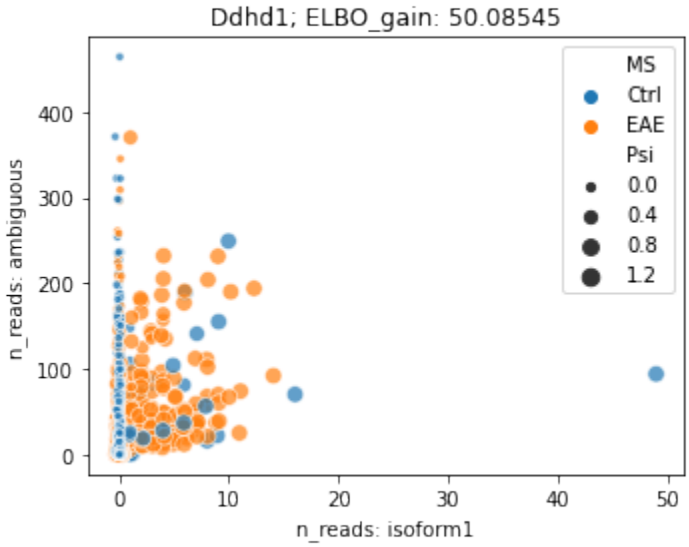
[11]:
```
rank_idx = np.argsort(adata.varm['ELBO_gain'][:, 0])[::-1]

fig = plt.figure(figsize=(20, 12))
brie.pl.counts(adata, genes=list(adata.var.index[rank_idx[:12]]),
               color='MS', add_val='ELBO_gain', nrow=3, alpha=0.7)
# fig.savefig(dat_dir + '../../figures/fig_s8_counts.png', dpi=300, bbox_inches='tight
↪')
plt.show()
```



Changed ratio between ambiguous reads

Some differential splicing events having changed ratio on ambigous reads and isoform specific reads

[12]:
```
### Differential splicing shown by isoform1 and ambiguous reads

fig = plt.figure(figsize=(5, 4))
brie.pl.counts(adata, genes='Ddhd1',
               color='MS', add_val='ELBO_gain',
               layers=['isoform1', 'ambiguous'],
               nrow=1, alpha=0.7)
plt.show()
```



[ ]:

4.6.3 BRIE2 option 2: splicing quantification and usage

If you aim to use splicing isoform proportions as phenotypes for downstream analysis, e.g., to identify cell types, or study its disease contribution, we suggest not including the cell feature as covariate when quantifying `Psi` to avoid circular analysis.

Instead you could use gene level features, e.g., sequence features we introduced in BRIE1, or you can use an aggregated value from all cells as an informative prior. Similar as BRIE1, this prior is learned from the data and has a logit-normal distribution, so the information prior usually won't be too strong.

In general, we recommend using the aggregation of cells as the prior thanks to its simplicity if it does not violate your biological hypothesis.

[13]:
```
adata_aggr = sc.read_h5ad(dat_dir + "/brie_quant_aggr.h5ad")

adata_aggr
```

[13]:
```
AnnData object with n_obs × n_vars = 2208 × 3780
    obs: 'samID', 'samCOUNT'
    var: 'GeneID', 'GeneName', 'TranLens', 'TranIDs', 'n_counts', 'n_counts_uniq'
    uns: 'brie_losses'
    obsm: 'Xc'
    varm: 'cell_coeff', 'effLen', 'intercept', 'p_ambiguous', 'sigma'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'ambiguous', 'isoform1', 'isoform2', 'poorQual
↪'
```

[14]:
```
## change gene index
print(np.mean(adata.var['old_index'] == adata_aggr.var.index))

adata_aggr.var.index = adata.var.index
```

```
1.0
```

Add meta data and gene-level annotation

Cell annotations and UMAP from gene-level expression. You can add any additional annotation.

[15]:
```
print(np.mean(adata.obs.index == adata_aggr.obs.index))
adata_aggr.obs['MS'] = adata.obs['MS'].copy()
adata_aggr.obs['isCD1'] = adata.obs['isCD1'].copy()
```

```
1.0
```

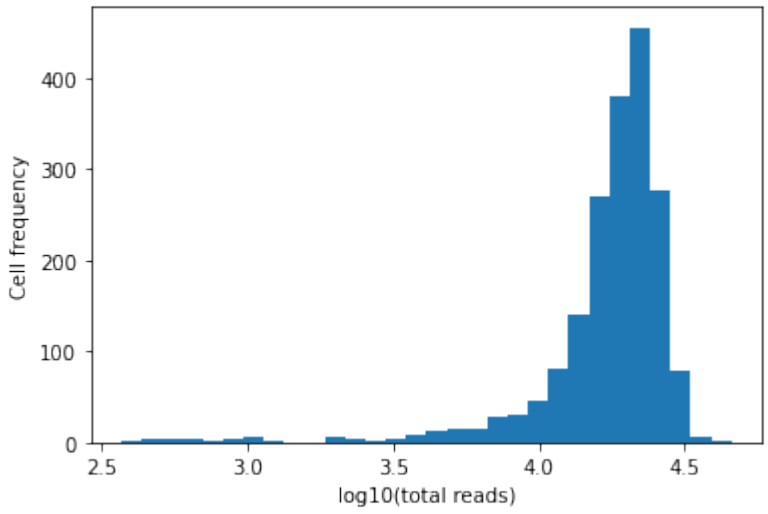
[16]:
```
dat_umap = np.genfromtxt(dat_dir + '/cell_X_umap.tsv', dtype='str', delimiter='\t')

mm = brie.match(adata_aggr.obs.index, dat_umap[:, 0])
idx = mm[mm != None].astype(int)

adata_aggr = adata_aggr[mm != None, :]
adata_aggr.obsm['X_GEX_UMAP'] = dat_umap[idx, 1:3].astype(float)
adata_aggr.obs['cluster'] = dat_umap[idx, 3]
adata_aggr.obs['combine'] = [adata_aggr.obs['cluster'][i] + '-' + adata_aggr.obs['MS
↪'][i]
                             for i in range(adata_aggr.shape[0])]
```

[17]:
```
adata_aggr
```

[17]:
```
AnnData object with n_obs × n_vars = 1882 × 3780
    obs: 'samID', 'samCOUNT', 'MS', 'isCD1', 'cluster', 'combine'
    var: 'GeneID', 'GeneName', 'TranLens', 'TranIDs', 'n_counts', 'n_counts_uniq'
    uns: 'brie_losses'
    obsm: 'Xc', 'X_GEX_UMAP'
    varm: 'cell_coeff', 'effLen', 'intercept', 'p_ambiguous', 'sigma'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'ambiguous', 'isoform1', 'isoform2', 'poorQual
↪'
```

Cell filtering

[18]:
```
plt.hist(np.log10(adata_aggr.X.sum(1)[:, 0] + 1), bins=30)
plt.xlabel("log10(total reads)")
plt.ylabel("Cell frequency")
plt.show()
```

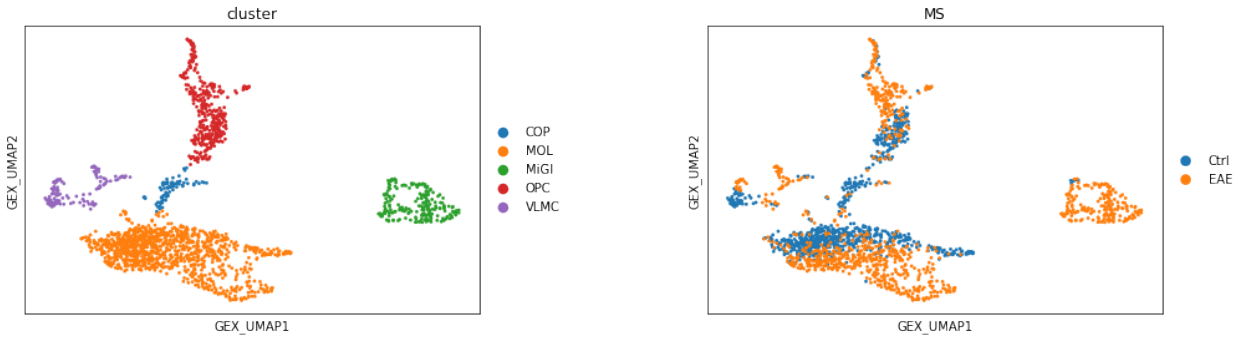


[19]:
```
# min_reads = 15000
min_reads = 3000

adata_aggr = adata_aggr[adata_aggr.X.sum(axis=1) > min_reads, :]
adata = adata[adata_aggr.obs.index, :]
```

4.6.4 Visualizing splicing phenotypes in Gene expression UMAP

[20]:
```
sc.pl.scatter(adata_aggr, basis='GEX_UMAP', color=['cluster', 'MS'], size=30)
```

```
/home/yuanhua/.conda/envs/TFProb/lib/python3.7/site-packages/anndata/_core/anndata.py:
↪1210: ImplicitModificationWarning: Initializing view as actual.
  "Initializing view as actual.", ImplicitModificationWarning
Trying to set attribute `.obs` of view, copying.
... storing 'MS' as categorical
Trying to set attribute `.obs` of view, copying.
... storing 'isCD1' as categorical
Trying to set attribute `.obs` of view, copying.
... storing 'cluster' as categorical
Trying to set attribute `.obs` of view, copying.
... storing 'combine' as categorical
```

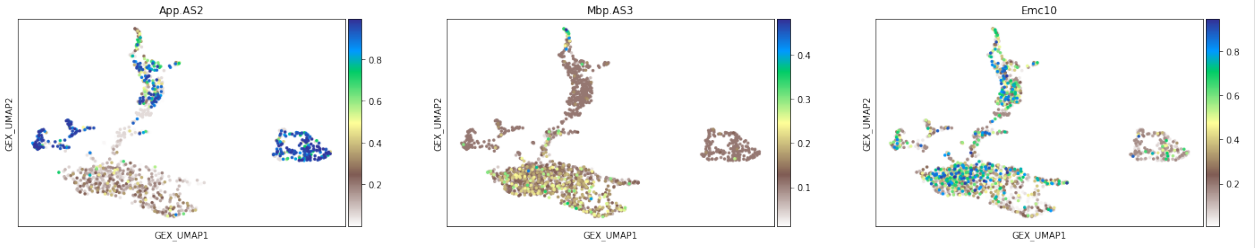


[21]:
```
sc.pl.scatter(adata_aggr, basis='GEX_UMAP', layers='Psi',
              color=['App.AS2', 'Mbp.AS3', 'Emc10'],
```

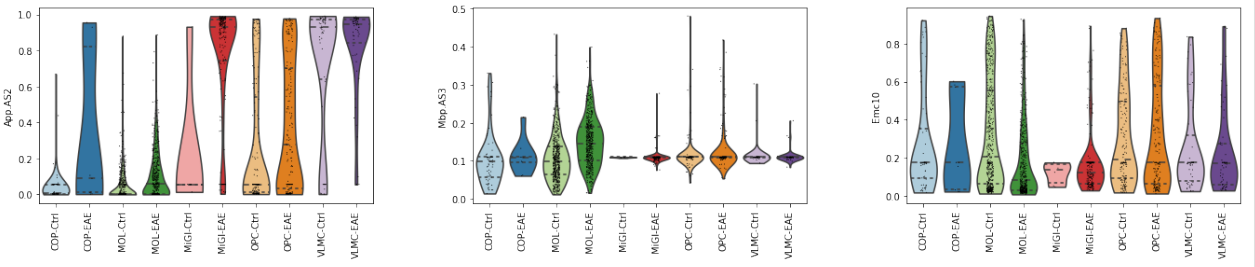
(continues on next page)

(continued from previous page)

```
                size=50, color_map='terrain_r') #'terrain_r'
```



[22]:
```
import seaborn as sns

sc.pl.violin(adata_aggr, ['App.AS2', 'Mbp.AS3', 'Emc10'],
             layer='Psi', groupby='combine', rotation=90,
             inner='quartile', palette=sns.color_palette("Paired"))
```



4.6.5 Dimension reduction on Psi

For downstream analysis, we only using splicing events that have been detected as differential splicing events. As mentioned earlier, the `Psi` is re-quantified by not using the cell-level information.

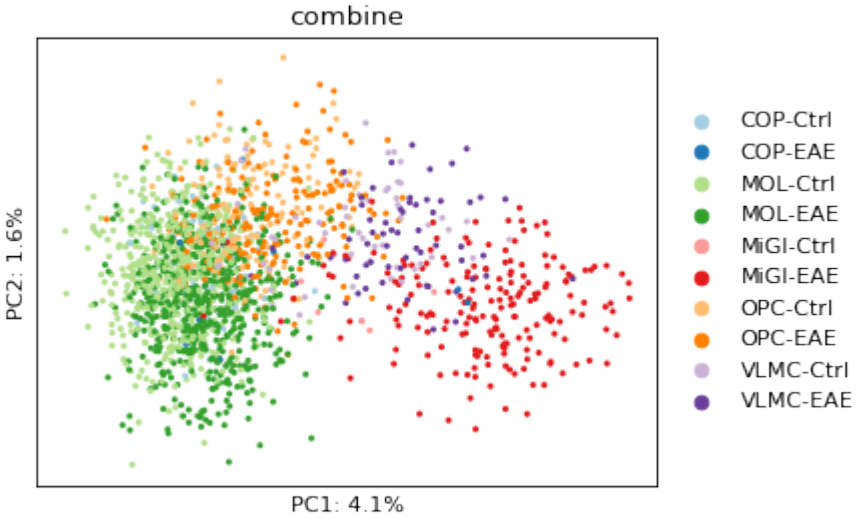
[23]:
```
idx = list(adata.varm['fdr'][:, 0] < 0.05)
adata_psi = adata_aggr[:, idx]

adata_psi.layers['X'] = adata_psi.X.astype(np.float64)
adata_psi.X = adata_psi.layers['Psi']
adata_psi.shape
```

[23]:
```
(1845, 368)
```

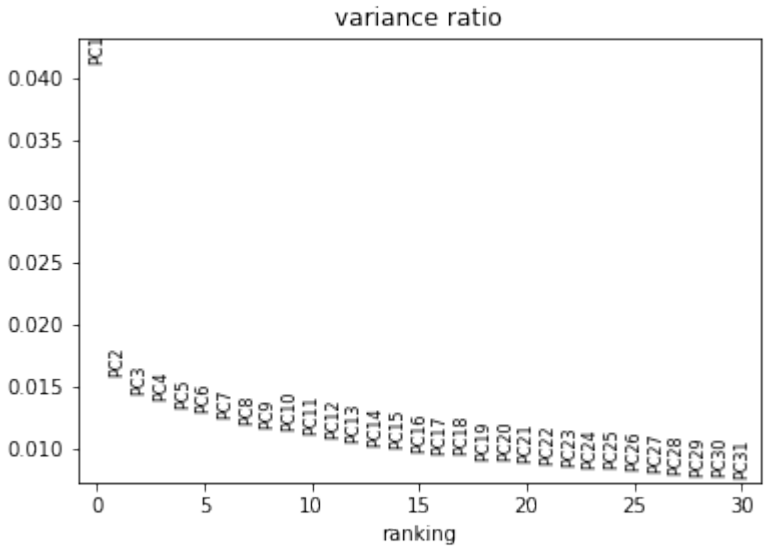
[24]:
```
sc.tl.pca(adata_psi, svd_solver='arpack')
```

[25]:
```
adata_psi.obs['Psi_PC1'] = adata_psi.obsm['X_pca'][:, 0]
adata_psi.obs['Psi_PC2'] = adata_psi.obsm['X_pca'][:, 1]
adata_psi.obs['Psi_PC3'] = adata_psi.obsm['X_pca'][:, 2]
```

[26]:
```
fig = plt.figure(figsize=(5, 3.7), dpi=80)
ax = plt.subplot(1, 1, 1)
sc.pl.pca(adata_psi, color='combine', size=30, show=False, ax=ax,
          palette=sns.color_palette("Paired"))
plt.xlabel("PC1: %.1f%%" %(adata_psi.uns['pca']['variance_ratio'][0] * 100))
plt.ylabel("PC2: %.1f%%" %(adata_psi.uns['pca']['variance_ratio'][1] * 100))
# plt.legend(loc=1)
plt.show()
```

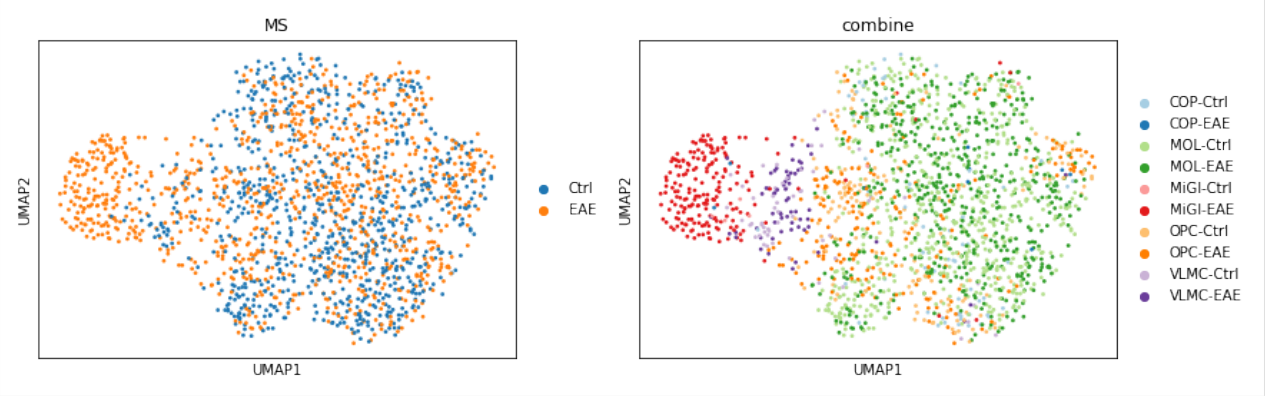


[27]:
```
sc.pl.pca_variance_ratio(adata_psi)
```



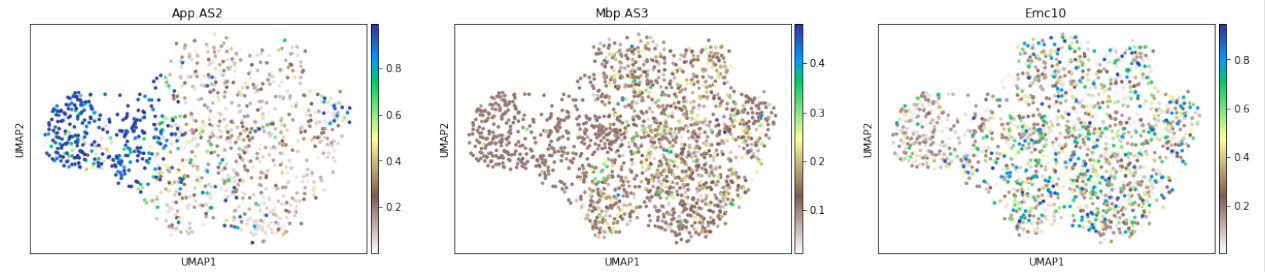
[28]:
```
sc.pp.neighbors(adata_psi, n_neighbors=10, n_pcs=20, method='umap')
sc.tl.umap(adata_psi)
```

[29]:
```
sc.pl.umap(adata_psi, color=['MS', 'combine'], size=30)
```



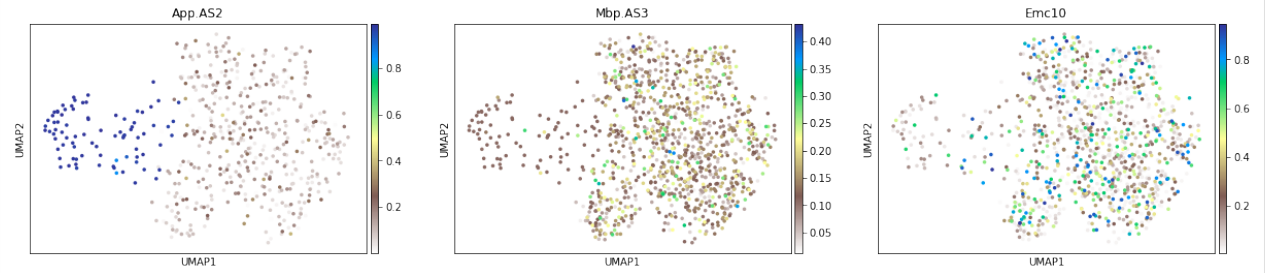
Visualise the Psi value on Psi-based UMAP

[30]:
```
sc.pl.umap(adata_psi, color=['App.AS2', 'Mbp.AS3', 'Emc10'], size=50, cmap='terrain_r
↪')
```

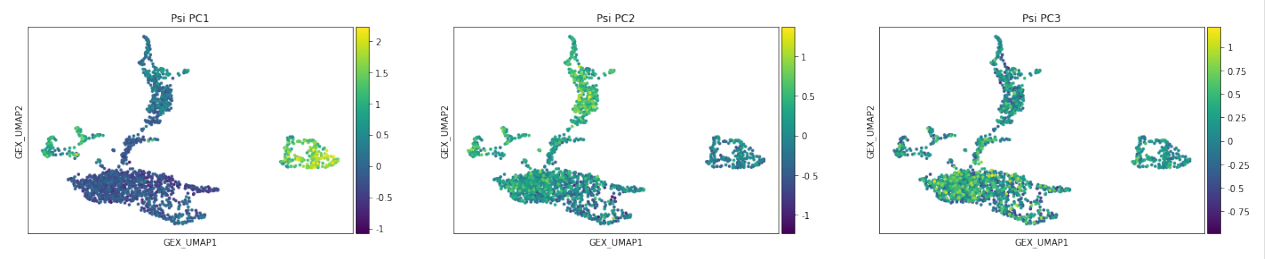


Only for the cells with confident Psi estimate

[31]:
```
adata_tmp = adata_psi[adata_psi[:, 'App.AS2'].layers['Psi_95CI'] < 0.3, :]
adata_tmp.X = adata_tmp.layers['Psi']
sc.pl.umap(adata_tmp, color=['App.AS2', 'Mbp.AS3', 'Emc10'], size=50, cmap='terrain_r
↪')
```



[32]:
```
sc.pl.scatter(adata_psi, basis='GEX_UMAP', color=['Psi_PC1', 'Psi_PC2', 'Psi_PC3'],␣
↪size=50)
```



[ ]:

Prediction of cell type

One potential usage of the quantified `Psi` is to identify cell types. Instead of using them to cluster cells, here we show that by using the `Psi` (and their principle compoments), the annotated cell types from gene expression can be accurately predicted.

[33]:
```
np.random.seed = 1
```

[34]:
```
_val, _idx = np.unique(pd.factorize(adata_psi.obs['cluster'])[0], return_index=True)
print(_val, _idx)
print(adata_psi.obs['cluster'][_idx])
```

```
[0 1 2 3 4] [  0   8  54  92 890]
SRR7102862     OPC
SRR7102870     MOL
SRR7102925     COP
SRR7102967    VLMC
SRR7103970    MiGl
Name: cluster, dtype: category
Categories (5, object): [COP, MOL, MiGl, OPC, VLMC]
```

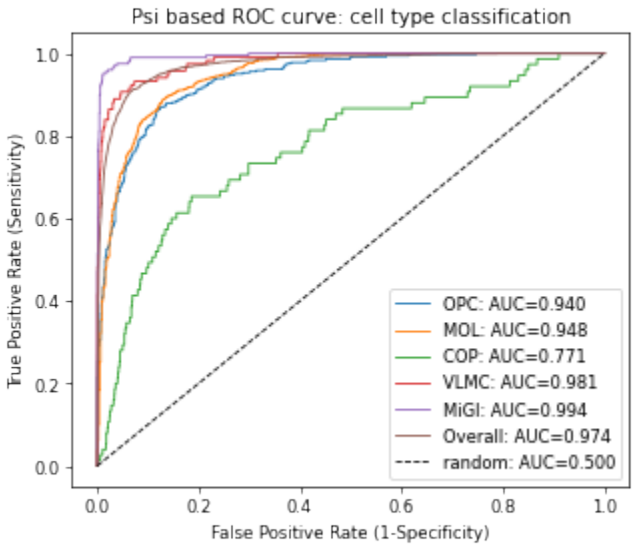
[35]:
```
import scipy.stats as st
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier
from hilearn import ROC_plot, CrossValidation

LogisticRegression = linear_model.LogisticRegression()
RF_class = RandomForestClassifier(n_estimators=100, n_jobs=-1)

X1 = adata_psi.obsm['X_pca'][:, :20]
Y1 = pd.factorize(adata_psi.obs['cluster'])[0]

CV = CrossValidation(X1, Y1)
# Y1_pre, Y1_score = CV.cv_classification(model=RF_class, folds=10)
Y2_pre, Y2_score = CV.cv_classification(model=LogisticRegression, folds=10)
```

[36]:
```
fig = plt.figure(figsize=(6, 5), dpi=60)
# ROC_plot(Y1, Y1_score[:,1], label="Random Forest", base_line=False)
# ROC_plot(Y1, Y2_score[:,1], label="Logistic Regress")

for i in range(5):
    ROC_plot(Y1 == i, Y2_score[:,i],
             label=adata_psi.obs['cluster'][_idx[i]],
```

(continues on next page)

(continued from previous page)

```
            base_line=False)
ROC_plot(np.concatenate([Y1 == i for i in range(5)]),
         Y2_score.T.reshape(-1), label='Overall')

plt.title("Psi based ROC curve: cell type classification")

# fig.savefig(dat_dir + '../../figures/fig_s7_classification.pdf', dpi=300, bbox_
↪inches='tight')
plt.show()
```



Prediction of MS state

Besides using Psi to predict the cell types, a more interesting task is to predict if a cell is in disease state (i.e., EAE state). Though by only using the `Psi`, it achieves moderate performance but clearly improves the prediction together with the gene expression.

[37]:
```
dat_GEX_pc = np.genfromtxt(dat_dir + 'top_20PC.tsv', dtype='str', delimiter='\t')
```

[38]:
```
idx = brie.match(adata_psi.obs.index, dat_GEX_pc[:, 0])
GEX_pc = dat_GEX_pc[idx, 1:].astype(float)
```

[39]:
```
adata_psi
```

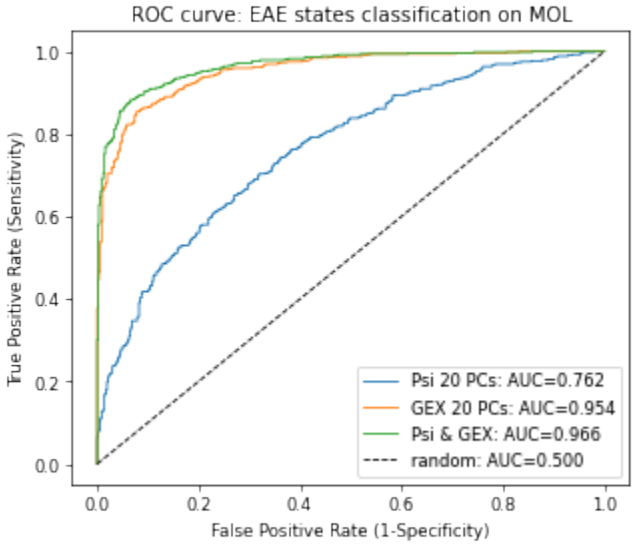
[39]:
```
AnnData object with n_obs × n_vars = 1845 × 368
    obs: 'samID', 'samCOUNT', 'MS', 'isCD1', 'cluster', 'combine', 'Psi_PC1', 'Psi_PC2
↪', 'Psi_PC3'
    var: 'GeneID', 'GeneName', 'TranLens', 'TranIDs', 'n_counts', 'n_counts_uniq'
    uns: 'brie_losses', 'cluster_colors', 'MS_colors', 'pca', 'combine_colors',
↪'neighbors', 'umap'
    obsm: 'Xc', 'X_GEX_UMAP', 'X_pca', 'X_umap'
    varm: 'cell_coeff', 'effLen', 'intercept', 'p_ambiguous', 'sigma', 'PCs'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'ambiguous', 'isoform1', 'isoform2', 'poorQual
↪', 'X'
    obsp: 'distances', 'connectivities'
```

[40]:
```
np.random.RandomState(0)
```

[40]:
```
RandomState(MT19937) at 0x7FDDE46AE270
```

[41]:
```
import scipy.stats as st
from sklearn import linear_model
from hilearn import ROC_plot, CrossValidation

# np.random.seed(0)

X1 = adata_psi.obsm['X_pca'][:, :20]
X2 = GEX_pc
X3 = np.append(GEX_pc, adata_psi.obsm['X_pca'][:, :20], axis=1)
Y1 = pd.factorize(adata_psi.obs['MS'])[0]

X1 = X1[adata_psi.obs['cluster'] == 'MOL']
X2 = X2[adata_psi.obs['cluster'] == 'MOL']
X3 = X3[adata_psi.obs['cluster'] == 'MOL']
Y1 = Y1[adata_psi.obs['cluster'] == 'MOL']

LogisticRegression = linear_model.LogisticRegression(max_iter=5000)
CV = CrossValidation(X1, Y1)
Y1_pre, Y1_score = CV.cv_classification(model=LogisticRegression, folds=10)

LogisticRegression = linear_model.LogisticRegression(max_iter=5000)
CV = CrossValidation(X2, Y1)
Y2_pre, Y2_score = CV.cv_classification(model=LogisticRegression, folds=10)

LogisticRegression = linear_model.LogisticRegression(max_iter=5000)
CV = CrossValidation(X3, Y1)
Y3_pre, Y3_score = CV.cv_classification(model=LogisticRegression, folds=10)
```

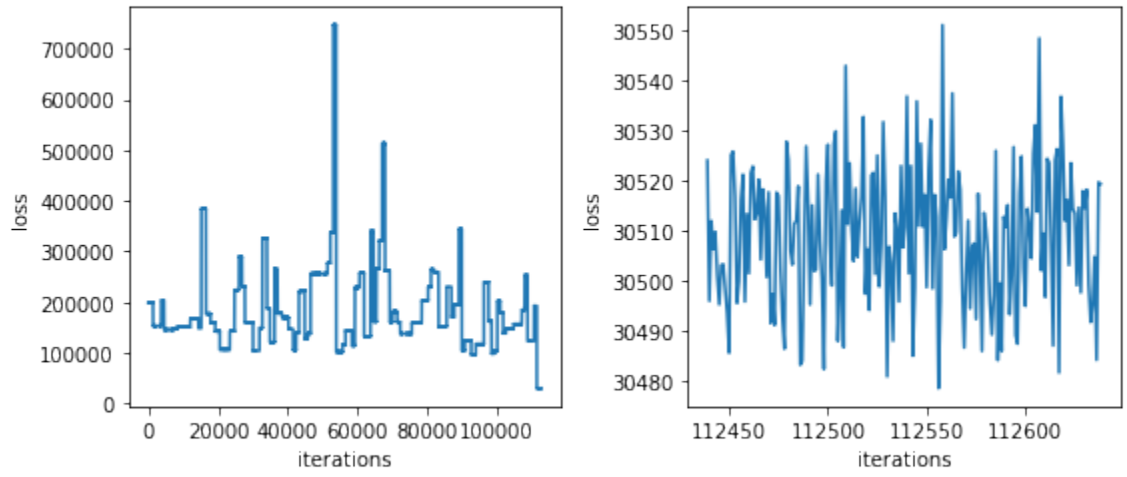
[42]:
```
fig = plt.figure(figsize=(6, 5), dpi=60)
ROC_plot(Y1, Y1_score[:,1], label="Psi 20 PCs", base_line=False)
ROC_plot(Y1, Y2_score[:,1], label="GEX 20 PCs", base_line=False)
ROC_plot(Y1, Y3_score[:,1], label="Psi & GEX")

plt.title("ROC curve: EAE states classification on MOL")

# fig.savefig(dat_dir + '../../figures/fig_s9_EAE_predict.pdf', dpi=300, bbox_inches=
↪'tight')
plt.show()
```



[ ]:

Techinical diagnosis

[43]:
```
## There are multiple batches in running jobs, so you will steps;
## the second part shows the last bit of the final batch

brie.pl.loss(adata.uns['brie_losses'])
```



[ ]:

4.7 scNT-seq

scNT-seq (Qiu, Hu, et al 2020) is a recently proposed technique to metabolically label newly synthesied RNAs in single cells. By using the labelling information, the inferred cellular transitions by Dynamo are hihgly consistent with

the stimulation time (see our reproduced analysis with scripts from the authors).

This relative-longe-period transition is generally difficult to be obtained from RNA velocity by only using total RNAs. Here, we will illustrate that the differential momentum genes could help correct the projected trajectory, thanks to using the stimulation time as a testing (i.e., supervised) covariate.

As BRIE2 takes ~30 minutes with GPU to detect the DMGs, we provide the pre-computed data with these BRIE2 scripts. You can run this notebook by downloading the data, e.g., using the following command line and unzip it into the `./data` folder:

```
wget http://ufpr.dl.sourceforge.net/project/brie-rna/examples/scNTseq/brie2_scNTseq.
↪zip
unzip -j brie2_scNTseq.zip -d ./data
```

4.7.1 Load packages

[1]:
```
import brie
import numpy as np
import scanpy as sc
import scvelo as scv
import matplotlib.pyplot as plt
scv.logging.print_version()
```

```
Running scvelo 0.2.1 (python 3.7.6) on 2020-11-06 11:33.
WARNING: There is a newer scvelo version available on PyPI:
 Your version:           0.2.1
 Latest version:         modeling
```

[2]:
```
scv.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
scv.settings.presenter_view = True  # set max width size for presenter view
scv.set_figure_params('scvelo')  # for beautified visualization
```

[3]:
```
# define the path you store the example data
dat_dir = "./data"
# dat_dir = '/storage/yhhuang/research/brie2/releaseDat/scNTseq/'
```

4.7.2 scVelo dynamical model with total RNAs

This may take a few minutes to run, so we provide the pre-computed the data as in the above zip file for the setting with top 2,000 genes. The commented Python scripts are below. You can get the `neuron_splicing_totalRNA.h5ad` from here generated by the adapted script. If you want to change to more hihgly variable genes, you can change `n_top_genes` to other value, e.g., 8000.

```
adata = scv.read(dat_dir + "/neuron_splicing_totalRNA.h5ad")

scv.pp.filter_and_normalize(adata, min_shared_counts=30, n_top_genes=2000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)

scv.tl.recover_dynamics(adata, var_names='all')
scv.tl.velocity(adata, mode='dynamical')
scv.tl.velocity_graph(adata)
scv.tl.latent_time(adata)
adata.write(dat_dir + "/scvelo_neuron_totalRNA_dynamical_2K.h5ad")
```

Cellular transitions on default selected velocity genes

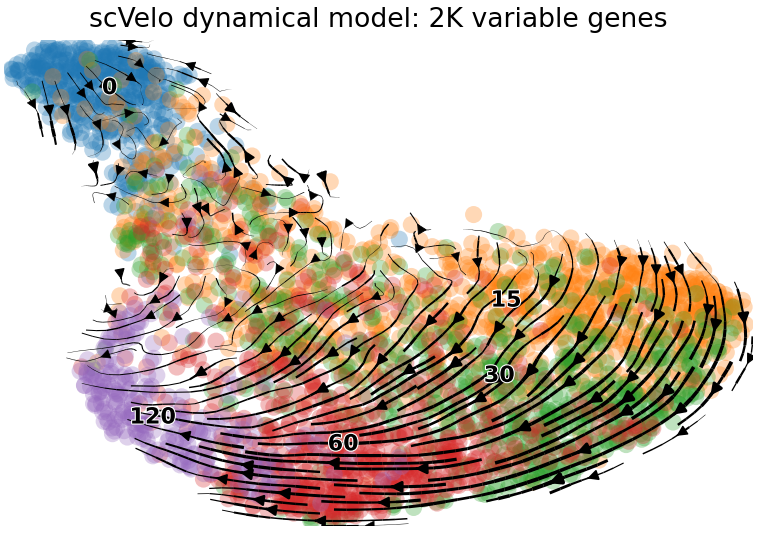
[4]:
```
adata = scv.read(dat_dir + "/scvelo_neuron_totalRNA_dynamical_2K.h5ad")

print(adata.shape, np.sum(adata.var['velocity_genes']))
adata
```

```
(3066, 1999) 132
```

[4]:
```
AnnData object with n_obs × n_vars = 3066 × 1999
    obs: 'cellname', 'time', 'early', 'late', 'initial_size_spliced', 'initial_size_
↪unspliced', 'initial_size', 'n_counts', 'velocity_self_transition', 'root_cells',
↪'end_points', 'velocity_pseudotime', 'latent_time'
    var: 'gene_short_name', 'gene_count_corr', 'means', 'dispersions', 'dispersions_
↪norm', 'fit_alpha', 'fit_beta', 'fit_gamma', 'fit_t_', 'fit_scaling', 'fit_std_u',
↪'fit_std_s', 'fit_likelihood', 'fit_u0', 'fit_s0', 'fit_pval_steady', 'fit_steady_u
↪', 'fit_steady_s', 'fit_variance', 'fit_alignment_scaling', 'fit_r2', 'velocity_
↪genes'
    uns: 'neighbors', 'pca', 'recover_dynamics', 'velocity_graph', 'velocity_graph_neg
↪', 'velocity_params'
    obsm: 'X_pca', 'X_umap'
    varm: 'PCs', 'loss'
    layers: 'Ms', 'Mu', 'fit_t', 'fit_tau', 'fit_tau_', 'spliced', 'unspliced',
↪'velocity', 'velocity_u'
    obsp: 'connectivities', 'distances'
```

[5]:
```
scv.pl.velocity_embedding_stream(adata, basis='umap', color=['time'],
                                 ax=None, show=True, legend_fontsize=10, dpi=80,
                                 title='scVelo dynamical model: 2K variable genes')
```

```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



For 8K top variable genes

You can skip this sub section if not using the 8K top genes

```
adata2 = scv.read(dat_dir + "/scvelo_neuron_totalRNA_dynamical_8K.h5ad")
print(adata2.shape, np.sum(adata2.var['velocity_genes']))

scv.pl.velocity_embedding_stream(adata2, basis='umap', color=['time'],
                                 ax=None, show=True, legend_fontsize=10,
                                 title='scVelo dynamical model: 8K variable genes')
```

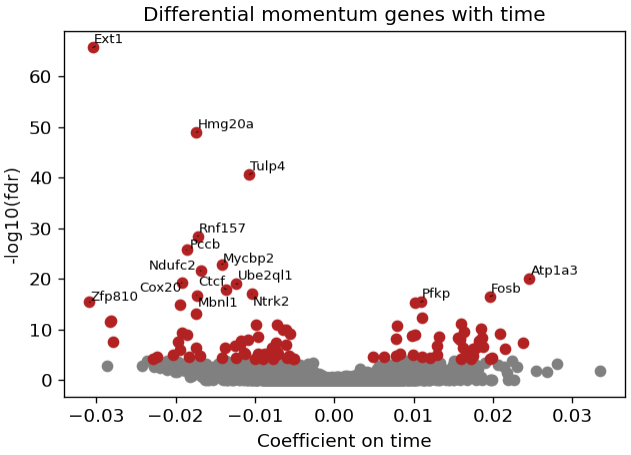
4.7.3 BRIE2 for differential momentum genes (DMGs)

Besides the large h5ad file, BRIE2 also saves the DMGs in a .tsv file brie_neuron_splicing_time.brie_ident.tsv for quick access.

[6]:
```
adata_brie = scv.read(dat_dir + "/brie_neuron_splicing_time.h5ad")

adata_brie
```

[6]:
```
AnnData object with n_obs × n_vars = 3066 × 7849
    obs: 'cellname', 'time', 'early', 'late'
    var: 'gene_short_name', 'n_counts', 'n_counts_uniq', 'loss_gene'
    uns: 'Xc_ids', 'brie_losses', 'brie_param', 'brie_version'
    obsm: 'X_umap', 'Xc'
    varm: 'ELBO_gain', 'cell_coeff', 'fdr', 'intercept', 'pval', 'sigma'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'spliced', 'unspliced'
```

[7]:
```
fig = plt.figure(figsize=(6, 4), dpi=60)
brie.pl.volcano(adata_brie, y='fdr', n_anno=16, adjust=True)
plt.title('Differential momentum genes with time')
plt.xlabel('Coefficient on time')
# plt.savefig(dat_dir + '../../figures/scNT_volcano.png', dpi=300)
plt.show()
```



RNA velocity on DMGs

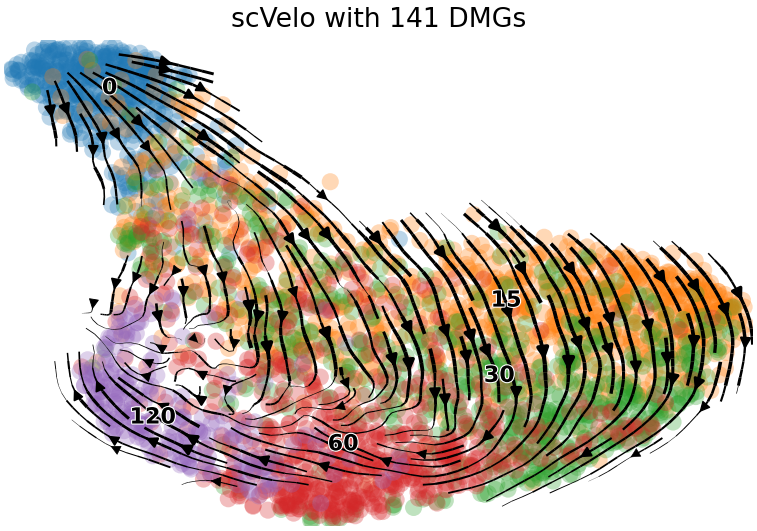
[8]:
```
idx = (np.min(adata_brie.varm['fdr'], axis=1) < 0.01)
gene_use = adata_brie.var.index[idx]
print(sum(idx), sum(brie.match(gene_use, adata.var.index) != None))
n_genes = sum(brie.match(gene_use, adata.var.index) != None)
```

(continues on next page)

(continued from previous page)

```
scv.tl.velocity_graph(adata, gene_subset=gene_use)
scv.pl.velocity_embedding_stream(adata, basis='umap', color=['time'],
                                 legend_fontsize=10,
                                 ax=None, show=True, dpi=80,
                                 title='scVelo with %d DMGs' %(n_genes))
```

```
280 141
computing velocity graph
    finished (0:00:01) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



Change cutoffs

[9]:
```
fig = plt.figure(figsize=(11, 4), dpi=60)
## cutoff 0.001
ax1 = plt.subplot(1, 2, 1)

idx1 = (np.min(adata_brie.varm['fdr'], axis=1) < 0.001)
gene_use1 = adata_brie.var.index[idx1]
n_gene1 = sum(brie.match(gene_use1, adata.var.index) != None)

print(sum(idx1), n_gene1)

scv.tl.velocity_graph(adata, gene_subset=gene_use1)
scv.pl.velocity_embedding_stream(adata, basis='umap', color=['time'],
                                 ax=ax1, show=False, legend_fontsize=10,
                                 title='scVelo with %d DMGs at FDR<0.001' %(n_gene1))
ax1.text(-0.15, 0.95, 'a', transform=ax1.transAxes, size=20, weight='bold')


## cutoff 0.05
ax2 = plt.subplot(1, 2, 2)
```

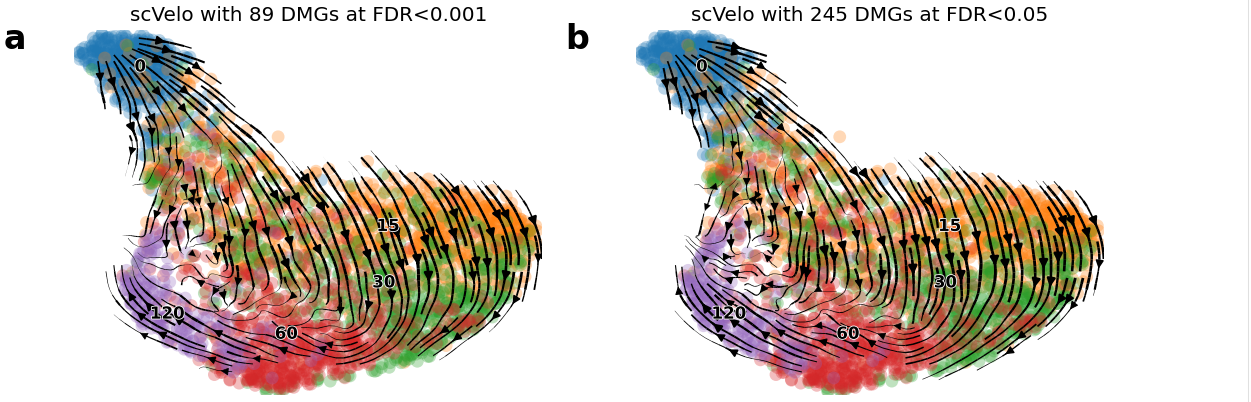
(continues on next page)

(continued from previous page)

```
idx2 = (np.min(adata_brie.varm['fdr'], axis=1) < 0.05)
gene_use2 = adata_brie.var.index[idx2]
n_gene2 = sum(brie.match(gene_use2, adata.var.index) != None)

print(sum(idx2), n_gene2)

scv.tl.velocity_graph(adata, gene_subset=gene_use2)
scv.pl.velocity_embedding_stream(adata, basis='umap', color=['time'],
                                 ax=ax2, show=False, legend_fontsize=10,
                                 title='scVelo with %d DMGs at FDR<0.05' %(n_gene2))
ax2.text(-0.15, 0.95, 'b', transform=ax2.transAxes, size=20, weight='bold')

# plt.tight_layout()
# plt.savefig(dat_dir + '../../figures/scNT_scVelo_brie_FDR.png', dpi=300)
# plt.show()
```

```
165 89
computing velocity graph
    finished (0:00:01) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
516 245
computing velocity graph
    finished (0:00:02) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```

[9]:
```
Text(-0.15, 0.95, 'b')
```



Visualize gene count

[10]:
```
idx = (np.min(adata_brie.varm['fdr'], axis=1) < 0.1)
gene_use = adata_brie.var.index[idx]

mm = brie.match(gene_use, adata.var.index) != None
gene_use[mm]
```

[10]:
```
Index(['Ncor1', 'App', 'Meis2', 'Tcf12', 'Elavl4', 'Cdk2ap1', 'Usp24', 'Ahi1',
       'Zfp608', 'Meaf6',
       ...
       'Zbtb11', 'Pdpk1', 'Pccb', 'Frem2', 'Sap18', 'Celf4', 'St6gal1',
       'Zfp704', 'Zkscan1', 'Mpp6'],
      dtype='object', length=319)
```

[11]:
```
## sorted by fdr
idx_sort = np.argsort(adata_brie[:, gene_use[mm]].varm['fdr'][:,0])
gene_use[mm][idx_sort]
```

[11]:
```
Index(['Ext1', 'Pccb', 'Ube2ql1', 'Ntrk2', 'Mbnl1', 'Fosb', 'Pfkp', 'Orc5',
       'Srd5a1', 'Chka',
       ...
       'Tjp1', 'Hivep3', 'Zkscan1', 'Mthfd1l', 'Prrc2c', 'Mllt3', 'Pdpk1',
       'St6gal1', 'Ktn1', 'Prkcb'],
      dtype='object', length=319)
```

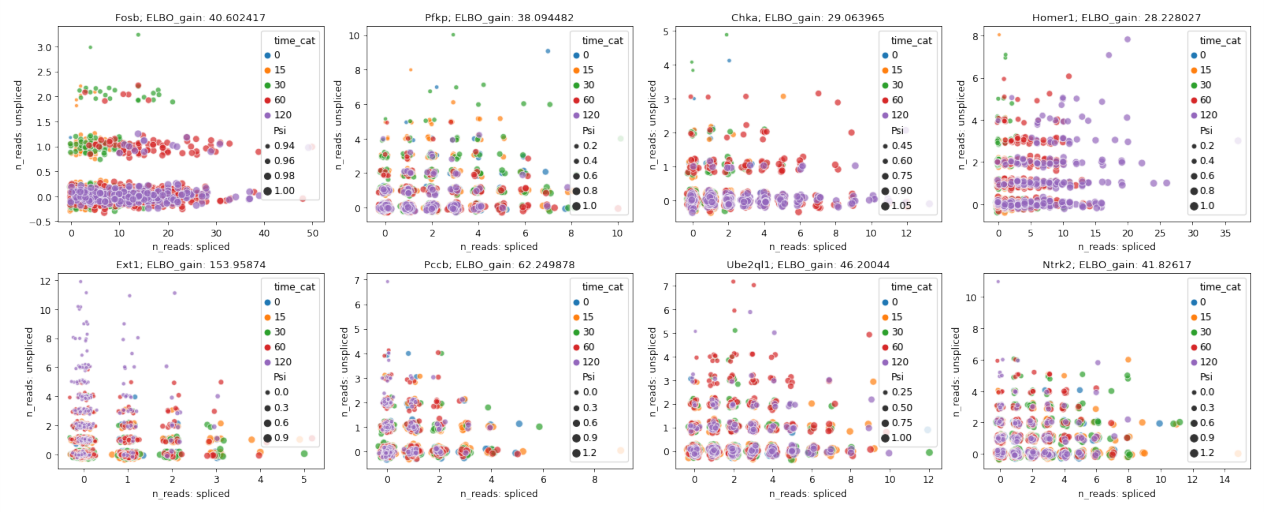
[12]:
```
## Only negative coefficient
gene_sorted = gene_use[mm][idx_sort]
gene_sorted[adata_brie[:, gene_sorted].varm['cell_coeff'][:, 0] < 0]
```

[12]:
```
Index(['Ext1', 'Pccb', 'Ube2ql1', 'Ntrk2', 'Mbnl1', 'Orc5', 'Srd5a1', 'Akap9',
       'Ank2', 'Dlgap1',
       ...
       'Gmds', 'Satb2', 'Tjp1', 'Hivep3', 'Zkscan1', 'Mthfd1l', 'Prrc2c',
       'Mllt3', 'St6gal1', 'Ktn1'],
      dtype='object', length=212)
```

[13]:
```
## Only positive coefficient
gene_sorted = gene_use[mm][idx_sort]
gene_sorted[adata_brie[:, gene_sorted].varm['cell_coeff'][:, 0] > 0]
```

[13]:
```
Index(['Fosb', 'Pfkp', 'Chka', 'Homer1', 'Erf', 'Nup98', 'Ncapg2', 'Ifrd1',
       'Rasgef1b', 'Tiparp',
       ...
       '1810030O07Rik', 'Jmy', 'Slc3a2', 'Kcnq5', 'Stx5a', 'Clk4', 'Fbl',
       'Sap18', 'Pdpk1', 'Prkcb'],
      dtype='object', length=107)
```

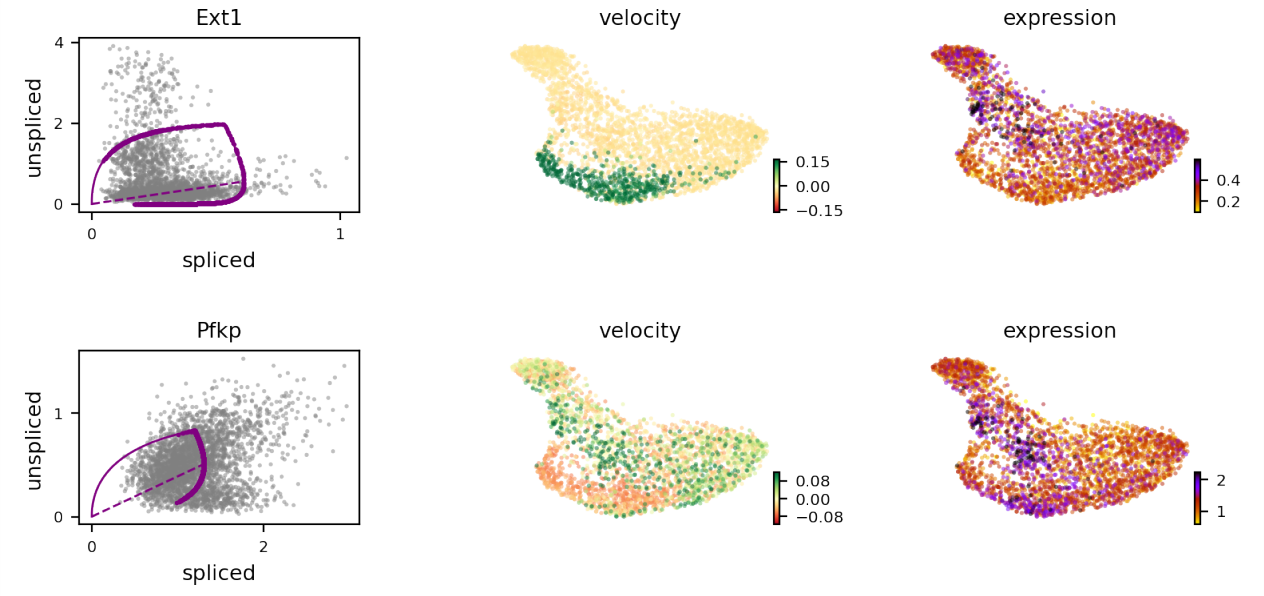
[17]:
```
adata_brie.obs['time_cat'] = adata_brie.obs['time'].astype('category')

gene_top_neg = ['Ext1', 'Pccb', 'Ube2ql1', 'Ntrk2']
gene_top_pos = ['Fosb', 'Pfkp', 'Chka', 'Homer1']

fig = plt.figure(figsize=(20, 8), dpi=40)
brie.pl.counts(adata_brie, genes=gene_top_pos + gene_top_neg,
               layers=['spliced', 'unspliced'],
               color='time_cat', add_val='ELBO_gain',
               nrow=2, alpha=0.7, legend='brief', noise_scale=0.1)

# plt.savefig(dat_dir + '../../figures/scNT_DMG_counts.png', dpi=150)
# plt.show()
```



[15]:
```
adata_brie[:, gene_top_pos + gene_top_neg].varm['cell_coeff']
```

[15]:
```
ArrayView([[ 0.0196298 ],
           [ 0.01087456],
           [ 0.01594826],
           [ 0.00791929],
           [-0.03045544],
           [-0.0186206 ],
           [-0.01236648],
           [-0.01042797]], dtype=float32)
```

[16]:
```
scv.pl.velocity(adata, var_names=['Ext1', 'Pfkp'], colorbar=True, ncols=1)
```



[ ]:

4.8 Dentate Gyrus

This processed Dentate Gyrus data set is downloaded from scVelo package, which is a very nice tool for RNA velocity quantification. Here, we will illustrate that the differential momentum genes could reveal biological informed trajectory, thanks to its supervised manner on informative gene detection.

You can run this notebook with our pre-computed data with these BRIE2 command lines, by downloading it e.g., using the following command line and unzip it into the `./data` folder:

```
wget http://ufpr.dl.sourceforge.net/project/brie-rna/examples/dentateGyrus/brie2_
↪dentateGyrus.zip
unzip -j brie2_dentateGyrus.zip -d ./data
```

4.8.1 Load packages

[1]:
```
import brie
import numpy as np
import scanpy as sc
import scvelo as scv
import matplotlib.pyplot as plt

scv.logging.print_version()
print('brie version: %s' %brie.__version__)
```

```
Running scvelo 0.2.1 (python 3.7.6) on 2020-11-06 11:53.
WARNING: There is a newer scvelo version available on PyPI:
 Your version:           0.2.1
 Latest version:         modeling
brie version: 2.0.5
```

[2]:
```
scv.settings.verbosity = 3  # show errors(0), warnings(1), info(2), hints(3)
scv.settings.presenter_view = True  # set max width size for presenter view
scv.set_figure_params('scvelo')  # for beautified visualization
```

[3]:
```
# define the path you store the example data
dat_dir = "./data"
# dat_dir = "/home/yuanhua/research/brie2/releaseDat/dentateGyrus/"
```

4.8.2 scVelo's default results

Fitting scVelo

Scvelo's stochastic and dynamical models may take a few minutes to run, so we pre-run it by the following codes, and the fitted data can be found in the downloaded zip file.

Stochastic model

```
adata = scv.datasets.dentategyrus()

scv.pp.filter_and_normalize(adata, min_shared_counts=30, n_top_genes=3000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)

scv.tl.velocity(adata)
```

(continues on next page)

(continued from previous page)

```
scv.tl.velocity_graph(adata)
adata.write(dat_dir + "/dentategyrus_scvelo_stoc_3K.h5ad")
```

[4]:
```
adata = scv.read(dat_dir + "/dentategyrus_scvelo_stoc_3K.h5ad")
```

[5]:
```
scv.utils.show_proportions(adata)
adata
```

```
Abundance of ['spliced', 'unspliced']: [0.9 0.1]
```

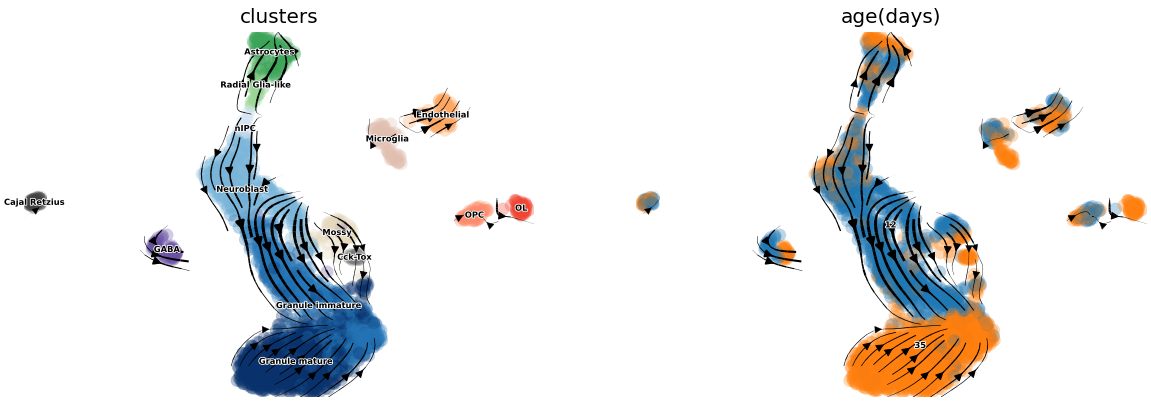
[5]:
```
AnnData object with n_obs × n_vars = 2930 × 2894
    obs: 'clusters', 'age(days)', 'clusters_enlarged', 'initial_size_unspliced',
↪'initial_size_spliced', 'initial_size', 'n_counts', 'velocity_self_transition'
    var: 'velocity_gamma', 'velocity_r2', 'velocity_genes'
    uns: 'clusters_colors', 'neighbors', 'pca', 'velocity_graph', 'velocity_graph_neg
↪', 'velocity_params'
    obsm: 'X_pca', 'X_umap'
    varm: 'PCs'
    layers: 'Ms', 'Mu', 'ambiguous', 'spliced', 'unspliced', 'variance_velocity',
↪'velocity'
    obsp: 'connectivities', 'distances'
```

[6]:
```
print(np.sum(adata.var['velocity_genes']))
```

```
634
```

[7]:
```
scv.tl.velocity_graph(adata, gene_subset=None)
scv.pl.velocity_embedding_stream(adata, basis='umap',
                                 color=['clusters', 'age(days)'],
                                 legend_fontsize=5, dpi=60)
```

```
computing velocity graph
    finished (0:00:04) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



[8]:
```
## Plotting with higher figure resolution

# scv.pl.velocity_embedding_stream(adata, basis='umap', color=['clusters'],
#                                  legend_fontsize=5, dpi=150, title='')
```

(continues on next page)

(continued from previous page)

```
# scv.pl.velocity_embedding(adata, basis='umap', arrow_length=2, arrow_size=2,␣
↪dpi=300, title='')
```

[ ]:

4.8.3 BRIE2's differential momentum genes (DMGs)

Here, we aim to use BRIE2 to detect the differential momentum genes between cell types and illustrate its performance on projecting RNA velocity to cellular transitions. The command line and design matrix are: run_brie2.sh and dentategyrus_cluster_OL.tsv.

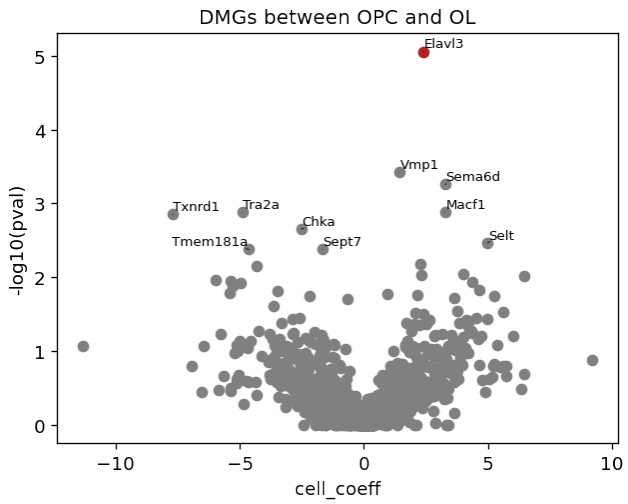
One main differentiation is between OPC and OL. So, we took out these 103 cells and run BRIE2 to detect the DMGs by testing the covariate `is_OL`.

[9]:
```
adata_brie_OL = scv.read(dat_dir + "/brie_dentategyrus_cluster_subOL.h5ad")
adata_OL = adata[adata_brie_OL.obs.index, :]
```

Vocalno plot for DMGs between OPC and OL. cell_ceoff means effect size on logit(Psi), Psi means proportion of spliced RNAs. Positive effect size means OL has higher proportion of spliced RNAs.

[10]:
```
plt.figure(figsize=(6, 4.5), dpi=60)
brie.pl.volcano(adata_brie_OL)
plt.title('DMGs between OPC and OL')
# plt.savefig(dat_dir + '../../brie2/figures/fig_s11_vocalno_DMG_OPCvsOL.pdf',␣
↪dpi=150)
```

[10]:
```
Text(0.5, 1.0, 'DMGs between OPC and OL')
```

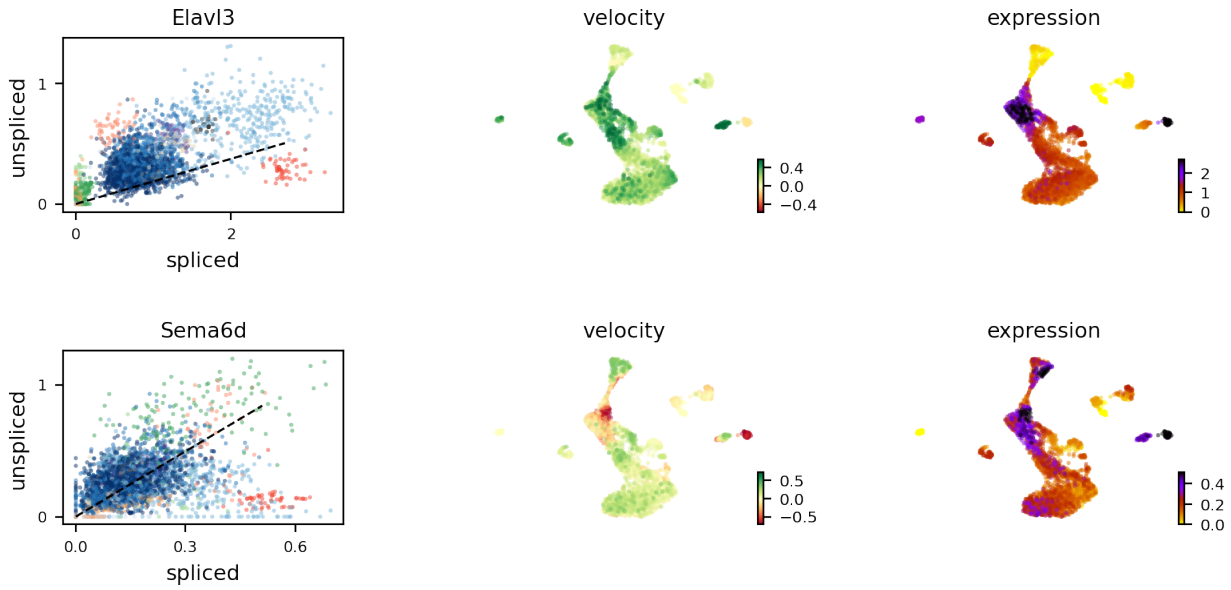


[11]:
```
top_genes_OL = adata_brie_OL.var.index[np.argsort(list(np.min(adata_brie_OL.varm['fdr
↪'], axis=1)))[:100]]
top_genes_OL[:4]
```

[11]:
```
Index(['Elavl3', 'Sema6d', 'Vmp1', 'Tra2a'], dtype='object', name='index')
```

[12]:
```
print(adata.var['velocity_r2'][top_genes_OL[:4]])

scv.pl.velocity(adata, var_names=top_genes_OL[:2], colorbar=True, ncols=1)
```

```
index
Elavl3    0.310708
Sema6d   -0.358877
Vmp1     -0.736003
Tra2a    -1.474734
Name: velocity_r2, dtype: float64
```

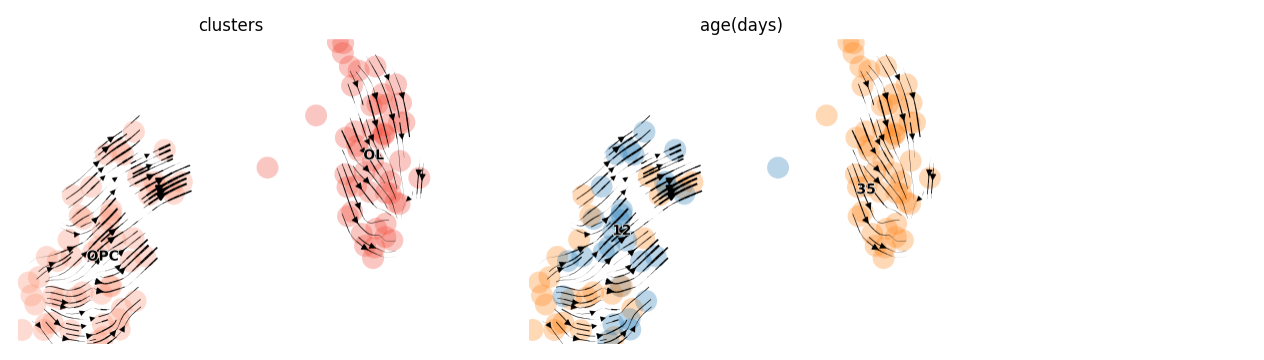


[13]:
```
idx = adata_brie_OL.varm['ELBO_gain'][:, 0] > 2
gene_use = adata_brie_OL.var.index[idx]

print(len(gene_use), sum(brie.match(gene_use, adata_OL.var.index) != None))

scv.tl.velocity_graph(adata_OL, gene_subset=gene_use)
scv.pl.velocity_embedding_stream(adata_OL, basis='umap',
                                 color=['clusters', 'age(days)'],
                                 legend_fontsize=10, dpi=50, size=1000)
```

```
48 48
computing velocity graph
... 100%
```

```
Trying to set attribute `.uns` of view, copying.
```

```
    finished (0:00:00) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



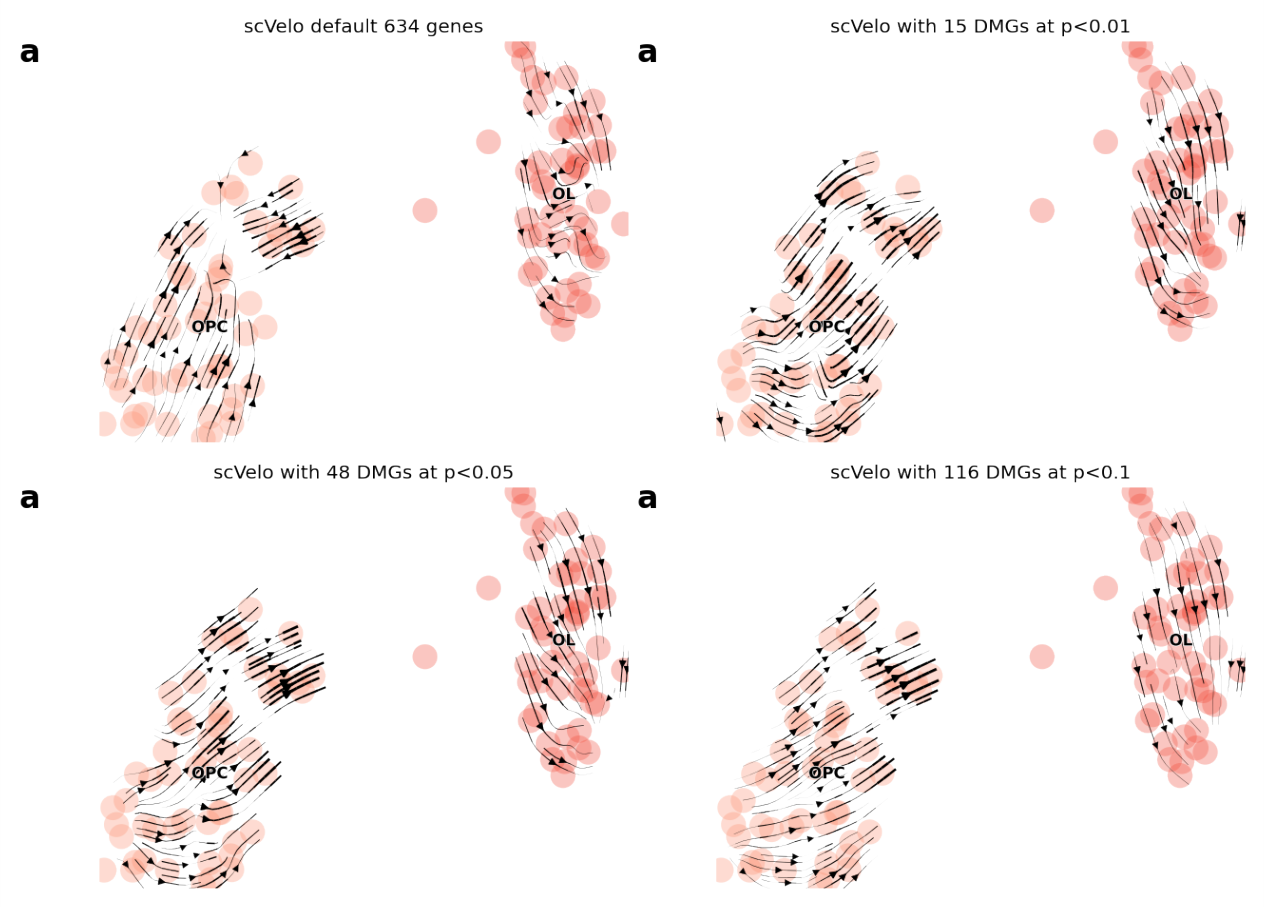
[14]:
```
fig = plt.figure(figsize=(11, 8), dpi=60)
## cutoff 0.001
ax1 = plt.subplot(2, 2, 1)

print(np.sum(adata_OL.var['velocity_genes']))
idx1 = adata_OL.var['velocity_genes']
scv.tl.velocity_graph(adata_OL, gene_subset=None)
scv.pl.velocity_embedding_stream(adata_OL, basis='umap', color=['clusters'],
                                 ax=ax1, show=False, legend_fontsize=10,
                                 size=1000, title='scVelo default %d genes'
↪%(sum(idx1)))
ax1.text(-0.15, 0.95, 'a', transform=ax1.transAxes, size=20, weight='bold')

## cutoff 0.05
ax1 = plt.subplot(2, 2, 2)
idx1 = adata_brie_OL.varm['pval'][:, 0] < 0.01
gene_use1 = adata_brie_OL.var.index[idx1]
print(sum(idx1), sum(brie.match(gene_use1, adata_OL.var.index) != None))

scv.tl.velocity_graph(adata_OL, gene_subset=gene_use1)
scv.pl.velocity_embedding_stream(adata_OL, basis='umap', color=['clusters'],
                                 ax=ax1, show=False, legend_fontsize=10,
                                 size=1000, title='scVelo with %d DMGs at p<0.01'
↪%(sum(idx1)))
ax1.text(-0.15, 0.95, 'a', transform=ax1.transAxes, size=20, weight='bold')

## cutoff 0.05
ax1 = plt.subplot(2, 2, 3)

idx1 = adata_brie_OL.varm['pval'][:, 0] < 0.05
gene_use1 = adata_brie_OL.var.index[idx1]
print(sum(idx1), sum(brie.match(gene_use1, adata_OL.var.index) != None))

scv.tl.velocity_graph(adata_OL, gene_subset=gene_use1)
scv.pl.velocity_embedding_stream(adata_OL, basis='umap', color=['clusters'],
                                 ax=ax1, show=False, legend_fontsize=10,
                                 size=1000, title='scVelo with %d DMGs at p<0.05'
↪%(sum(idx1)))
ax1.text(-0.15, 0.95, 'a', transform=ax1.transAxes, size=20, weight='bold')

## cutoff 0.05
ax1 = plt.subplot(2, 2, 4)
idx1 = adata_brie_OL.varm['pval'][:, 0] < 0.1
gene_use1 = adata_brie_OL.var.index[idx1]
```

(continues on next page)

(continued from previous page)

```
print(sum(idx1), sum(brie.match(gene_use1, adata_OL.var.index) != None))

scv.tl.velocity_graph(adata_OL, gene_subset=gene_use1)
scv.pl.velocity_embedding_stream(adata_OL, basis='umap', color=['clusters'],
                                 ax=ax1, show=False, legend_fontsize=10,
                                 size=1000, title='scVelo with %d DMGs at p<0.1'
↪%(sum(idx1)))
ax1.text(-0.15, 0.95, 'a', transform=ax1.transAxes, size=20, weight='bold')


plt.tight_layout()
# plt.savefig(dat_dir + '../../brie2/figures/fig_s12_OPLvsOL_direction.png', dpi=300)
plt.show()
```

```
634
computing velocity graph
    finished (0:00:00) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
15 15
computing velocity graph
    finished (0:00:00) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
48 48
computing velocity graph
    finished (0:00:00) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
116 116
computing velocity graph
    finished (0:00:00) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



[ ]:

We then detect the DMGs for each cell type vs the rest and use them to explore their impact on cellular transitions.

[15]:
```
adata_brie = scv.read(dat_dir + "/brie_dentategyrus_cluster.h5ad")
cdr = np.array((adata_brie.X > 0).mean(0))[0, :]

adata_brie
```

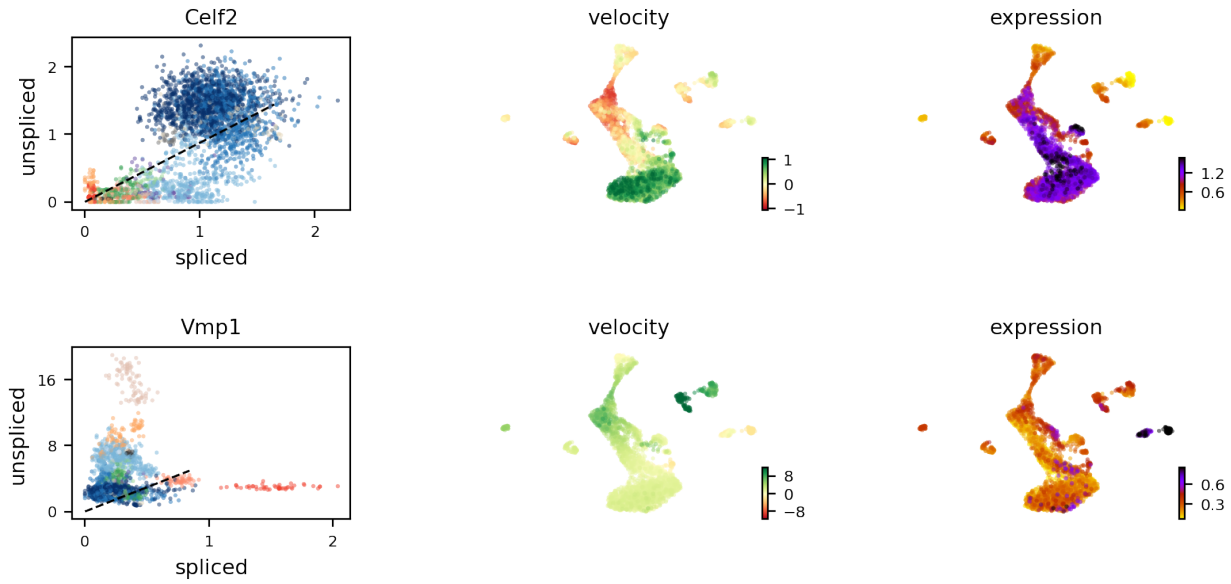
[15]:
```
AnnData object with n_obs × n_vars = 2930 × 2879
    obs: 'clusters', 'age(days)', 'clusters_enlarged', 'initial_size_spliced',
↪'initial_size_unspliced', 'initial_size'
    var: 'n_counts', 'n_counts_uniq', 'loss_gene'
    uns: 'Xc_ids', 'brie_losses', 'brie_param', 'brie_version', 'clusters_colors'
    obsm: 'X_umap', 'Xc'
    varm: 'ELBO_gain', 'cell_coeff', 'fdr', 'intercept', 'pval', 'sigma'
    layers: 'Psi', 'Psi_95CI', 'Z_std', 'ambiguous', 'spliced', 'unspliced'
```

[16]:
```
top_genes_brie = adata_brie.var.index[np.argsort(list(np.min(adata_brie.varm['fdr'],␣
↪axis=1)))[:100]]
top_genes_brie[:4]
```

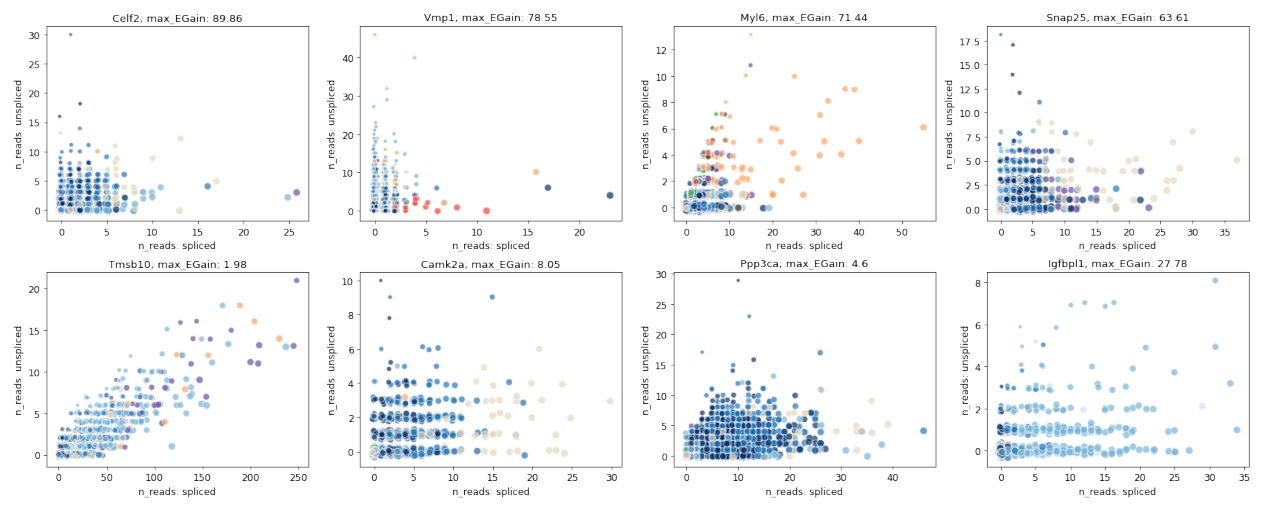
[16]:
```
Index(['Celf2', 'Vmp1', 'Myl6', 'Snap25'], dtype='object', name='index')
```

[17]:
```
print(adata.var['velocity_r2'][top_genes_brie[:4]])
scv.pl.velocity(adata, var_names=top_genes_brie[:2], colorbar=True, ncols=1)
```

```
index
Celf2      0.318048
Vmp1      -0.736003
Myl6       0.338808
Snap25    -0.970810
Name: velocity_r2, dtype: float64
```

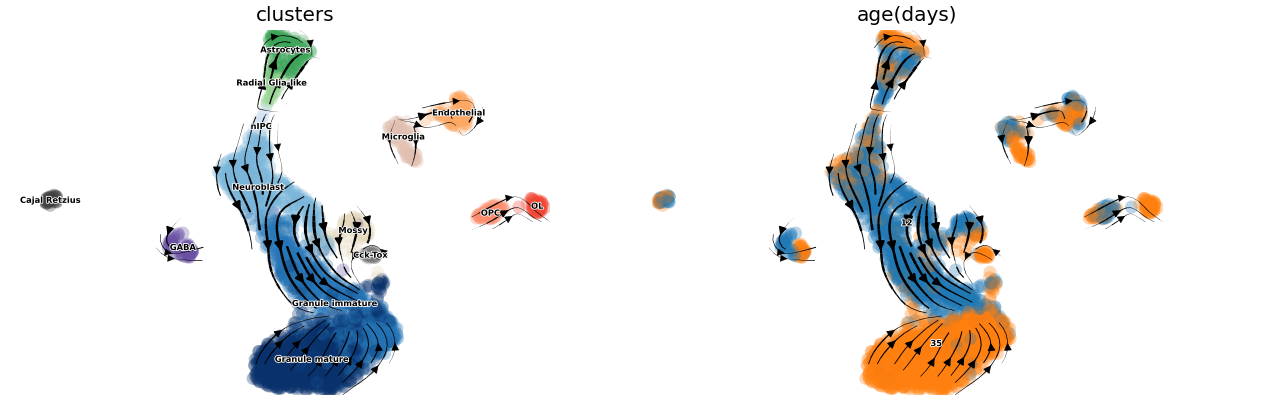


[18]:
```
import seaborn as sns

scVelo_genes = ['Tmsb10', 'Camk2a', 'Ppp3ca', 'Igfbpl1']
adata_brie.varm['max_EGain'] = np.round(np.max(adata_brie.varm['ELBO_gain'], axis=1,␣
↪keepdims=True), 2)

fig = plt.figure(figsize=(20, 8), dpi=40)
brie.pl.counts(adata_brie, genes=np.append(top_genes_brie[:4], scVelo_genes),
               layers=['spliced', 'unspliced'],
               color='clusters', add_val='max_EGain',
               nrow=2, alpha=0.7, legend=False,
               palette=sns.color_palette(adata_brie.uns['clusters_colors']),
               hue_order=np.unique(adata_brie.obs['clusters']), noise_scale=0.1)

# plt.savefig(dat_dir + '../../brie2/figures/fig_s10_DMG_counts.png', dpi=200)
plt.show()
```



[19]:
```
idx = (np.min(adata_brie.varm['fdr'], axis=1) < 0.05) * (cdr > 0.15)
gene_use = adata_brie.var.index[idx]

print(len(gene_use), sum(brie.match(gene_use, adata.var.index) != None))

scv.tl.velocity_graph(adata, gene_subset=gene_use)
scv.pl.velocity_embedding_stream(adata, basis='umap',
                                 color=['clusters', 'age(days)'],
                                 legend_fontsize=5, dpi=60)
```

```
335 335
computing velocity graph
    finished (0:00:02) --> added
    'velocity_graph', sparse matrix with cosine correlations (adata.uns)
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



[20]:
```
## Plotting with higher figure resolution

# scv.pl.velocity_embedding_stream(adata, basis='umap', color=['clusters'],
#                                  legend_fontsize=5, dpi=150, title='')

# scv.pl.velocity_embedding(adata, basis='umap', arrow_length=2, arrow_size=2,␣
↪dpi=300, title='')
```

[ ]:

4.8.4 scVelo's dynamical model

Dynamical model

```
adata = scv.datasets.dentategyrus()

scv.pp.filter_and_normalize(adata, min_shared_counts=30, n_top_genes=3000)
scv.pp.moments(adata, n_pcs=30, n_neighbors=30)

scv.tl.recover_dynamics(adata, var_names='all')
scv.tl.velocity(adata, mode='dynamical')
scv.tl.velocity_graph(adata)
scv.tl.latent_time(adata)
adata.write(dat_dir + "/dentategyrus_scvelo_dyna_3K.h5ad")
```

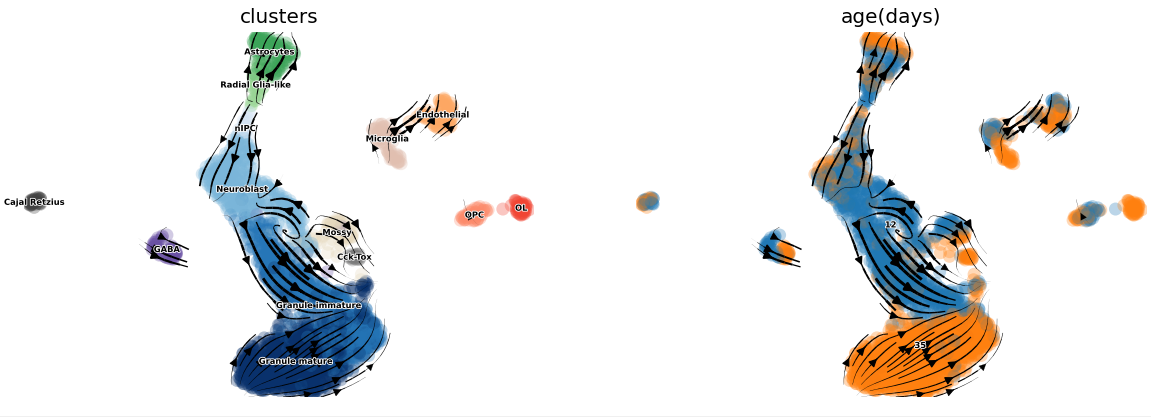
[21]:
```
adata_dyna = scv.read(dat_dir + "/dentategyrus_scvelo_dyna_3K.h5ad")
```

[22]:
```
print(np.sum(adata_dyna.var['velocity_genes']))
```

```
1066
```

[23]:
```
# scv.tl.velocity_graph(adata_dyna, gene_subset=None)
scv.pl.velocity_embedding_stream(adata_dyna, basis='umap',
                                 color=['clusters', 'age(days)'],
                                 legend_fontsize=5, dpi=60)
```

```
computing velocity embedding
    finished (0:00:00) --> added
    'velocity_umap', embedded velocity vectors (adata.obsm)
```



[ ]:

4.9 Prior distribtuion

[3]:
```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow_probability import distributions as tfd
```

[21]:
```
import matplotlib.pyplot as plt
```

4.9.1 Logit-Normal

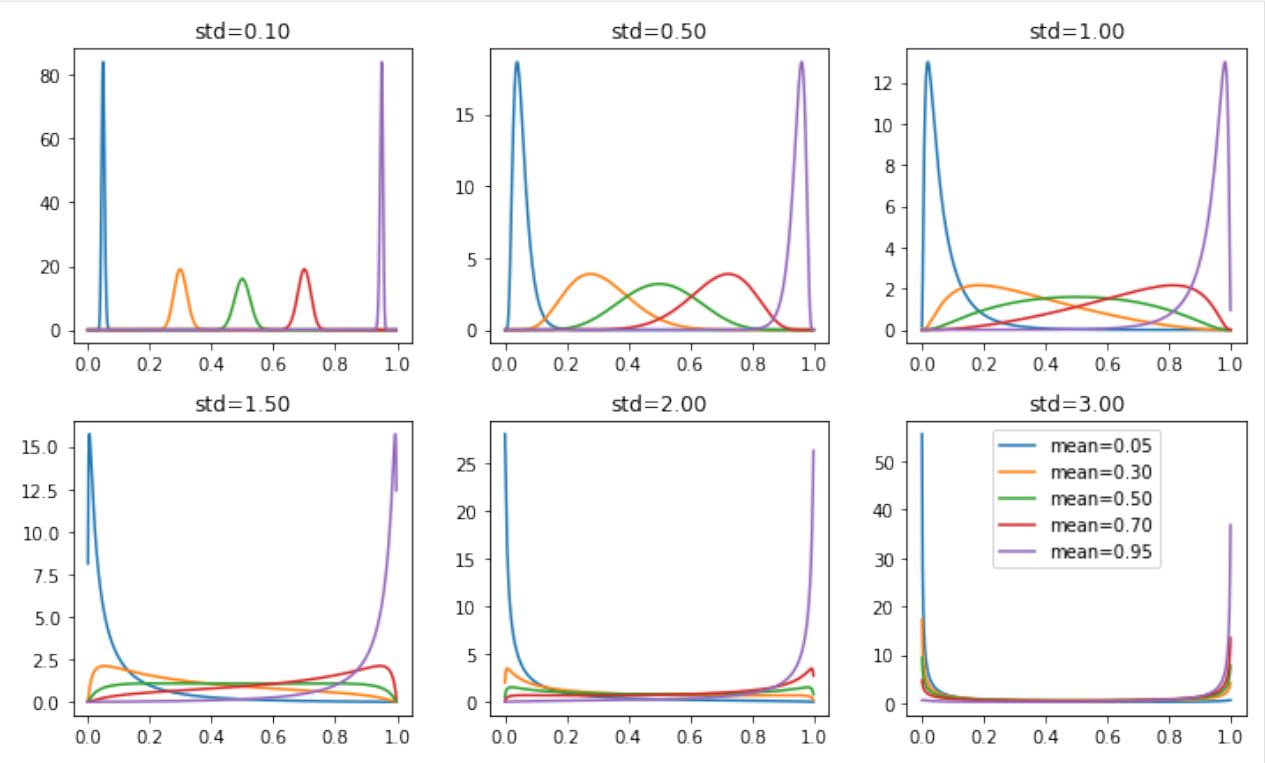
See more on logit-normal distribution Wikipedia page.

In practicce, we use $\mu = 0, \sigma = 3.0$ as prior

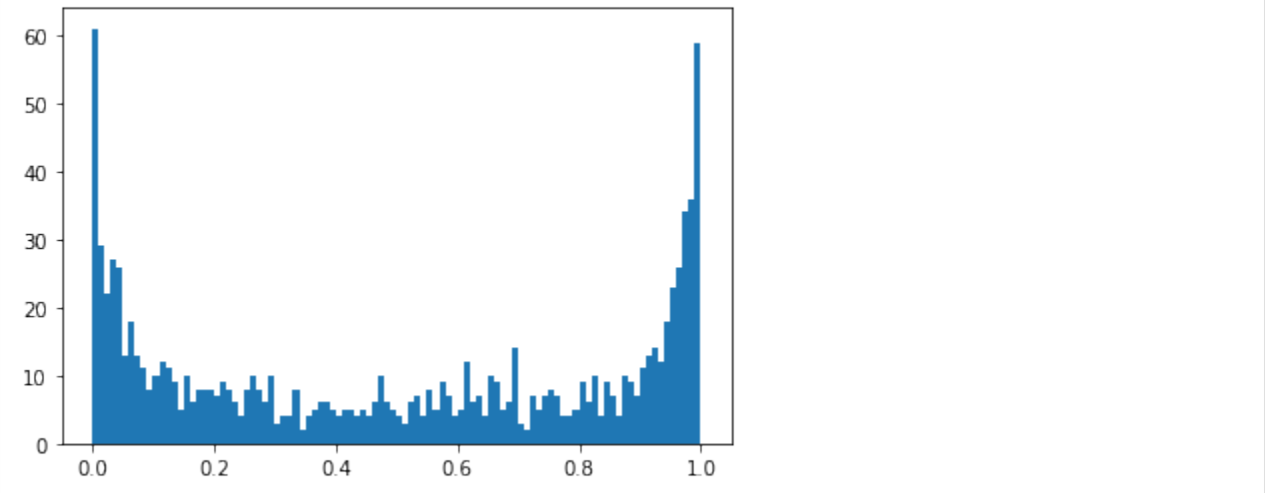
[30]:
```
def _logit(x):
    return tf.math.log(x / (1 - x))

_logit(np.array([0.6]))
```

[30]:
```
<tf.Tensor: id=7912, shape=(1,), dtype=float64, numpy=array([0.40546511])>
```

[40]:
```
_means = np.array([0.05, 0.3, 0.5, 0.7, 0.95], dtype=np.float32)
_vars = np.array([0.1, 0.5, 1.0, 1.5, 2.0, 3.0], dtype=np.float32)

xx = np.arange(0.001, 0.999, 0.001).astype(np.float32)

fig = plt.figure(figsize=(10, 6))
for j in range(len(_vars)):
    plt.subplot(2, 3, j + 1)
    for i in range(len(_means)):
        _model = tfd.Normal(_logit(_means[i:i+1]), _vars[j:j+1])
        _pdf = _model.prob(_logit(xx)) * 1 / (xx * (1 - xx))
        plt.plot(xx, _pdf, label="mean=%.2f" %(_means[i]))
    plt.title("std=%.2f" %(_vars[j]))
    if j == 5:
        plt.legend(loc="best")

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



[73]:
```
_model = tfd.Normal([0], [3])
plt.hist(np.array(tf.sigmoid(_model.sample(1000))).reshape(-1), bins=100)
plt.show()
```



4.9.2 Gamma distribution

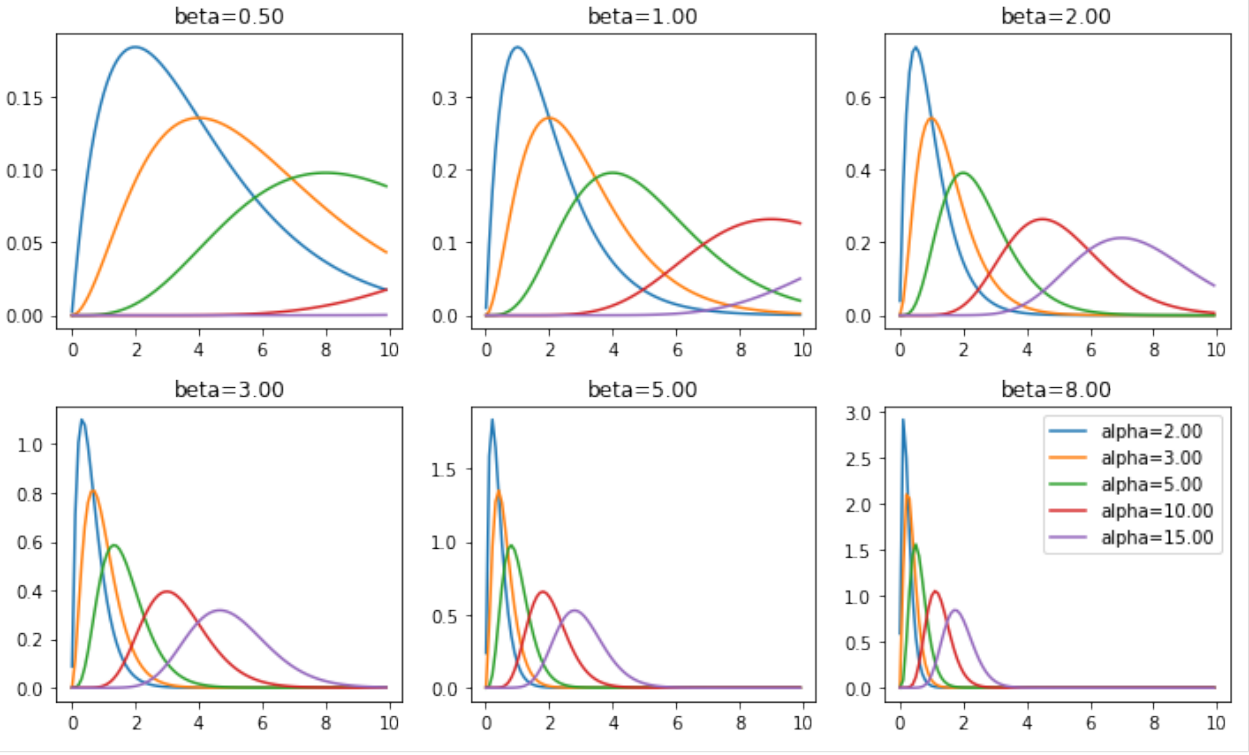
See more on Gamma distribution Wikipedia page In practice, we use $\alpha = 10, \beta = 3$ as prior

[65]:
```
_alpha = np.array([2, 3, 5, 10, 15], dtype=np.float32)
_beta = np.array([0.5, 1, 2, 3, 5, 8], dtype=np.float32)
```

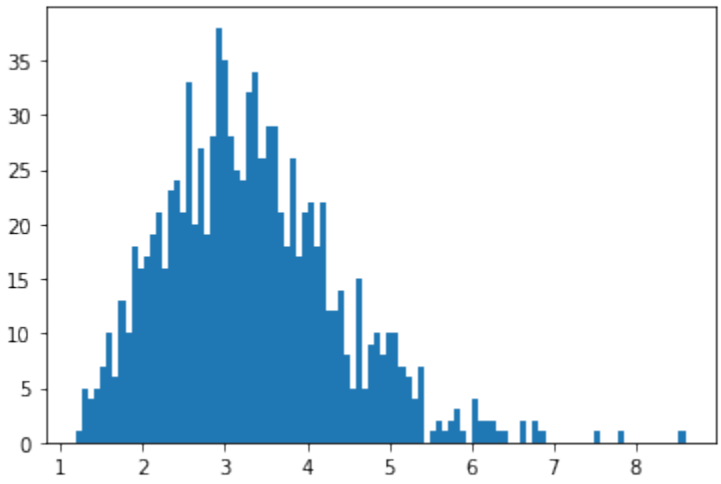
(continues on next page)

(continued from previous page)

```
xx = np.arange(0.01, 10, 0.1).astype(np.float32)

fig = plt.figure(figsize=(10, 6))
for j in range(len(_vars)):
    plt.subplot(2, 3, j + 1)
    for i in range(len(_means)):
        _model = tfd.Gamma(_alpha[i:i+1], _beta[j:j+1])
        _pdf = _model.prob(xx)
        plt.plot(xx, _pdf, label="alpha=%.2f" %(_alpha[i]))
    plt.title("beta=%.2f" %(_beta[j]))
    if j == 5:
        plt.legend(loc="best")

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



[70]:
```
_model = tfd.Gamma([10], [3])
plt.hist(_model.sample(1000).numpy().reshape(-1), bins=100)
plt.show()
```


35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8

[ ]: