evolocity documentation

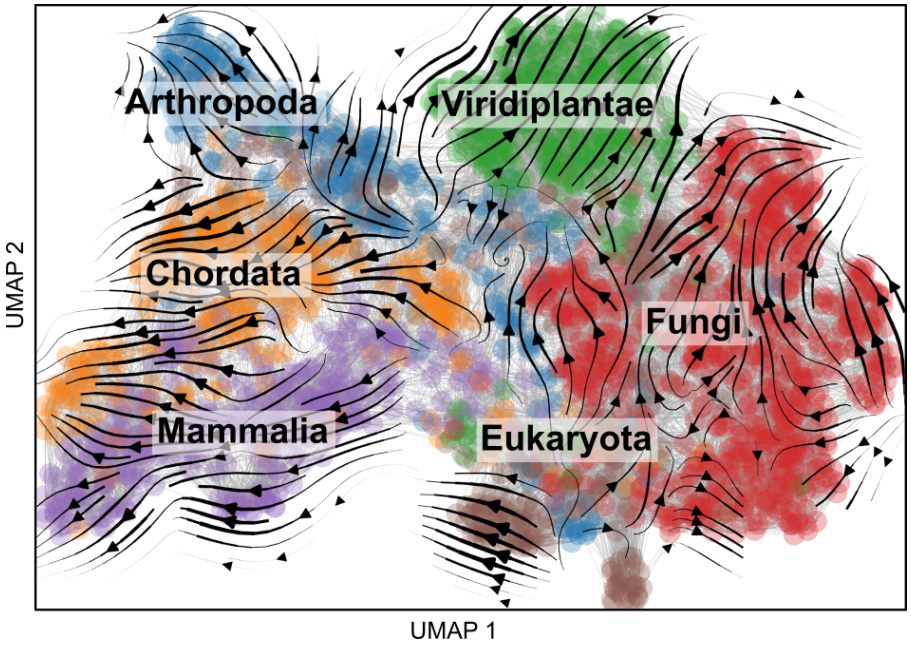
Release 1.0.1

Brian Hie

Mar 02, 2023

MAIN

Evolocity implements evolutionary velocity (evo-velocity), which models a protein sequence landscape as an evolutionary "vector field" by using the local evolutionary predictions enabled by language models to enable global evolutionary insight.



Evo-velocity uses the change in languge model likelihoods to estimate directionality between two biological sequences. Then, over an entire sequence similarity network, this procedure is used to direct network edges. Finally, network diffusion analysis can identify roots, order sequences in pseudotime, and identify mutations driving the velocity.

Evolocity is a fork of the scVelo tool for RNA velocity and relies on many aspects of the Scanpy library for high-dimensional biological data analysis. Like Scanpy and scVelo, evolocity makes use of anndata, an extremely convenient way to store and organize biological data.

CHAPTER

ONE

QUICK START

1.1 Installation

You should be able to install evolocity using `pip`:

```
python -m pip install evolocity
```

1.2 API example

Below is a quick Python example of using evolocity to load and analyze sequences in a FASTA file.

```
import evolocity as evo
import scanpy as sc

# Load sequences and compute language model embeddings.
fasta_fname = 'data.fasta'
adata = evo.pp.featurize_fasta(fasta_fname)

# Construct sequence similarity network.
evo.pp.neighbors(adata)

# Run evolocity analysis.
evo.tl.velocity_graph(adata)

# Embed network and velocities in two-dimensions and plot.
sc.tl.umap(adata)
evo.tl.velocity_embedding(adata)
evo.pl.velocity_embedding_grid(adata)
evo.pl.velocity_embedding_stream(adata)
```

1.2.1 Installation

We recommend Python v3.6 or higher.

PyPI, Virtualenv, or Anaconda

Install evolocity from PyPI using:

```
python -m pip install evolocity
```

If you get a `Permission denied` error, use `python -m pip install evolocity --user` instead.

Dependencies

- anndata - annotated data object.
- scanpy - toolkit for high-dimensional data analysis.
- numpy, scipy, pandas, scikit-learn, matplotlib.

Using fast neighbor search via hnswlib further requires (optional):

```
pip install pybind11 hnswlib
```

evolocity - Evolutionary velocity with protein language models

1.2.2 API

Import evolocity as:

```
import evolocity as evo
```

After reading the data (`evo.pp.featurize_fasta`) or loading an in-built dataset (`evo.datasets.*`), the typical workflow consists of subsequent calls of preprocessing (`evo.pp.*`), analysis tools (`evo.tl.*`), and plotting (`evo.pl.*`).

Preprocessing (pp)

Featurization (language model embedding)

pp.featurize_seqs(seqs[, model_name, mkey, ...])	Embeds a list of sequences.
pp.featurize_fasta(fname[, model_name, ...])	Embeds a FASTA file.

evolocity.pp.featurize_seqs

evolocity.pp.**featurize_seqs**(*seqs*, *model_name='esm1b'*, *mkey='model'*, *embed_batch_size=3000*, *use_cache=False*, *cache_namespace='protein'*)

Embeds a list of sequences.

Takes a list of sequences and returns an `Anndata` object with sequence embeddings in the *adata.X* matrix.

Parameters

seqs : *list* List of protein sequences.

model_name : *str* (default: *'esm1b'*) Language model used to compute likelihoods.

mkey : *str* (default: *'model'*) Name at which language model is stored.

embed_batch_size : *int* (default: *3000*) Batch size to embed sequences. Lower to fit into GPU memory.

use_cache : *bool* (default: *False*) Cache embeddings to disk for faster future loading.

cache_namespace : *str* (default: *'protein'*) Namespace at which to store cache.

Returns

- Returns an `Anndata` object with the attributes
- *.X* – Matrix where rows correspond to sequences and columns are language model embedding dimensions
- **seq** (*.obs*) – Sequences corresponding to rows in *adata.X*
- **model** (*.uns*) – language model

evolocity.pp.featurize_fasta

`evolocity.pp.`**`featurize_fasta`**(*fname*, *model_name='esm1b'*, *mkey='model'*, *embed_batch_size=3000*, *fasta_metadata_record=False*, *use_cache=True*, *cache_namespace=None*)

Embeds a FASTA file.

Takes a FASTA file containing sequences and returns an `Anndata` object with sequence embeddings in the *adata.X* matrix.

An optional argument (*fasta_metadata_record*) allows for loading metadata directly from the FASTA file.

Parameters

fname : *str* Path to FASTA file.

model_name : *str* (default: *'esm1b'*) Language model used to compute likelihoods.

mkey : *str* (default: *'model'*) Name at which language model is stored.

embed_batch_size : *int* (default: *3000*) Batch size to embed sequences. Lower to fit into GPU memory.

fasta_metadata_record : *bool* (default: *False*) If *True*, assumes metadata is storred in FASTA record as *key=value* pairs that are separated by vertical bar "|" characters. Otherwise, does not attempt to load metadata from the FASTA.

use_cache : *bool* (default: *False*) Cache embeddings to disk for faster future loading.

cache_namespace : *str* (default: *'protein'*) Namespace at which to store cache.

Returns

- Returns an `Anndata` object with the attributes
- *.X* – Matrix where rows correspond to sequences and columns are language model embedding dimensions
- **seq** (*.obs*) – Sequences corresponding to rows in *adata.X*
- **model** (*.uns*) – language model

Landscape (nearest neighbors graph construction)

pp.neighbors(adata[, n_neighbors, n_pcs, ...])	Construct sequence similarity neighborhood graph.

evolocity.pp.neighbors

evolocity.pp.**neighbors**(*adata*, *n_neighbors=50*, *n_pcs=None*, *use_rep='X'*, *use_highly_variable=True*, *knn=True*, *random_state=0*, *method='umap'*, *metric='euclidean'*, *metric_kwds=None*, *num_threads=- 1*, *copy=False*)

Construct sequence similarity neighborhood graph.

The neighbor graph methods (umap, hnsw, sklearn) only differ in runtime and yield the same result as scanpy. Connectivities are computed with adaptive kernel width as proposed in Haghverdi et al. 2016 (doi:10.1038/nmeth.3971).

Parameters

adata Annotated data matrix.

n_neighbors The size of local neighborhood (in terms of number of neighboring data points) used for manifold approximation. Larger values result in more global views of the manifold, while smaller values result in more local data being preserved. In general values should be in the range 2 to 100. If *knn* is *True*, number of nearest neighbors to be searched. If *knn* is *False*, a Gaussian kernel width is set to the distance of the *n_neighbors* neighbor.

n_pcs : *int* or *None* (default: None) Number of principal components to use. If not specified, the full space is used of a pre-computed PCA, or 30 components are used when PCA is computed internally.

use_rep : *None*, *'X'* or any key for *.obsm* (default: *'X'*) Use the indicated representation. If *None*, the representation is chosen automatically: for .n_vars < 50, .X is used, otherwise 'X_pca' is used.

use_highly_variable : *bool* (default: True) Whether to use highly variable genes only, stored in .var['highly_variable'].

knn If *True*, use a hard threshold to restrict the number of neighbors to *n_neighbors*, that is, consider a knn graph. Otherwise, use a Gaussian Kernel to assign low weights to neighbors more distant than the *n_neighbors* nearest neighbor.

random_state A numpy random seed.

method : {{'umap', 'hnsw', 'sklearn'}} (default: *'umap'*) Method to compute neighbors, only differs in runtime. The 'hnsw' method is most efficient and requires to *pip install hnswlib*. Connectivities are computed with adaptive kernel.

metric A known metric's name or a callable that returns a distance.

metric_kwds Options for the metric.

num_threads Number of threads to be used (for runtime).

copy Return a copy instead of writing to adata.

Returns

- Depending on *copy*, updates or returns *adata* with the following
- **connectivities** (sparse matrix (*.uns['neighbors']*, dtype *float32*)) – Weighted adjacency matrix of the neighborhood graph of data points. Weights should be interpreted as connectivities.
- **distances** (sparse matrix (*.uns['neighbors']*, dtype *float32*)) – Instead of decaying weights, this stores distances for each pair of neighbors.

Tools (tl)

Velocity estimation

tl.velocity_graph(adata[, model_name, mkey, ...])	Computes velocity scores at each edge in the graph.
tl.velocity_embedding(data[, basis, vkey, ...])	Projects the velocities into any embedding.

evolocity.tl.velocity_graph

evolocity.tl.**velocity_graph**(*adata*, *model_name='esm1b'*, *mkey='model'*, *score='lm'*, *seqs=None*, *vkey='velocity'*, *n_recurse_neighbors=0*, *random_neighbors_at_max=None*, *mode_neighbors='distances'*, *include_set=None*, *copy=False*, *verbose=True*)

Computes velocity scores at each edge in the graph.

At each edge connecting two sequences $(x^{(a)}, x^{(b)})$, computes a score

$$v_{ab} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \left[\log p\left(x_i^{(b)} | x^{(a)}\right) - \log p\left(x_i^{(a)} | x^{(b)}\right) \right]$$

where $\mathcal{M} = \left\{ i : x_i^{(a)} \neq x_i^{(b)} \right\}$ is the set of positions at which the amino acid residues disagree.

Parameters

adata : Anndata Annoated data matrix.

model_name : *str* (default: *'esm1b'*) Language model used to compute likelihoods.

mkey : *str* (default: *'model'*) Name at which language model is stored.

score : *str* (default: *'lm'*) Type of velocity score.

seqs : *list* (default: *'None'*) List of sequences; defaults to those in *adata.obs['seq']*.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

n_recurse_neighbors : *int* (default: *0*) Number of recursions for neighbors search.

random_neighbors_at_max : *int* or *None* (default: *None*) If number of iterative neighbors for an individual node is higher than this threshold, a random selection of such are chosen as reference neighbors.

mode_neighbors : *str* (default: *'distances'*) Determines the type of KNN graph used. Options are 'distances' or 'connectivities'. The latter yields a symmetric graph.

include_set : *set* (default: *None*) Set of characters to explicitly include.

verbose : *bool* (default: *True*) Print logging output.

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

Returns

- Returns or updates *adata* with the attributes
- **model** (*.uns*) – language model
- **velocity_graph** (*.uns*) – sparse matrix with transition probabilities

evolocity.tl.velocity_embedding

evolocity.tl.**velocity_embedding**(*data*, *basis=None*, *vkey='velocity'*, *scale=1*, *self_transitions=True*, *use_negative_cosines=True*, *direct_pca_projection=None*, *retain_scale=False*, *autoscale=True*, *all_comps=True*, *T=None*, *copy=False*)

Projects the velocities into any embedding.

Given normalized difference of the embedding positions $\tilde{\delta}_{ij} = \frac{x_j - x_i}{\|x_j - x_i\|}$. the projections are obtained as expected displacements with respect to the transition matrix $\tilde{\pi}_{ij}$ as

$$\tilde{\nu}_i = E_{\tilde{\pi}_{i\cdot}}[\tilde{\delta}_{i\cdot}] = \sum_{j \neq i} \left(\tilde{\pi}_{ij} - \frac{1}{n} \right) \tilde{\delta}_{ij}.$$

Parameters

data : AnnData Annotated data matrix.

basis : *str* (default: *'tsne'*) Which embedding to use.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

scale : *int* (default: 1) Scale parameter of gaussian kernel for transition matrix.

self_transitions : *bool* (default: *True*) Whether to allow self transitions, based on the confidences of transitioning to neighboring nodes.

use_negative_cosines : *bool* (default: *True*) Whether to project node-to-node transitions with negative cosines into negative/opposite direction.

direct_pca_projection : *bool* (default: *None*) Whether to directly project the velocities into PCA space, thus skipping the velocity graph.

retain_scale : *bool* (default: *False*) Whether to retain scale from high dimensional space in embedding.

autoscale : *bool* (default: *True*) Whether to scale the embedded velocities by a scalar multiplier, which simply ensures that the arrows in the embedding are properly scaled.

all_comps : *bool* (default: *True*) Whether to compute the velocities on all embedding components.

T : *csr_matrix* (default: *None*) Allows the user to directly pass a transition matrix.

copy : *bool* (default: *False*) Return a copy instead of writing to *adata*.

Returns

- Returns or updates *adata* with the attributes
- **velocity_basis** (*.obsm*) – coordinates of velocity projection on embedding

Pseudotime and trajectory inference

tl.terminal_states(data[, vkey, groupby, ...])	Computes terminal states (root and end points).
tl.velocity_pseudotime(adata[, vkey, ...])	Computes pseudotime based on the evolocity graph.

evolocity.tl.terminal_states

evolocity.tl.**terminal_states**(*data*, *vkey='velocity'*, *groupby=None*, *groups=None*, *self_transitions=False*, *eps=0.001*, *random_state=0*, *exp_scale=50*, *copy=False*, ***kwargs*)

Computes terminal states (root and end points).

The end points and root nodes are obtained as stationary states of the velocity-inferred transition matrix and its transposed, respectively, which is given by left eigenvectors corresponding to an eigenvalue of 1, i.e.

$$^{\text{end}} = ^{\text{end}}\pi, \quad ^{\text{root}} = ^{\text{root}}\pi^{\mathrm{T}}.$$

```
evo.tl.terminal_states(adata)
evo.pl.scatter(adata, color=['root_nodes', 'end_points'])
```

Parameters

data : AnnData Annotated data matrix.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider. Only to be set, if each group is assumed to have a distinct lineage with an independent root and end point.

groups : *str*, *list* or *np.ndarray* (default: *None*) Groups selected to find terminal states on. Must be an element of .obs[groupby]. To be specified only for very distinct/disconnected clusters.

self_transitions : *bool* (default: *False*) Allow transitions from one node to itself.

eps : *float* (default: 1e-3) Tolerance for eigenvalue selection.

random_state : *int* or None (default: 0) Seed used by the random number generator. If *None*, use the *RandomState* instance by *np.random*.

copy : *bool* (default: *False*) Return a copy instead of writing to data.

****kwargs** Passed to evolocity.tl.transition_matrix(), e.g. basis, weight_diffusion.

Returns

- Returns or updates *data* with the attributes
- **root_nodes** (*.obs*) – sparse matrix with transition probabilities.
- **end_points** (*.obs*) – sparse matrix with transition probabilities.

evolocity.tl.velocity_pseudotime

evolocity.tl.**velocity_pseudotime**(*adata*, *vkey='velocity'*, *rank_transform=True*, *groupby=None*, *groups=None*, *root_key=None*, *end_key=None*, *use_ends=False*, *n_dcs=10*, *use_velocity_graph=True*, *save_diffmap=None*, *return_model=None*, ***kwargs*)

Computes pseudotime based on the evolocity graph.

Velocity pseudotime is a random-walk based distance measures on the velocity graph. After computing a distribution over root cells obtained from the velocity-inferred transition matrix, it measures the average number

of steps it takes to reach a cell after start walking from one of the root cells. Contrarily to diffusion pseudotime, it implicitly infers the root cells and is based on the directed velocity graph instead of the similarity-based diffusion kernel.

Parameters

adata : AnnData Annotated data matrix

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

rank_transform : *bool* (default: *True*) Perform final rank transformation.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider.

groups : *str*, *list* or *np.ndarray* (default: *None*) Groups selected to find terminal states on. Must be an element of adata.obs[groupby]. Only to be set, if each group is assumed to have a distinct lineage with an independent root and end point.

root_key : *int* (default: *None*) Index of root cell to be used. Computed from velocity-inferred transition matrix if not specified.

end_key : *int* (default: *None*) Index of end point to be used. Computed from velocity-inferred transition matrix if not specified.

n_dcs : *int* (default: 10) The number of diffusion components to use.

use_velocity_graph : *bool* (default: *True*) Whether to use the velocity graph. If False, it uses the similarity-based diffusion kernel.

save_diffmap : *bool* (default: *None*) Whether to store diffmap coordinates.

return_model : *bool* (default: *None*) Whether to return the vpt object for further inspection.

****kwargs** Further arguments to pass to VPT (e.g. min_group_size, allow_kendall_tau_shift).

Returns

- Updates *adata* with the attributes
- **velocity_pseudotime** (*.obs*) – Velocity pseudotime obtained from velocity graph.

Interpretation

tl.onehot_msa(adata[, reference, ...])	Aligns and one-hot-encodes sequences.
tl.residue_scores(adata[, basis, scale, ...])	Score mutations by associated evolocity.
tl.random_walk(data[, root_node, ...])	Runs a random walk on the evolocity graph.

evolocity.tl.onehot_msa

evolocity.tl.**onehot_msa**(*adata*, *reference=None*, *seq_id_fields=None*, *key='onehot'*, *seq_key='seq'*, *backend='mafft'*, *dirname='target/evolocity_alignments'*, *n_threads=1*, *copy=False*)

Aligns and one-hot-encodes sequences.

By default, uses the MAFFT aligner (https://mafft.cbrc.jp/alignment/software/), which can be installed via conda using

```
conda install -c bioconda mafft
```

Parameters

adata : Anndata Annoated data matrix.

reference : *int* (default: None) Index corresponding to a sequence in *adata* to be used as the main reference sequence for the alignment.

seq_id_fields : *list* (default: None) List of fields in *adata.obs* to store in FASTA IDs.

key : *str* (default: *'onehot'*) Name at which the embedding is stored.

seq_key : *str* (default: *'seq'*) Name of sequences in *.obs*.

backend : *str* (default: *None*) Sequence alignment tool.

dirname : *str* (default: *'target/evolocity_alignments'*) Directory under which to place alignment files.

n_threads : *int* (default: 1) Number of threads for sequence alignment.

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

Returns

- Returns or updates *adata* with the attributes
- **X_onehot** (*.obsm*) – one-hot embeddings
- **seqs_msa** (*.obs*) – aligned sequences

evolocity.tl.residue_scores

evolocity.tl.**residue_scores**(*adata*, *basis='onehot'*, *scale=1.0*, *key='residue_scores'*, *copy=False*)

Score mutations by associated evolocity.

Requires running `evo.tl.onehot_msa` first.

Parameters

adata : Anndata Annoated data matrix.

basis : *str* (default: *'onehot'*) Name of one-hot embedding

scale : *float* (default: *1.*) Scale parameter of gaussian kernel for transition matrix.

key : *str* (default: *'residue_scores'*) Name at which to place scores.

copy : *bool* (default: *False*) Return a copy instead of writing to adata.

Returns

- Returns or updates *adata* with the attributes
- **residue_scores** (*.uns*) – per-residue velocity scores

evolocity.tl.random_walk

evolocity.tl.**random_walk**(*data*, *root_node=0*, *walk_length=10*, *n_walks=1*, *forward_walk=True*, *path_key='rw_paths'*, *vkey='velocity'*, *groupby=None*, *groups=None*, *self_transitions=False*, *eps=0.001*, *random_state=0*, *copy=False*, ***kwargs*)

Runs a random walk on the evolocity graph.

Parameters

data : AnnData Annotated data matrix.

root_node : *int* (default: *0*) Index of node at which to start random walk.

walk_length : *int* (default: *10*) Number of steps in walk.

n_walks : *int* (default: *1*) Number of walks to take.

forward_walk : *bool* (default: *True*) Whether to go in the same or reverse direction of evolocity.

path_key : *str* (default: *'rw_paths'*) Name at which to store the random walks.

vkey : *str* (default: *'velocity'*) Name of velocity estimates to be used.

groupby : *str*, *list* or *np.ndarray* (default: *None*) Key of observations grouping to consider. Only to be set, if each group is assumed to have a distinct lineage with an independent root and end point.

groups : *str*, *list* or *np.ndarray* (default: *None*) Groups selected to find terminal states on. Must be an element of .obs[groupby]. To be specified only for very distinct/disconnected clusters.

self_transitions : *bool* (default: *False*) Allow transitions from one node to itself.

eps : *float* (default: 1e-3) Tolerance for eigenvalue selection.

random_state : *int* or None (default: 0) Seed used by the random number generator. If *None*, use the *RandomState* instance by *np.random*.

copy : *bool* (default: *False*) Return a copy instead of writing to data.

****kwargs** Passed to evolocity.tl.transition_matrix(), e.g. scale, basis.

Returns

- Returns or updates *data* with the attributes
- **rw_paths** (*.uns*) – rows of matrix correspond to random walks, columns correspond to steps

Plotting (pl)

Also see scanpy's plotting API for additional visualization functionality, including UMAP scatter plots.

Velocity embeddings

pl.velocity_embedding(adata[, basis, vkey, ...])	Scatter plot of velocities on the embedding.
pl.velocity_embedding_grid(adata[, basis, ...])	Scatter plot of velocities on a grid.
pl.velocity_embedding_stream(adata[, basis, ...])	Stream plot of velocities on the embedding.
pl.velocity_contour(adata[, ptkey, ...])	Contour plot of pseudotime with velocity grid.

evolocity.pl.velocity_embedding

evolocity.pl.**velocity_embedding**(*adata*, *basis=None*, *vkey='velocity'*, *density=None*, *arrow_size=None*, *arrow_length=None*, *scale=None*, *X=None*, *V=None*, *recompute=None*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.2*, *perc=None*, *sort_order=True*, *groups=None*, *components=None*, *projection='2d'*, *legend_loc='none'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Scatter plot of velocities on the embedding.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

arrow_size : *float* or triple *headlength, headwidth, headaxislength* (default: 1) Size of arrows.

arrow_length : *float* (default: 1) Length of arrows.

scale : *float* (default: 1) Length of velocities in the embedding.

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations or variables

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth : *bool* or *int* (default: *None*) Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

evolocity.pl.velocity_embedding_grid

evolocity.pl.**velocity_embedding_grid**(*adata*, *basis=None*, *vkey='velocity'*, *density=None*, *smooth=None*, *min_mass=None*, *arrow_size=None*, *arrow_length=None*, *arrow_color=None*, *scale=None*, *autoscale=True*, *n_neighbors=None*, *recompute=None*, *X=None*, *V=None*, *X_grid=None*, *V_grid=None*, *principal_curve=False*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.2*, *perc=None*, *sort_order=True*, *groups=None*, *components=None*, *projection='2d'*, *legend_loc='none'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Scatter plot of velocities on a grid.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

arrow_size : *float* or triple *headlength, headwidth, headaxislength* (default: 1) Size of arrows.

arrow_length : *float* (default: 1) Length of arrows.

scale : *float* (default: 1) Length of velocities in the embedding.

min_mass : *float* or *None* (default: *None*) Minimum threshold for mass to be shown. It can range between 0 (all velocities) and 100 (large velocities).

smooth : *bool* or *int* (default: *None*) Multiplication factor for scale in Gaussian kernel around grid point.

n_neighbors : *int* (default: None) Number of neighbors to consider around grid point.

X : *np.ndarray* (default: None) embedding grid point coordinates

V : *np.ndarray* (default: None) embedding grid velocity coordinates

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations or variables

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth : *float* (default: 1) Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

evolocity.pl.velocity_embedding_stream

evolocity.pl.**velocity_embedding_stream**(*adata*, *basis=None*, *vkey='velocity'*, *density=None*, *smooth=None*, *min_mass=None*, *cutoff_perc=None*, *arrow_color=None*, *linewidth=None*, *n_neighbors=None*, *recompute=None*, *color=None*, *use_raw=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.3*, *perc=None*, *X=None*, *V=None*, *X_grid=None*, *V_grid=None*, *sort_order=True*, *groups=None*, *components=None*, *legend_loc='on data'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Stream plot of velocities on the embedding.

Parameters

adata : AnnData Annotated data matrix.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

smooth : *bool* or *int* (default: *None*) Multiplication factor for scale in Gaussian kernel around grid point.

min_mass : *float* (default: 1) Minimum threshold for mass to be shown. It can range between 0 (all velocities) and 5 (large velocities only).

cutoff_perc : *float* (default: *None*) If set, mask small velocities below a percentile threshold (between 0 and 100).

linewidth : *float* (default: 1) Line width for streamplot.

n_neighbors : *int* (default: None) Number of neighbors to consider around grid point.

X : *np.ndarray* (default: None) Embedding grid point coordinates

V : *np.ndarray* (default: None) Embedding grid velocity coordinates

basis : *str* or list of *str* (default: *None*) Key for embedding. If not specified, use 'umap', 'tsne' or 'pca' (ordered by preference).

vkey : *str* or list of *str* (default: *None*) Key for velocity / steady-state ratio to be visualized.

color : *str*, list of *str* or *None* (default: *None*) Key for annotations of observations or variables

use_raw : *bool* (default: *None*) Use *raw* attribute of *adata* if present.

layer : *str*, list of *str* or *None* (default: *None*) Specify the layer for *color*.

color_map : *str* (default: *matplotlib.rcParams['image.cmap']*) String denoting matplotlib color map.

colorbar : *bool* (default: *False*) Whether to show colorbar.

palette : list of *str* (default: *None*) Colors to use for plotting groups (categorical annotation).

size : *float* (default: 5) Point size.

alpha : *float* (default: 1) Set blending - 0 transparent to 1 opaque.

linewidth Scaling factor for the width of occurring lines.

linecolor : *str* ir list of *str* (default: 'k') Color of lines from velocity fits, linear fits and polynomial fits

perc : tuple, e.g. [2,98] (default: *None*) Specify percentile for continuous coloring.

groups : *str* or list of *str* (default: *all groups*) Restrict to a few categories in categorical observation annotation. Multiple categories can be passed as list with ['cluster_1', 'cluster_3'], or as string with 'cluster_1, cluster_3'.

sort_order : *bool* (default: *True*) For continuous annotations used as color parameter, plot data points with higher values on top of others.

components : *str* or list of *str* (default: '1,2') For instance, ['1,2', '2,3'].

projection : *{'2d', '3d'} (default: '2d')* Projection of plot.

legend_loc : *str (default: 'none')* Location of legend, either 'on data', 'right margin' or valid keywords for matplotlib.legend.

legend_fontsize : *int* (default: *None*) Legend font size.

legend_fontweight : {'normal', 'bold', ...} (default: *None*) Legend font weight. A numeric value in range 0-1000 or a string. Defaults to 'bold' if *legend_loc = 'on data'*, otherwise to 'normal'. Available are *['light', 'normal', 'medium', 'semibold', 'bold', 'heavy', 'black']*.

legend_fontoutline Line width of the legend font outline in pt. Draws a white outline using the path effect `withStroke`.

right_margin : *float* or list of *float* (default: *None*) Adjust the width of the space right of each plotting panel.

left_margin : *float* or list of *float* (default: *None*) Adjust the width of the space left of each plotting panel.

xlabel : *str* (default: *None*) Label of x-axis.

ylabel : *str* (default: *None*) Label of y-axis.

title : *str* (default: *None*) Provide title for panels either as, e.g. *["title1", "title2", ...]*.

fontsize : *float* (default: *None*) Label font size.

figsize : tuple (default: *(7,5)*) Figure size.

xlim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict x-limits of the axis.

ylim : tuple, e.g. [0,1] or *None* (default: *None*) Restrict y-limits of the axis.

add_density : *bool* or *str* or *None* (default: *None*) Whether to show density of values along x and y axes. Color of the density plot can also be passed as *str*.

add_assignments : *bool* or *str* or *None* (default: *None*) Whether to add assignments to the model curve. Color of the assignments can also be passed as *str*.

add_linfit : *bool* or *str* or *None* (default: *None*) Whether to add linear regression fit to the data points. Color of the line can also be passed as *str*. Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_polyfit : *bool* or *str* or *int* or *None* (default: *None*) Whether to add polynomial fit to the data points. Color of the polyfit plot can also be passed as *str*. The degree of the polynomial fit can be passed as *int* (default is 2 for quadratic fit). Fitting with or without an intercept by passing *'intercept'* or *'no_intercept'*. A colored regression line with intercept is obtained with *'intercept, blue'*.

add_rug : *str* or *None* (default: *None*) If categorical observation annotation (e.g. 'clusters') is given, a rugplot is attached to the x-axis showing the data membership to each of the categories.

add_text : *str* (default: *None*) Text to be added to the plot, passed as *str*.

add_text_pos : *tuple*, e.g. [0.05, 0.95] (defaut: *[0.05, 0.95]*) Text position. Default is *[0.05, 0.95]*, positioning the text at top right corner.

add_outline : *bool* or *str* (default: *False*) Whether to show an outline around scatter plot dots. Alternatively a string of cluster names can be passed, e.g. 'cluster_1, clusters_3'.

outline_width : tuple type *scalar* or *None* (default: *(0.3, 0.05)*) Width of the inner and outer outline

outline_color : tuple of type *str* or *None* (default: *('black', 'white')*) Inner and outer matplotlib color of the outline

n_convolve : *int* or *None* (default: *None*) If *int* is given, data is smoothed by convolution along the x-axis with kernel size *n_convolve*.

smooth Whether to convolve/average the color values over the nearest neighbors. If *int*, it specifies number of neighbors.

rescale_color : *tuple* (default: *None*) Boundaries for color rescaling, e.g. [0, 1], setting min/max values of the colorbar.

color_gradients : *str* or *np.ndarray* (default: *None*) Key for *.obsm* or array with color gradients by categories.

dpi : *int* (default: 80) Figure dpi.

frameon : *bool* (default: *True*) Draw a frame around the scatter plot.

ncols : *int* (default: *None*) Number of panels per row.

nrows : *int* (default: *None*) Number of panels per column.

wspace : *float* (default: None) Adjust the width of the space between multiple panels.

hspace : *float* (default: None) Adjust the height of the space between multiple panels.

show : *bool*, optional (default: *None*) Show the plot, do not return axis.

save : *bool* or *str*, optional (default: *None*) If *True* or a *str*, save the figure. A string is appended to the default filename. Infer the filetype if ending on {'.pdf', '.png', '.svg'}.

ax : *matplotlib.Axes*, optional (default: *None*) A matplotlib axes object. Only works if plotting a single component.

Returns *matplotlib.Axis* if *show==False*

evolocity.pl.velocity_contour

evolocity.pl.**velocity_contour**(*adata*, *ptkey='pseudotime'*, *rank_transform=True*, *use_ends=False*, *fill=True*, *levels=10*, *basis=None*, *vkey='velocity'*, *density=None*, *smooth=None*, *pt_smooth=None*, *min_mass=None*, *arrow_size=None*, *arrow_length=None*, *arrow_color=None*, *scale=None*, *autoscale=True*, *n_neighbors=None*, *recompute=None*, *X=None*, *V=None*, *X_grid=None*, *V_grid=None*, *PF_grid=None*, *color=None*, *layer=None*, *color_map=None*, *colorbar=True*, *palette=None*, *size=None*, *alpha=0.5*, *offset=1*, *vmin=None*, *vmax=None*, *perc=None*, *sort_order=True*, *groups=None*, *components=None*, *projection='2d'*, *legend_loc='none'*, *legend_fontsize=None*, *legend_fontweight=None*, *xlabel=None*, *ylabel=None*, *title=None*, *fontsize=None*, *figsize=None*, *dpi=None*, *frameon=None*, *show=None*, *save=None*, *ax=None*, *ncols=None*, ***kwargs*)

Contour plot of pseudotime with velocity grid.

Parameters

adata : AnnData Annotated data matrix.

ptkey : *str* (default: *pseudotime*) Name of pseudotime values.

rank_transform : *bool* (default: *True*) Perform final rank transformation.

use_ends : *bool* (default: *False*) Use end terminal nodes in pseudotime computation.

levels : *int* (default: 10) Number of contour levels.

pt_smooth : *float* (default: *None*) Pseudotime two-dimensional smoothing.

density : *float* (default: 1) Amount of velocities to show - 0 none to 1 all

arrow_size : *float* or triple *headlength, headwidth, headaxislength* (default: 1) Size of arrows.

arrow_length : *float* (default: 1) Length of arrows.

scale : *float* (default: 1) Length of velocities in the embedding.

min_mass : *float* or *None* (default: *None*) Minimum threshold for mass to be shown. It can range between 0 (all velocities) and 100 (large velocities).

smooth : *float* (default: 0.5) Multiplication factor for scale in Gaussian kernel around grid point.

n_neighbors : *int* (default: None) Number of neighbors to consider around grid point.

X : *np.ndarray* (default: None) embedding grid point coordinates

V : *np.ndarray* (default: None) embedding grid velocity coordinates

{scatter}

Returns *matplotlib.Axis* if *show==False*

Mutation interpretation

pl.residue_scores(adata[, percentile_keep, ...])	Heat map of per-residue velocity scores.
pl.residue_categories(adata[, positions, ...])	Scatter plot of mutations.

evolocity.pl.residue_scores

evolocity.pl.**residue_scores**(*adata*, *percentile_keep=0.0*, *basis='onehot'*, *key='residue_scores'*, *cmap='RdBu'*, *save=None*)

Heat map of per-residue velocity scores.

Visualize scores generated by `evo.tl.residue_scores`.

Parameters

adata : AnnData Annotated data matrix.

percentile_keep : *float* (default: *0.*) Filter out sites below this percentile.

basis : *str* (default: *'onehot'*) Name of one-hot embedding.

cmap : *str* (default: *'RdBu'*) Color map to use.

Returns *matplotlib.Axis* if *save* is not *None*

evolocity.pl.residue_categories

evolocity.pl.**residue_categories**(*adata*, *positions=None*, *n_plot=5*, *namespace='residue_categories'*, *reference=None*, *verbose=True*)

Scatter plot of mutations.

Plots the mutations with the strongest association with evolocity in a scatter plot. Requires running `evo.tl.residue_scores` first.

Parameters

adata : AnnData Annotated data matrix.

positions : *list* (default: *None*) Positions to visualize; if *None*, will use the *n_plot* top positions.

n_plot : *int* (default: *5*) Number of positions to plot.

namespace : *str* (default: *'residue_categories'*) Namespace for generated plot files.

verbose : *bool* (default: *True*) Print information about plotted sites.

Datasets

datasets.nucleoprotein()	Influenza A nucleoprotein.
datasets.cytochrome_c()	Eukaryotic cytochrome c.

evolocity.datasets.nucleoprotein

evolocity.datasets.**nucleoprotein**()
Influenza A nucleoprotein.

Data downloaded from the NIAID Influenza Research Database (https://www.fludb.org/). Most sequences include metadata on the sampling year and the influenza subtype.

The sequence landscape shows structure according to both sampling year and subtype.

Returns Returns *adata* object

evolocity.datasets.cytochrome_c

evolocity.datasets.**cytochrome_c**()
Eukaryotic cytochrome c.

Data downloaded from UniProt (https://www.uniprot.org/). Sequences are from the "cytochrome c" family and filtered to preserve the largest mode in sequence lengths and to preserve eukyarotic proteins.

The sequence landscape capture the diversification of the eukaryota.

Returns Returns *adata* object

Settings

set_figure_params([style, dpi, dpi_save, . . .])	Set resolution/size, styling and format of figures.

evolocity.set_figure_params

evolocity.**set_figure_params**(*style='evolocity'*, *dpi=100*, *dpi_save=300*, *frameon=None*, *vector_friendly=True*, *transparent=False*, *fontsize=12*, *figsize=None*, *color_map=None*, *facecolor=None*, *format='pdf'*, *ipython_format='png2x'*)
Set resolution/size, styling and format of figures.

Parameters

style : *str* (default: *None*) Init default values for `matplotlib.rcParams` suited for

evolocity or *scanpy*. Use *None* for the default matplotlib values.

dpi : *int* (default: *None*) Resolution of rendered figures - affects the size of figures in notebooks.

dpi_save : *int* (default: *None*) Resolution of saved figures. This should typically be higher to achieve publication quality.

frameon : *bool* (default: *None*) Add frames and axes labels to scatter plots.

vector_friendly : *bool* (default: *True*) Plot scatter plots using *png* backend even when exporting as *pdf* or *svg*.

transparent : *bool* (default: *False*) Save figures with transparent background. Sets *rcParams['savefig.transparent']*.

fontsize : *int* (default: 14) Set the fontsize for several *rcParams* entries.

figsize : *[float, float]* (default: *None*) Width and height for default figure size.

color_map : *str* (default: *None*) Convenience method for setting the default color map.

facecolor : *str* (default: *None*) Sets backgrounds *rcParams['figure.facecolor']* and *rcParams['axes.facecolor']* to *facecolor*.

format : *{'png', 'pdf', 'svg', etc.} (default: 'pdf')* This sets the default format for saving figures: *file_format_figs*.

ipython_format : list of *str* (default: 'png2x') Only concerns the notebook/IPython environment; see *IPython.core.display.set_matplotlib_formats* for more details.

1.2.3 References

Code: Github repo

Paper: "Evolutionary velocity with protein language models predicts evolutionary dynamics of diverse proteins" by Brian Hie, Kevin Yang, and Peter Kim.

1.2.4 Evolocity of influenza A nucleoprotein

Here, we will walk through the steps for running an evolocity analysis on nucleoprotein sequences as described in our paper. This notebook is also available on Colab.

First, we will need to install scanpy and evolocity:

[ ]:
```
!pip install scanpy evolocity
```

Now, import the necessary packages

[13]:
```
import evolocity as evo
import scanpy as sc
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as ss
```

We have provided the nucleoprotein sequences, and their corresponding language model embeddings, within an AnnData object, which is a convenient way to store high-dimensional biological data.

We have also already constructed the nearest neighbors graph, which we used to learn a two-dimensional UMAP embedding and perform graph-based Louvain clustering. We have also provided the associated metadata including subtype and sampling year.

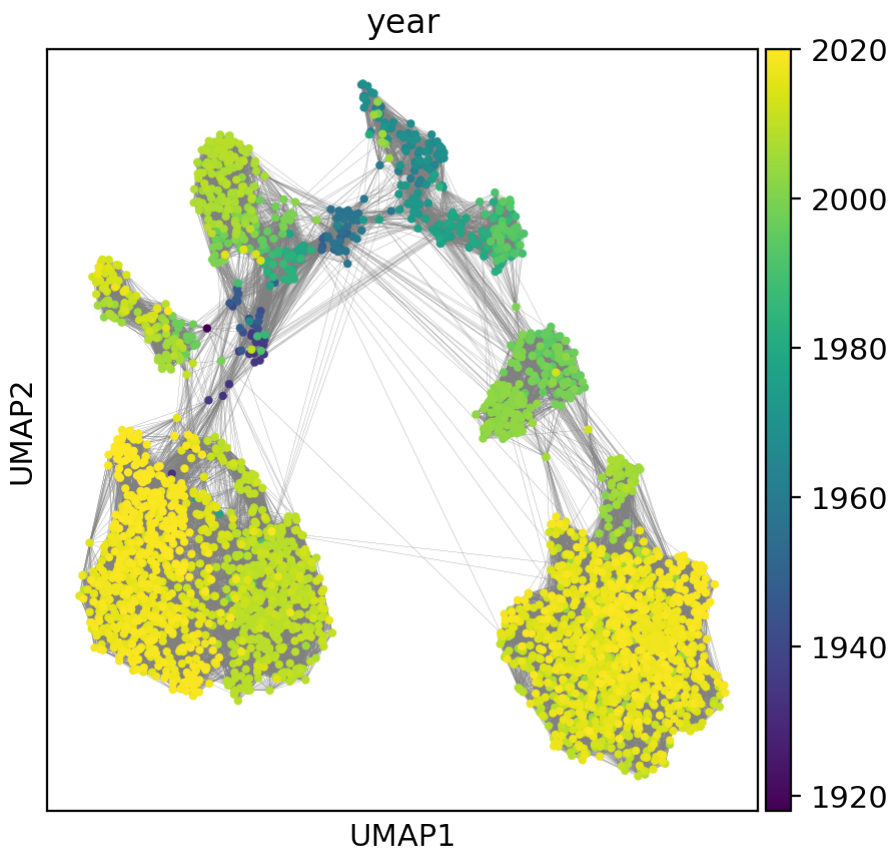
[6]:
```
adata = evo.datasets.nucleoprotein()
adata
```

```
HBox(children=(FloatProgress(value=0.0, max=22765936.0), HTML(value='')))
```

[6]:
```
AnnData object with n_obs × n_vars = 3304 × 1280
    obs: 'n_seq', 'seq', 'gene_id', 'embl_id', 'subtype', 'year', 'date', 'country',
↪'host', 'resist_adamantane', 'resist_oseltamivir', 'virulence', 'transmission',
↪'seqlen', 'homology', 'gong2013_step', 'louvain'
    uns: 'louvain', 'neighbors', 'umap'
    obsm: 'X_umap'
    obsp: 'connectivities', 'distances'
```

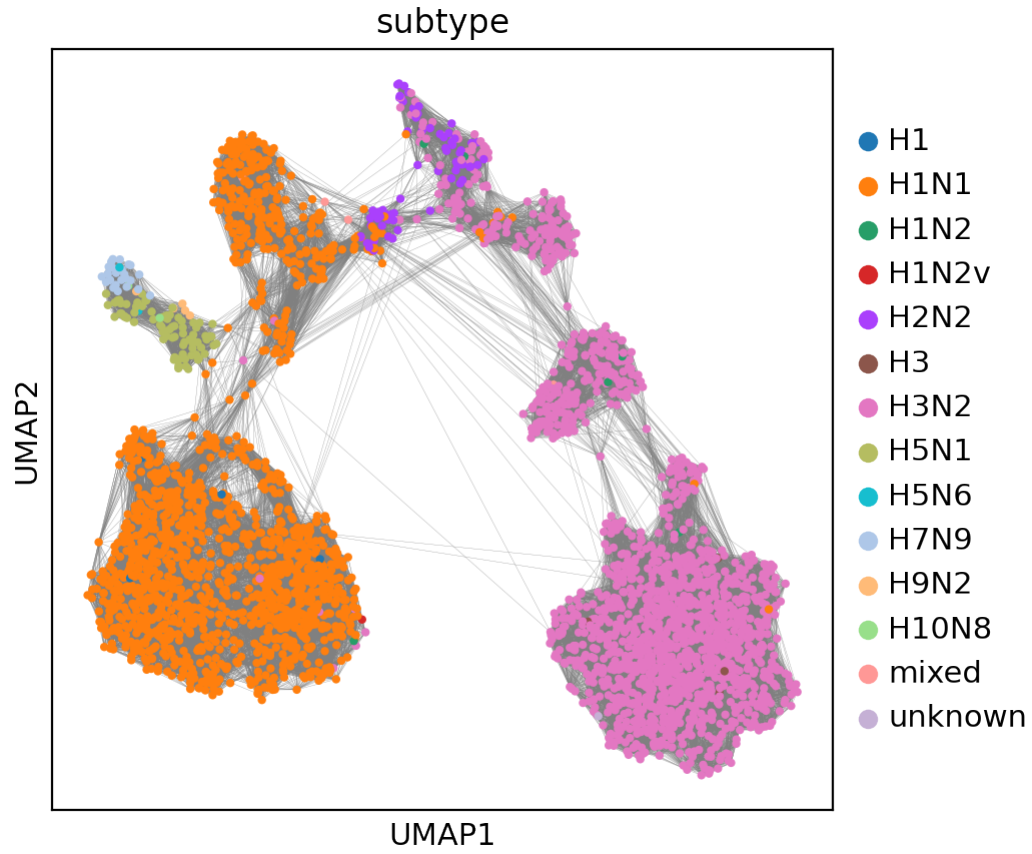
Now, we can use scanpy to plot the UMAP and color according to sampling year.

[7]:
```
evo.set_figure_params(dpi_save=500, figsize=(5, 5))
sc.pl.umap(adata, color='year', edges=True,)
```



We can also color the same UMAP according to subtype:

[6]:
```
sc.pl.umap(adata, color='subtype', edges=True,)
```



Now we need to install and download the ESM-1b language model. We first install the package:

[8]:
```
!pip install fair-esm
```

```
Collecting fair-esm
  Downloading https://files.pythonhosted.org/packages/b7/40/
↪3e757e044a9283a9b80b872aab32246fb0e4d725a57a0ef6ab7b9ef5e1c9/fair_esm-0.3.1-py3-
↪none-any.whl
Installing collected packages: fair-esm
Successfully installed fair-esm-0.3.1
```

Now we need to use evolocity to compute scores for each edge in the nearest neighbors graph.

The command below is the most time-consuming step of this tutorial. First, the command below will download ESM-1b (which takes about 25 minutes).

Then, it will compute the sequence likelihoods and the velocity scores. **Likelihood computation runs much faster with GPU hardware acceleration** (which you can set by going to "Edit" -> "Notebook settings"). Likelihood computation takes about 25 minutes (on a GPU) and score computation takes about 40 minutes.

[9]:
```
evo.tl.velocity_graph(adata)
```

```
Downloading: "https://dl.fbaipublicfiles.com/fair-esm/models/esm1b_t33_650M_UR50S.pt"␣
↪to /root/.cache/torch/hub/checkpoints/esm1b_t33_650M_UR50S.pt
Downloading: "https://dl.fbaipublicfiles.com/fair-esm/regression/esm1b_t33_650M_UR50S-
↪contact-regression.pt" to /root/.cache/torch/hub/checkpoints/esm1b_t33_650M_UR50S-
↪contact-regression.pt
```

(continues on next page)

(continued from previous page)

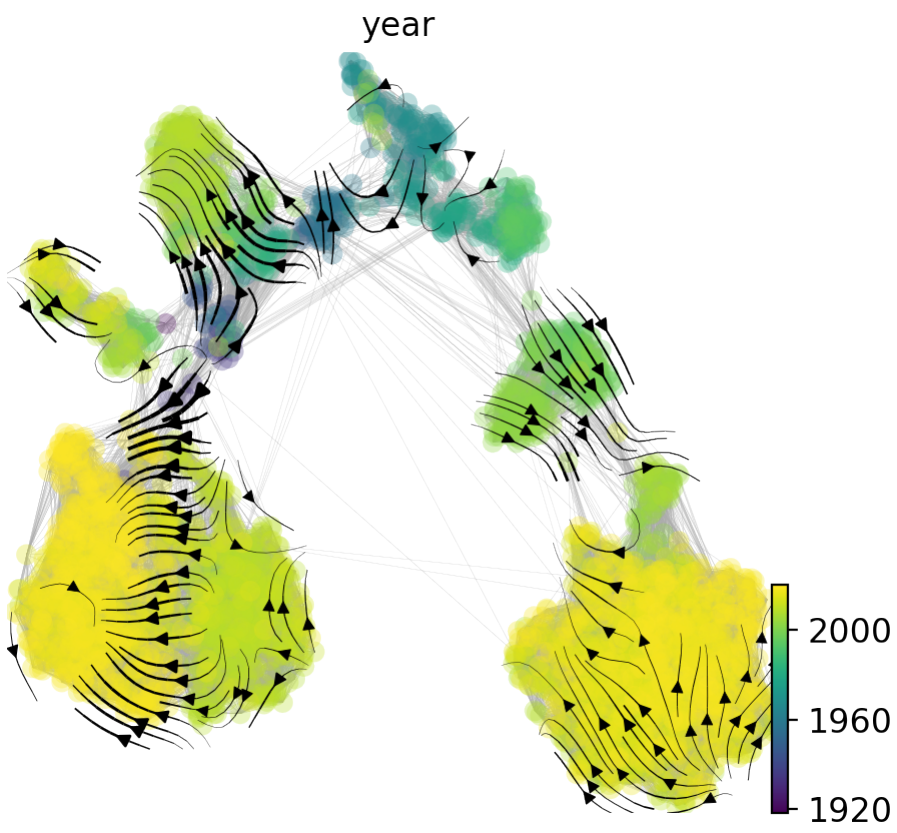
```
100%|| 3304/3304 [26:33<00:00,  2.07it/s]
  0%|          | 0/3304 [00:00<?, ?it/s]
```

```
100%|| 3304/3304 [38:49<00:00,  1.42it/s]
```

Now, we can project the velocities into two-dimensional UMAP space and visualize as a streamplot:

[15]:
```
evo.tl.velocity_embedding(adata, basis='umap', scale=1.)
ax = evo.pl.velocity_embedding_stream(
    adata, basis='umap', min_mass=4., smooth=1., density=1.2,
    color='year', show=False,
)
sc.pl._utils.plot_edges(ax, adata, 'umap', 0.1, '#aaaaaa')
```

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



We can also compute terminal states (the root and end nodes) according to a diffusion process and visualize them:

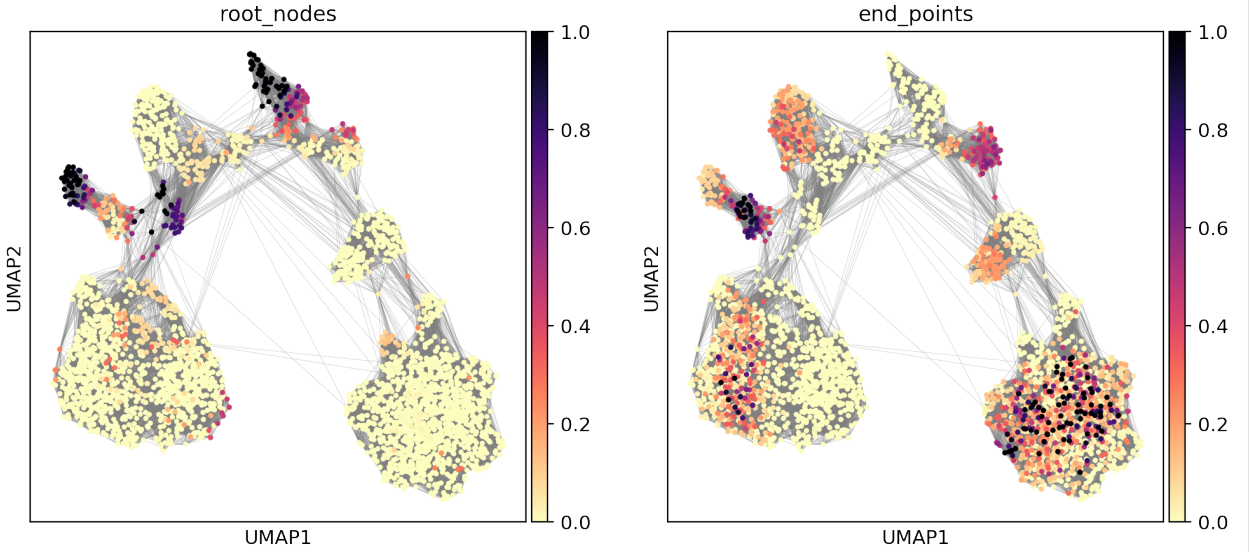
[38]:
```
evo.tl.terminal_states(adata)
sc.pl.umap(
    adata, color=[ 'root_nodes', 'end_points' ],
    color_map=plt.cm.get_cmap('magma').reversed(),
```

(continues on next page)

(continued from previous page)

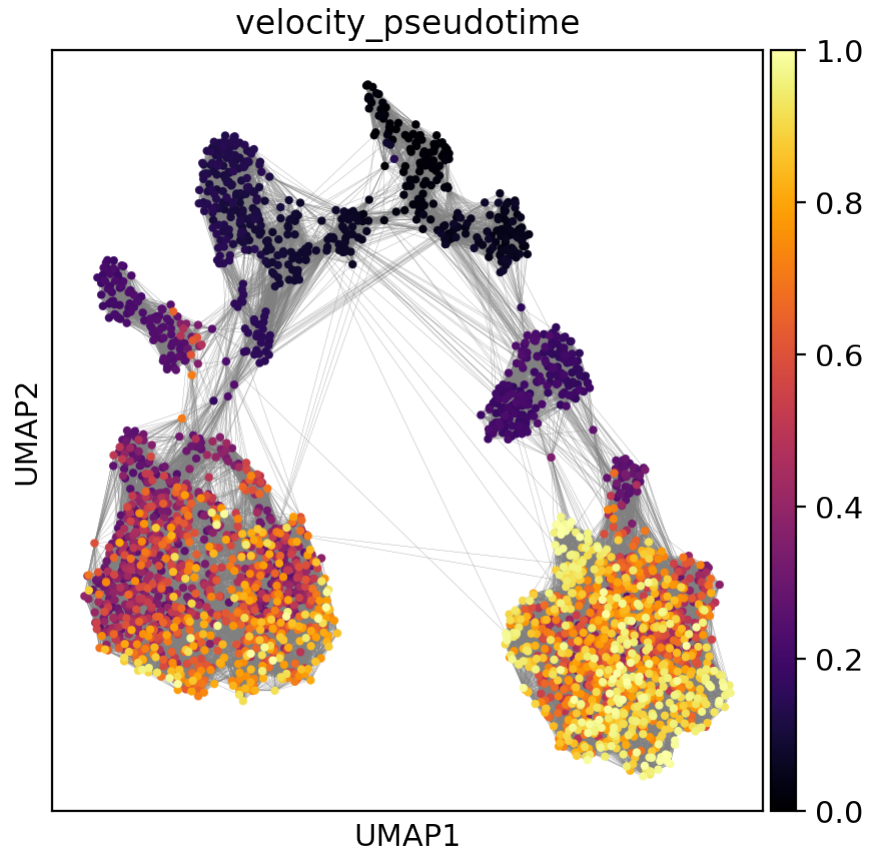
```
    edges=True,
)
```

```
computing terminal states
    identified 7 regions of root nodes and 5 regions of end points .
    finished (0:00:00) --> added
    'root_nodes', root nodes of Markov diffusion process (adata.obs)
    'end_points', end points of Markov diffusion process (adata.obs)
```



Now, we can use the root nodes to compute and visualize diffusion pseudotime on the graph:

[39]:
```
evo.tl.velocity_pseudotime(adata)
sc.pl.umap(
    adata, color='velocity_pseudotime', edges=True, cmap='inferno',
)
```



Finally, we can compute the correlation between pseudotime and sampling time to confirm that evolocity captures the temporal evolution of nucleoprotein!

[40]:
```
nnan_idx = (np.isfinite(adata.obs['year']) &
            np.isfinite(adata.obs['velocity_pseudotime']))

adata_nnan = adata[nnan_idx]

print('Pseudotime-time Spearman r = {}, P = {}'
      .format(*ss.spearmanr(adata_nnan.obs['velocity_pseudotime'],
                            adata_nnan.obs['year'],
                            nan_policy='omit')))
```

```
Pseudotime-time Spearman r = 0.48792075278872044, P = 4.1389753064609613e-197
```

1.2.5 Evolocity of cytochrome c

Here we will analyze cytochrome c sequences from eukaryotes as obtained from UniProt. This tutorial is also available on Colab.

Also, see our tutorial on how to perform evolocity analysis on influenza A nucleoprotein for a more detailed walk-through.

First, we need to install the required packages:

[ ]:
```
!pip install scanpy evolocity
```

Now we need to import the packages:

[2]:
```
import evolocity as evo
import matplotlib.pyplot as plt
import numpy as np
import scanpy as sc
import seaborn as sns
```

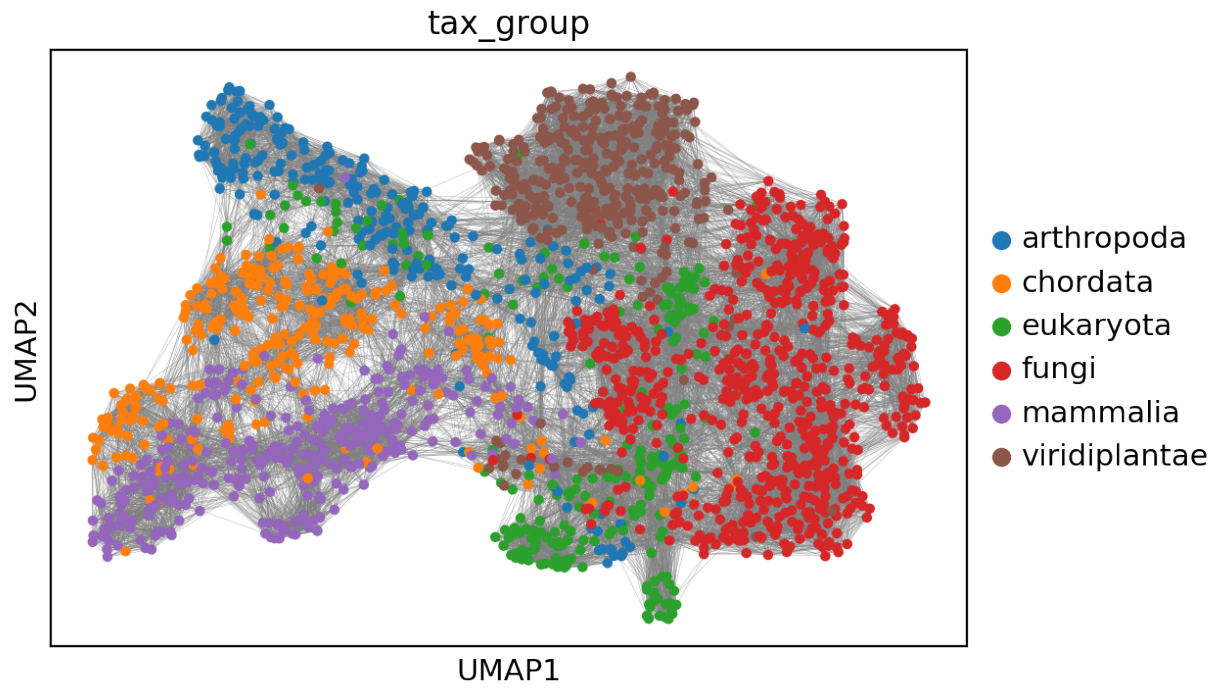
And now we can download the data for cytochrome c.

We have already computed the language model embeddings for these sequences and also parsed the metadata in the `'tax_group'` entry in `adata.obs`. We have also constructed the nearest neighbors graph and run the UMAP algorithm to embed the graph into two dimensions.

[ ]:
```
adata = evo.datasets.cytochrome_c()
```

Now, we can plot the graph in UMAP space colored by taxonomy.

[4]:
```
evo.set_figure_params(dpi_save=500, figsize=(6, 4))
sc.pl.umap(adata, color='tax_group', edges=True,)
```



Now we need to install ESM-1b so we can compute the velocity scores

[ ]:
```
!pip install fair-esm
```

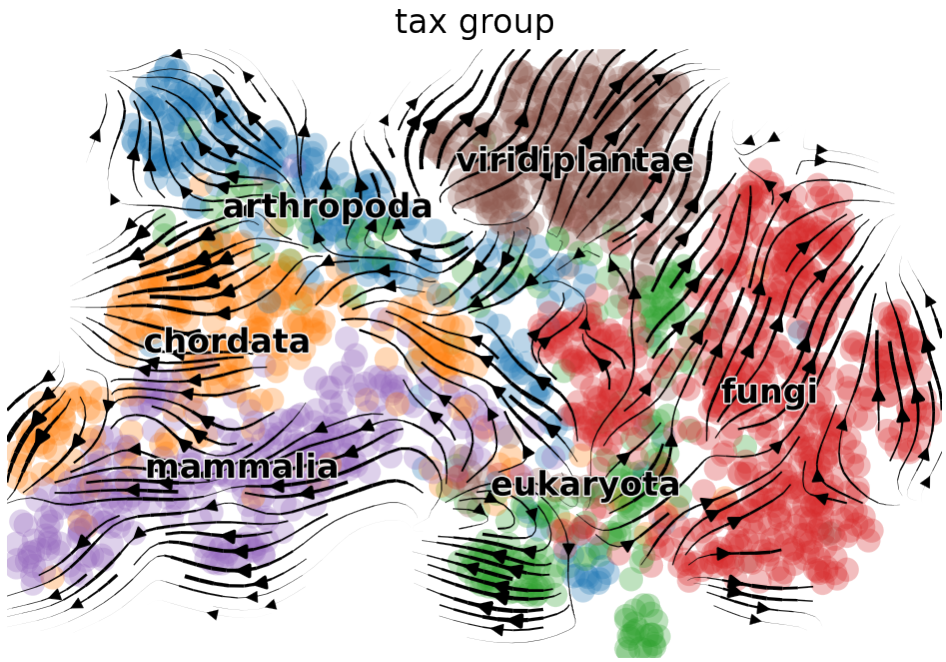
Now we can compute the velocity scores for each edge in the sequence landscape. Downloading the model can take up to 25 minutes. Computing the likelihoods **with GPU acceleration** takes around 2 minutes and computing the velocity scores also takes around 2 minutes.

[ ]:
```
evo.tl.velocity_graph(adata)
```

Now, we can project the velocities into two-dimensional UMAP space and then visualize the velocities as a streamplot:

[30]:
```
evo.tl.velocity_embedding(adata, basis='umap', scale=1.)
evo.pl.velocity_embedding_stream(
    adata, basis='umap', min_mass=1., smooth=1., linewidth=1.,
    color='tax_group',
)
```

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



CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX