Schema

Release 0.1.0

Sep 17, 2022

Contents

1 Overview 3

2 Installation 7

3 API 9

4 Data Integration Examples 15

5 Visualization Examples 27

6 Datasets 31

7 References 33

8 Indices and tables 35

Python Module Index 37

Index 39

Schema is a Python library for the synthesis and integration of heterogeneous single-cell modalities. **It is designed for the case where the modalities have all been assayed for the same cells simultaneously.** Here are some of the analyses that you can do with Schema:

- infer cell types jointly across modalities.
- perform spatial transcriptomic analyses to identify differntially-expressed genes in cells that display a specific spatial characteristic.
- create informative t-SNE & UMAP visualizations of multimodal data by infusing information from other modalities into scRNA-seq data.

Schema offers support for the incorporation of more than two modalities and can also simultaneously handle batch effects and metadata (e.g., cell age).

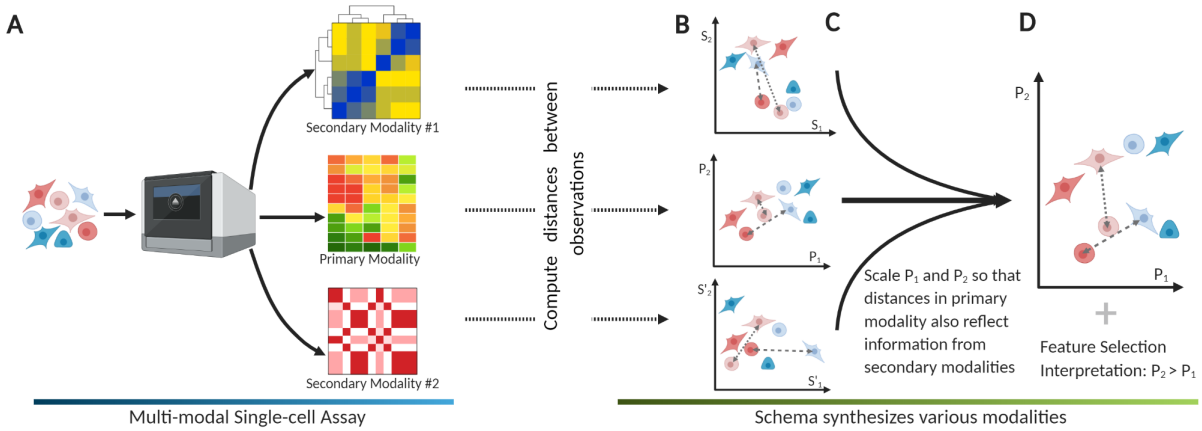
Schema is based on a metric learning approach and formulates the modality-synthesis problem as a quadratic programming problem. Its Python-based implementation can efficiently process large datasets without the need of a GPU.

Read the documentation. We encourage you to report issues at our Github page ; you can also create pull reports there to contribute your enhancements. If Schema is useful in your research, please consider citing our papers: Genome Biology (2021), with preprint in bioRxiv (2019).

CHAPTER 1

Overview

Schema is a general algorithm for integrating heterogeneous data modalities. While it has been specially designed for multi-modal single-cell biological datasets, it should work in other multi-modal contexts too.



Schema is designed for single-cell assays where multiple modalities have been *simultaneously* measured for each cell. For example, this could be simultaneously-asayed ("paired") scRNA-seq and scATAC-seq data, or a spatial-transcriptomics dataset (e.g. 10x Visium, Slideseq or STARmap). Schema can also be used with just a scRNA-seq dataset where some per-cell metadata is available (e.g., cell age, donor information, batch ID etc.). With this data, Schema can help perform analyses like:

- Characterize cells that look similar transcriptionally but differ epigenetically.
- Improve cell-type inference by combining RNA-seq and ATAC-seq data.
- In spatially-resolved single-cell data, identify differentially expressed genes (DEGs) specific to a spatial pattern.
- **Improved visualizations**: tune t-SNE or UMAP plots to more clearly arrange cells along a desired manifold.
- Simultaneously account for batch effects while also integrating other modalities.

1.1 Intuition

To integrate multi-modal data, Schema takes a metric learning approach. Each modality is interepreted as a multi-dimensional space, with observations mapped to points in it (**B** in figure above). We associate a distance metric with each modality: the metric reflects what it means for cells to be similar under that modality. For example, Euclidean distances between L2-normalized expression vectors are a proxy for coexpression. Across the three graphs in the figure (**B**), the dashed and dotted lines indicate distances between the same pairs of observations.

Schema learns a new distance metric between points, informed jointly by all the modalities. In Schema, we start by designating one high-confidence modality as the *primary* (i.e., reference) and the remaining modalities as *secondary*—we've found scRNA-seq to typically be a good choice for the primary modality. Schema transforms the primary-modality space by scaling each of its dimensions so that the distances in the transformed space have a higher (or lower, if desired!) correlation with corresponding distances in the secondary modalities (**C,D** in the figure above). You can choose any distance metric for the secondary modalities, though the primary modality's metric needs to be Euclidean. The primary modality can be pre-transformed by a PCA or NMF transformation so that the scaling occurs in this latter space; this can often be more powerful because the major directions of variance are now axis-aligned and hence can be scaled independently.

1.2 Advantages

In generating a shared-space representation, Schema is similar to statistical approaches like CCA (canonical correlation analysis) and deep-learning methods like autoencoders (which map multiple representations into a shared latent space). Each of these approaches offers a different set of trade-offs. Schema, for instance, requires the output space to be a linear transformation of the primary modality. Doing so allows it to offer the following advantages:

- **Interpretability**: Schema identifies which features of the primary modality were important in maximizing its agreement with the secondary modalities. If the features corresponded to genes (or principal components), this can directly be interpreted in terms of gene importances.
- **Regularization**: single-cell data can be sparse and noisy. As we discuss in our paper, unconstrained approaches like CCA and autoencoders seek to maximize the alignment between modalities without any other considerations. In doing so, they can pick up on artifacts rather than true biology. A key feature of Schema is its regularization: if enforces a limit on the distortion of the primary modality, making sure that the final result remains biologically informative.
- **Speed and flexibility**: Schema is a based on a fast quadratic programming approach that allows for substantial flexibility in the number of secondary modalities supported and their relative weights. Also, arbitrary distance metrics (i.e., kernels) are supported for the secondary modalities.

1.3 Quick Start

Install via pip

```
pip install schema_learn
```

Example: correlate gene expression with developmental stage. We demonstrate use with Anndata objects here.

```
import schema
adata = schema.datasets.fly_brain()  # adata has scRNA-seq data & cell age

sqp = schema.SchemaQP( min_desired_corr=0.99, # require 99% agreement with original␣
↪scRNA-seq distances
```

(continues on next page)

(continued from previous page)

```
                        params= {'decomposition_model': 'nmf', 'num_top_components':␣
↪20} )

#correlate the gene expression with the 'age' parameter
mod_X = sqp.fit_transform( adata.X, # primary modality
                           [ adata.obs['age'] ], # list of secondary modalities
                           [ 'numeric' ] )  # datatypes of secondary modalities

gene_wts = sqp.feature_weights() # get a ranking of gene wts important to the␣
↪alignment
```

1.4 Paper & Code

Schema is described in the paper *Schema: metric learning enables interpretable synthesis of heterogeneous single-cell modalities* (http://doi.org/10.1101/834549)

Source code available at: https://github.com/rs239/schema

CHAPTER 2

Installation

We recommend Python v3.6 or higher.

2.1 PyPI, Virtualenv, or Anaconda

You can use `pip` (or `pip3`):

```
pip install schema_learn
```

2.2 Docker

Schema has been designed to be compatible with the popular and excellent single-cell Python package, Scanpy. We recommend installing the Docker image recommended by Scanpy maintainers and then using `pip`, as described above, to install Schema in it.

CHAPTER 3

API

class schema.**SchemaQP**(*min_desired_corr=0.99*, *mode='affine'*, *params={}*)

Bases: object

Schema is a tool for integrating simultaneously-assayed data modalities

The SchemaQP class provides a sklearn type fit+transform API for constrained affine transformations of input datasets such that the transformed data is in agreement with all the input datasets.

Parameters

- **min_desired_corr** (*float in [0,1)*) – This parameter controls the severity of the primary modality's transformation, specifying the minimum required correlation between distances in the original space and those in the transformed space. It thus controls the trade-off between deviating further away from the primary modality's original representation and achieving greater agreement with the secondary modalities. Values close to one result in lower distortion of the primary modality while those close to zero enable transformations offering greater agreement between the modalities.

  RECOMMENDED VALUES: In typical single-cell use cases, high values (> 0.80) will probably work best. With these, the distortion will be low, but still be enough for Schema to extract relevant information from the secondary modalities. Furthermore, the feature weights computed by Schema should still be quite infromative.

  The default value of 0.99 is a safe choice to start with; it poses low risk of deviating too far from the primary modality.

  Later, you can experiment with a range of values (e.g., 0.95 0.90, 0.80), or use feature-weights aggregated across an ensemble of choices. Alternatively, you can use cross-validation to identify the best setting

- **mode** (*string*) – Whether to perform a general affine transformation or just a scaling transformation

  – *affine* first does a mapping to PCA or NMF space (you can specify num_top_components via the *params* argument). Schema does a scaling transform in the mapped space and then converts everything back to the regular space. The final result is thus an affine transformation in the regular space.

– *scale* does not do a PCA or NMF mapping, and directly applies the scaling transformation. **Note**: This can be slow if the primary modality's dimensionality is over 100.

RECOMMENDED VALUES: *affine* is the default. You may need *scale* only in certain cases:

– You have a limited number of features on which you directly want Schema to compute feature-weights.

– You want to do a change-of-basis transform other PCA or NMF. If so, you will need to do that yourself and then call SchemaQP with the transformed primary dataset with mode='scale'.

- **params** (*dict*) – Dictionary of key-value pairs, specifying additional configuration parameters. Here are the important ones:

  – *decomposition_model*: "pca" or "nmf" (default=pca)

  – *num_top_components*: (default=50) number of PCA (or NMF) components to use when mode=="affine". We recommend this setting be <= 100. Schema's runtime is quadratic in this number.

  You can ignore the rest on your first pass; the default values are pretty reasonable:

  – *dist_npairs*: (default=2000000). How many pt-pairs to use for computing pairwise distances. value=None means compute exhaustively over all n*(n-1)/2 pt-pairs. Not recommended for n>5000. Otherwise, the given number of pt-pairs is sampled randomly. The sampling is done in a way in which each point will be represented roughly equally.

  – *scale_mode_uses_standard_scaler*: 1 or 0 (default=0), apply the standard scaler in the scaling mode

  – *do_whiten*: 1 or 0 (default=1). When mode=="affine", should the change-of-basis loadings be made 1-variance?

Returns A SchemaQP object on which you can call fit(...), transform(...) or fit_transform(....).

explore_param_mincorr(*d, secondary_data_val_list, secondary_data_type_list, secondary_data_wt_list=None, min_desired_corr_values=[0.999, 0.99, 0.95, 0.9, 0.8, 0.5, 0.2, 0], addl_fit_kwargs={}, addl_feature_weights_kwargs={}*)

Helper function to explore multiple choices of the min_desired_corr param.

For a range of min_desired_corr parameter values, it performs a *fit*, gets the *feature_weights*, and also the achieved distance correlation between the transformed data and the primary/secondary modalities. While this method is simply a convenience wrapper around other public methods, it is nonetheless useful for exploring the best choice of min_desired_corr for your application. For example, if you're doing batch correction and hence set a secondary modality's wt to be negative, you want the distance correlation of batch information and transformed data to go to zero, not beyond that into negative correlation territory. This function can help you identify an appropriate min_desired_corr value.

The required arguments are the same as those for a call to *fit* (which this method calls, under the hood). The default list of possible values for min_desired_corr is a good place to start.

Parameters

- **d** (Numpy 2-d *array* or Pandas *dataframe*) – Same as in *fit*: the primary dataset (e.g. scanpy/anndata's .X).
- **secondary_data_val_list** (list of 1-d or 2-d Numpy arrays or Pandas series, each with same number of rows as *d*) – Same as in *fit*: the secondary datasets you want to align the primary data towards.

- **secondary_data_type_list** (*list of strings*) – Same as in *fit*: the datatypes of the secondary modalities.
- **secondary_data_wt_list** (list of floats, **optional**) – Same as in *fit*: user-specified wts for each secondary dataset (default= list of 1's)
- **min_desired_corr_values** (*list of floats, each value v being 0 <= v < 1*) – list of min_desired_corr values to explore. The default is [0.999, 0.99, 0.95, 0.9, 0.8, 0.5, 0.2, 0]
- **addl_fit_kwargs** (*dict*) – additional named arguments passed on to fit(...)
- **addl_feature_weights_kwargs** (*dict*) – named arguments passed on to feature_weights(...)

Returns

a tuple with 4 entries. In the first 3 below, each row of the dataframe corresponds to a min_desired_corr value

a) Dataframe of starting and ending distance correlations (see *get_start_end_dist_correlations* for details)

b) Dataframe of feature weights, produced by a call to *feature_weights*

c) Dataframe of QP solution wts. Same as feature weights if mode='scale', otherwise this corresponds to the QP-computed wts in the PCA/NMF space

d) Dictionary of SchemaQP objects, keyed by the min_desired_corr parameter. You can use them for *transform* calls.

feature_weights(*affine_map_style='top-k-loading'*, *k=1*)

Return the feature weights computed by Schema

If SchemaQP was initialized with *mode=scale*, the weights returned are directly the weights from the quadratic programming (QP), with a weight > 1 indicating the feature was up-weighted. The *affine_map_style* argument is ignored.

However, if *mode=affine* was used, the QP-computed weights correspond to columns of the PCA or NMF decomposition. In that case, this functions maps them back to the primary dataset's features. This can be done in three different ways, as specified by the *affine_map_style* parameter.

You can build your own mapping from PCA/NMF weights to primary-modality feature weights. The instance's *_wts* member is the numpy array that contains QP-computed weights, and *_decomp_mdl* is the sklearn-computed NMF/PCA decomposition. You can also look at the source code of this function to get a sense of how to use them.

Parameters

- **affine_map_style** (*string, one of 'softmax-avg' or 'top-k-rank' or 'top-k-loading', default='top-k-loading'*) – Governs how QP-computed weights for PCA/NMF columns are mapped back to primary-modality features (typically, genes from a scRNA-seq dataset).

  Default is 'top-k-loading', which considers only the top-k PCA/NMF columns by QP-computed weight and computes the average loading of a gene across these. The second argument specifies k (default=1)

  Another choice is 'softmax-avg', which computes gene weights by a softmax-type summation of loadings across the PCA/NMF columns, with each column's weight proportional to exp(QP wt), and only columns with QP weight > 1 being considered. *k is ignored here*.

Yet another choice is 'top-k-rank', which considers only the top-k PCA/NMF columns by QP-computed weight and computes the average rank of a gene across their loadings. The second argument specifies k (default=1)

In all approaches, PCA loadings are first converted to absolute value, since PCA columns are unique up to a sign.

- **k** (*int, >= 0*) – The number of PCA/NMF columns to average over, when affine_map_style = top-k-loading or top-k-rank.

returns : a vector of floats, the same size as the primary dataset's dimensionality

fit(*d*, *secondary_data_val_list*, *secondary_data_type_list*, *secondary_data_wt_list=None*, *secondary_data_dist_kernels=None*, *d0=None*, *d0_dist_transform=None*)
Compute the optimal Schema transformation, first performing a change-of-basis transformation if required.

Given the primary dataset *d* and a list of secondary datasets, fit a linear transformation (*d_new*) such that the correlation between squared pairwise distances in *d_new* and those in secondary datasets is maximized while the correlation between the original *d* and the transformed *d_new* remains above min_desired_corr.

The first three arguments are required, the next is useful, and the rest should be rarely used.

Parameters

- **d** (Numpy 2-d *array* or Pandas *dataframe*) – The primary dataset (e.g. scanpy/anndata's .X).

  The rows are observations (e.g., cells) and the cols are variables (e.g., gene expression). The default distance measure computed is L2: sum((point1-point2)**2). Also see *d0_dist_transform*.

- **secondary_data_val_list** (list of 1-d or 2-d Numpy arrays or Pandas series, each with same number of rows as *d*) – The secondary datasets you want to align the primary data towards.

  Columns in Anndata .obs or .obsm variables work well.

- **secondary_data_type_list** (*list of strings*) – The datatypes of the secondary modalities.

  Each element of the list can be one of *numeric, feature_vector, categorical, feature_vector_categorical*. The list's length should match the length of secondary_data_val_list

  - *numeric*: one floating-pt value for each observation. The default distance measure is Euclidean: (point1-point2)^2
  - *feature_vector*: a k-dimensional vector for each observation. The default distance measure is Euclidean: sum_{i}((point1[i]-point2[i])^2)
  - *categorical*: a label for each observation. The default distance measure checks for equality: 1*(val1!=val2)
  - *feature_vector_categorical*: a vector of labels for each observation. Each column can take on categorical values, so the distance between two points is sum_{i}(point1[i]==point2[i])

- **secondary_data_wt_list** (list of floats, **optional**) – User-specified wts for each secondary dataset (default= list of 1's)

  If specified, the list's length should match the length of secondary_data_val_list. When multiple secondary modalities are specified, this parameter allows you to control their relative weight in seeking an agreement with the primary.

Note: you can try to get a mapping that *disagrees* with a dataset_info instead of *agreeing*. To do so, pass in a negative number (e.g., -1) here. This works even if you have just one secondary dataset

- **secondary_data_dist_kernels** (list of functions, **optional**) – The transformations to apply on secondary dataset's L2 distances before using them for correlations.

  If specified, the length of the list should match that of *secondary_data_val_list*. Each function should take a non-negative float and return a non-negative float.

  Handle with care: Most likely, you don't need this parameter.

- **d0** (A 1-d or 2-d Numpy array, **optional**) – An alternative representation of the primary dataset.

  This is useful if you want to provide the primary dataset in two forms: one for transforming and another one for computing pairwise distances to use in the QP constraint; if so, *d* is used for the former, while *d0* is used for the latter. If specified, it should have the same number of rows as *d*.

  Handle with care: Most likely, you don't need this parameter.

- **d0_dist_transform** (float -> float function, **optional**) – The transformation to apply on d or d0's L2 distances before using them for correlations.

  This function should take a non-negative float as input and return a non-negative float.

  Handle with care: Most likely, you don't need this parameter.

Returns None

fit_transform(*d*, *secondary_data_val_list*, *secondary_data_type_list*, *secondary_data_wt_list=None*, *secondary_data_dist_kernels=None*, *d0=None*, *d0_dist_transform=None*)

Calls fit(..) with exactly the arguments given; then calls transform(d). See documentation for fit(....) and transform(...) respectively.

get_start_end_dist_correlations()

Return the starting and ending distance correlations between primary and secondary modalities

Note: the distance correlations reported out (even between the primary and secondary modalities) may vary from run to run, since the underlying algorithm samples a set of point pairs to compute its estimates.

Returns

a tuple with 3 entries:

a) distance correlation between primary and transformed space. This should always be >= min_desired_corr but it can be substantially greater than min_desired_corr if the optimal solution requires that.

b) vector of distance correlations between primary and secondary modalities, and

c) vector of distance correlations between transformed dataset and secondary modalities

reset_maxwt_param(*w_max_to_avg*)

Reset the w_max_to_avg param

Parameters **w_max_to_avg** (*float*) – The upper-bound on the ratio of Schema weights (w's) largest element to w's avg element. Making it large will allow This parameter controls the 'deviation' in feature weights and make it large will allow for more severe transformations.

Handle with care: We recommend keeping this parameter at its default value (1000); that keeps this constraint very loose and ensures that min_desired_corr remains the binding constraint. Later, as you get a better sense for the right min_desired_corr values for your data, you can experiment with this too. To really constrain this, set it in the (1-5] range, depending on how many features you have.

reset_mincorr_param(*min_desired_corr*)
Reset the min_desired_corr.

Useful when you want to iterate over multiple choices of this parameter but want to re-use the computed PCA or NMF change-of-basis transform.

Parameters **min_desired_corr** (*float in [0,1)*) – The new value of minimum required correlation between original and transformed distances

transform(*d*)
Given a dataset *d*, apply the fitted transform to it

Parameters **d** (*Numpy 2-d array*) – The primary modality data on which to apply the transformation.

d must have with same number of columns as in *fit(...)*. The rows are observations (e.g., cells) and the cols are variables (e.g., gene expression).

Returns a 2-d Numpy array with the same shape as *d*

CHAPTER 4

Data Integration Examples

4.1 API-usage Examples

Note: The code snippets below show how Schema could be used for hypothetical datasets and illustrates the API usage. In the next sections (*Paired RNA-seq and ATAC-seq*, *Paired-Tag*) and in Visualization, we describe worked examples where we also provide the dataset to try things on. We are working to add more datasets.

Example Correlate gene expression 1) positively with ATAC-Seq data and 2) negatively with Batch information.

```
atac_50d = sklearn.decomposition.TruncatedSVD(50).fit_transform( atac_cnts_sp_matrix)

sqp = SchemaQP(min_corr=0.9)

# df is a pd.DataFrame, srs is a pd.Series, -1 means try to disagree
mod_X = sqp.fit_transform( df_gene_exp, # gene expression dataframe: rows=cells,␣
↪cols=genes
                           [ atac_50d, batch_id],  # batch_info can be a pd.Series or␣
↪np.array. rows=cells
                           [ 'feature_vector', 'categorical'],
                           [ 1, -1]) # maximize combination of (agreement with ATAC-
↪seq + disagreement with batch_id)

gene_wts = sqp.feature_weights() # get gene importances
```

Example Correlate gene expression with three secondary modalities.

```
sqp = SchemaQP(min_corr = 0.9) # lower than the default, allowing greater distortion␣
↪of the primary modality
sqp.fit( adata.X,
         [ adata.obs['col1'], adata.obs['col2'], adata.obsm['Matrix1'] ],
         [ "categorical", "numeric", "feature_vector"]) # data types of the three␣
↪modalities
mod_X = sqp.transform( adata.X) # transform
gene_wts = sqp.feature_weights() # get gene importances
```

4.2 Paired RNA-seq and ATAC-seq

Here, we integrate simultaneously assayed RNA- and ATAC-seq data from Cao et al.'s sci-CAR study of mouse kidney cells. Specifically, we'll try to do better cell-type inference by considering both RNA-seq and ATAC-seq data simultaneously. The original study has ground-truth labels for most of the cell types, allowing us to benchmark automatically-computed clusters (generated by Leiden clustering here). As we'll show, a key challenge here is that the ATAC-seq data is very sparse and noisy. Naively incorporating it with RNA-seq can actually be counter-productive—the joint clustering from a naive approach can actually have a *lower* overlap with the ground truth labels than if we were to just use RNA-seq-based clustering.

Note: This example involves generating Leiden clusters; you will need to install the *igraph* and *leidenalg* Python packages if you want to use them:

```
pip install igraph
pip install leidenalg
```

Let's start by getting the data. We have preprocessed the original dataset, done some basic cleanup, and put it into an AnnData object that you can download. Please remember to also cite the original study if you use this dataset.

```
import schema
adata = schema.datasets.scicar_mouse_kidney()
print(adata.shape, adata.uns['atac.X'].shape)
print(adata.uns.keys())
```

As you see, we have stored the ATAC data (as a sparse numpy matrix) in the .uns slots of the anndata object. Also look at the *adata.obs* dataframe which has t-SNE coordinates, ground-truth cell type names (as assigned by Cao et al.) and cluster colors etc. You'll notice that some cells don't have ground truth assignments. When evaluating, we'll skip those.

To use the ATAC-seq data, we reduce its dimensionality to 50. Instead of PCA, we apply *TruncatedSVD* since the ATAC counts matrix is sparse.

```
svd2 = sklearn.decomposition.TruncatedSVD(n_components= 50, random_state = 17)
H2 = svd2.fit_transform(adata.uns["atac.X"])
```

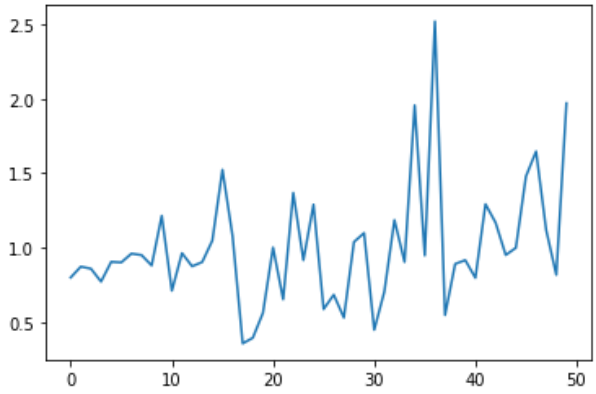
Next, we run Schema. We choose RNA-seq as the primary modality because 1) it has lower noise than ATAC-seq, and 2) we want to investigate which of its features (i.e., genes) are important during the integration. We will first perform a NMF transformation on the RNA-seq data. For the secondary modality, we'll use the dimensionality-reduced ATAC-seq. We require a positive correlation between the two (*secondary_data_wt_list = [1]* below). **Importantly, we force Schema to generate a low-distortation transformation** : the correlation of distances between original RNA-seq space and the transformed space, *min_desired_corr* is required to be >99%. This low-distortion capability of Schema is crucial here, as we'll demonstrate.

In the *params* settings below, the number of randomly sampled point-pairs has been bumped up to 5M (from default=2M). It helps with the accuracy and doesn't cost too much computationally. We also turned off *do_whiten* (default=1, i.e., true). When *do_whiten=1*, Schema first rescales the PCA/NMF transformation so that each axis has unit variance; typically, doing so is "nice" from a theoretical/statistical perspective. But it can interfere with downstream analyses (e.g., Leiden clustering here).

```
sqp99 = schema.SchemaQP(0.99, mode='affine', params= {"decomposition_model":"nmf",
                                                      "num_top_components":50,
                                                      "do_whiten": 0,
                                                      "dist_npairs": 5000000})
dz99 = sqp99.fit_transform(adata.X, [H2], ['feature_vector'], [1])
```

Let's look at the feature weights. Since we ran the code in 'affine' mode, the raw weights from the quadratic program will correspond to the 50 NMF factors. Three of these factors seem to stand out; most other weights are quite low.

```
plt.plot(sqp99._wts)
```



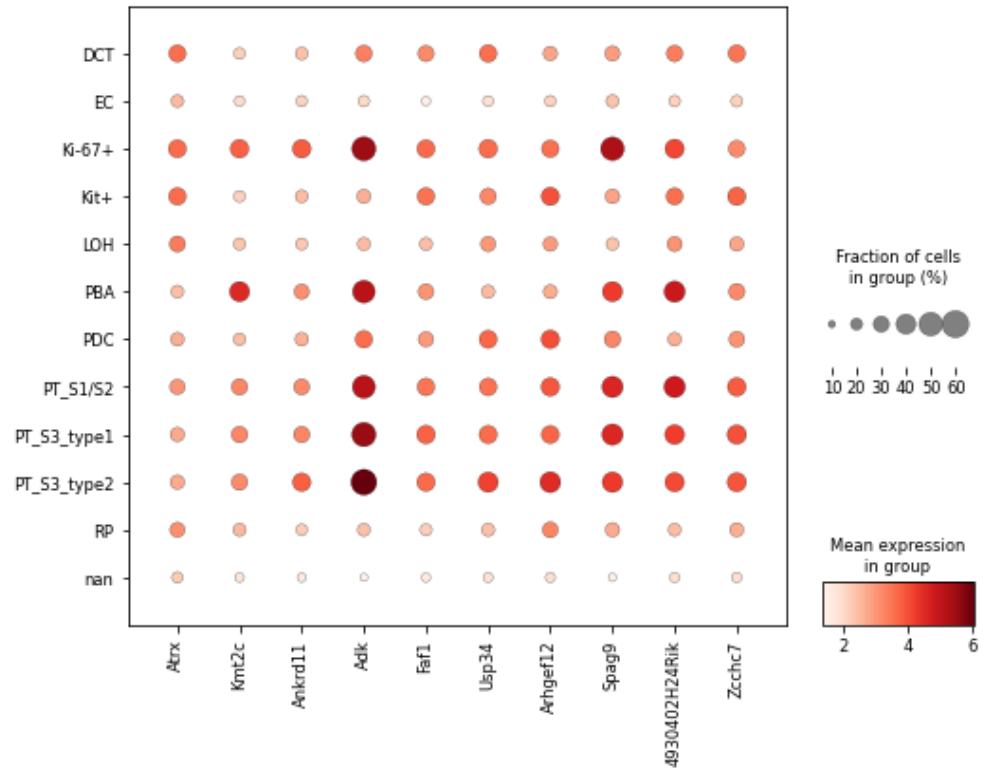
Schema offers a helper function to convert these NMF (or PCA) feature weights to gene weights. The function offers a few ways of doing so, but the default is to simply average the loadings across the top-k factors:

```
v99 = sqp99.feature_weights("top-k-loading", 3)
```

Let's do a dotplot to visualize how the expression of these genes varies by cell name. We plot the top 10 genes by importance here.

```
dfv99 = pd.DataFrame({"gene": adata.var_names, "v":v99}).sort_values("v",␣
↪ascending=False).reset_index(drop=True)
sc.pl.dotplot(adata, dfv99.gene.head(10).tolist(),'cell_name_short', figsize=(8,6))
```

As you'll notice, theese gene seem to be differentially expressed in PT cells, PBA and Ki-67+ cells. Essentially, these are cell types where ATAC-seq data was most informative. As we'll see shortly, it is preciely in these cells where Schema is able to offer the biggest improvement.



For a comparison later, let's also do a Schema run without a strong distortion control. Below, we set the *min_desired_corr* parameter to 0.10 (i.e., 10%). Thus, the ATAC-seq data will get to influence the transformation a lot more.

```
sqp10 = schema.SchemaQP(0.10, mode='affine', params= {"decomposition_model":"nmf",
                                                      "num_top_components":50,
                                                      "do_whiten": 0,
                                                      "dist_npairs": 5000000})
dz10 = sqp10.fit_transform(adata.X, [H2], ['feature_vector'], [1])
```

Finally, let's do Leiden clustering of the RNA-seq, ATAC-seq, and the two Schema runs. We'll compare the cluster assignments to the ground truth cell labels. Intuitively, by combining RNA-seq and ATAC-seq, one should be able to get a more biologically accurate clustering. We visually evaluate the clusterings below; in the paper, we've supplemented this with more quantitative estimates.

```
import schema.utils
fcluster = schema.utils.get_leiden_clustering #feel free to try your own clustering␣
↪algo

ld_cluster_rna = fcluster(sqp99._decomp_mdl.transform(adata.X.todense()))
ld_cluster_atac = fcluster(H2)
ld_cluster_sqp99 = fcluster(dz99)
ld_cluster_sqp10 = fcluster(dz10)
```

```
x = adata.obs.tsne_1
y = adata.obs.tsne_2
idx = adata.obs.rgb.apply(lambda s: isinstance(s,str) and '#' in s).values.tolist()
↪#skip nan cells

fig, axs = plt.subplots(3,2, figsize=(10,15))
```

(continues on next page)

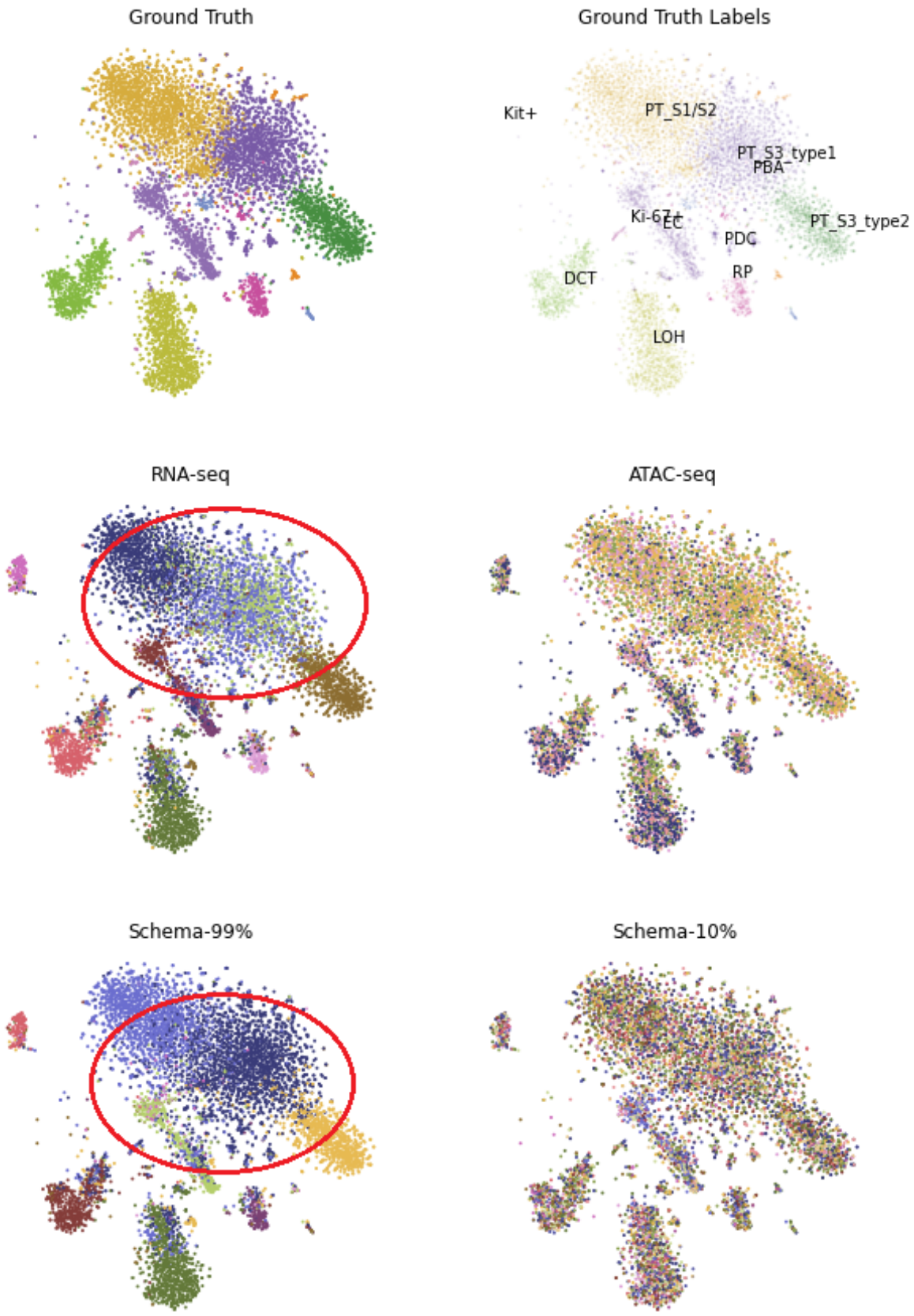
(continued from previous page)

```
axs[0][0].scatter(x[idx], y[idx], c=adata.obs.rgb.values[idx], s=1)
axs[0][0].set_title('Ground Truth')
axs[0][1].scatter(x[idx], y[idx], c=adata.obs.rgb.values[idx], s=1, alpha=0.1)
axs[0][1].set_title('Ground Truth Labels')
for c in np.unique(adata.obs.cell_name_short[idx]):
    if c=='nan': continue
    cx,cy = x[adata.obs.cell_name_short==c].mean(), y[adata.obs.cell_name_short==c].
↪mean()
    axs[0][1].text(cx,cy,c,fontsize=10)
axs[1][0].scatter(x[idx], y[idx], c=ld_cluster_rna[idx], cmap='tab20b', s=1)
axs[1][0].set_title('RNA-seq')
axs[1][1].scatter(x[idx], y[idx], c=ld_cluster_atac[idx], cmap='tab20b', s=1)
axs[1][1].set_title('ATAC-seq')
axs[2][0].scatter(x[idx], y[idx], c=ld_cluster_sqp99[idx], cmap='tab20b', s=1)
axs[2][0].set_title('Schema-99%')
axs[2][1].scatter(x[idx], y[idx], c=ld_cluster_sqp10[idx], cmap='tab20b', s=1)
axs[2][1].set_title('Schema-10%')

for ax in np.ravel(axs): ax.axis('off')
```

Below, we show the figures in a 3x2 panel of t-SNE plots. In the first row, the left panel shows the cells colored by ground-truth cell types; the right panel is basically the same but lists the cell types explicitly. The next row shows cells colored by RNA- or ATAC-only clustering. Notice how noisy the ATAC-only clustering is! This is not a bug in our analysis– less than 0.3% of ATAC count matrix entries are non-zero and the sparsity of the ATAC data makes it difficult to produce high-quality cell type estimates.

The third row shows cells colored by Schema-based clustering at 99% (left) and 10% (right) *min_desired_corr* thresholds. With Schema at a low-distortion setting (i.e., *min_desired_corr = 99%*), notice that PT cells and Ki-67+ cells, circled in red, are getting more correctly classified now. This improvement of the Schema-implied clustering over the RNA-seq-only clustering can be quantified by measuring the overlap with ground truth cell grouping, as we do in the paper.

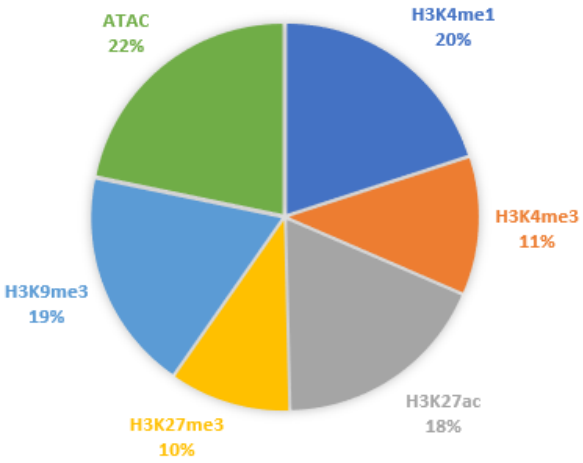
This is a key strength of Schema — even with a modality that is sparse and noisy (like ATAC-seq here), it can nonetheless extract something of value from the noisy modality because the constraint on distortion of the primary modality acts as a regularization. This is also why we recommend that your highest-confidence modality be set as the primary. Lastly as demonstration, if we relax the distortion constraint by setting *min_desired_corr = 10%*, you'll notice that the noise of ATAC-seq data does swamp out the RNA-seq signal. With an unconstrained approach (e.g., CCA or some deep learning approaches), this ends being a major challenge.


Ground Truth
Ground Truth Labels
Kit+
PT_S1/S2
PT_S3_type1
PBA
Ki-67+
EC
PDC
PT_S3_type2
DCT
RP
LOH
RNA-seq
ATAC-seq
Schema-99%
Schema-10%

4.3 Paired-Tag

Here we synthesize simultaneously assayed RNA-seq, ATAC-seq and histone-modification data at a single-cell resolution, from the Paired-Tag protocol described in Zhu et al.'s study of adult mouse frontal cortex and hippocampus (Nature Methods, 2021). This is a fascinating dataset with five different histone modifications assayed separately (3 repressors and 2 activators), in addition to RNA-seq and ATAC-seq. As in the original study, we consider each of the histone modifications as a separate modality, implying a hepta-modal assay!

Interestingly, though, the modalities are available only in pairwise combinations with RNA-seq: some cells were assayed for H3K4me1 & RNA-seq while another set of cells provided ATAC-seq & RNA-seq data, and so on. Here's the overall distribution of non-RNA-seq modalities across 64,849 cells.



This organization of data might be tricky to integrate with a method which expects *each* modality to be available for *all* cells and has difficulty accomodating partial coverage of some modalities. Of course, you could always fall back to an integrative approach that treats each modality's cell population as independent, but then you miss out on the simultaneously-multimodal aspect of this data.

With Schema, you can have your cake and eat it too! We do 6 two-way integrations (RNA-seq as the primary modality against each of the other modalities) using the subsets of cells available in each case. Schema's interpretable and linear framework makes it easy to combine these. Once Schema computes the optimal transformation of RNA-seq that aligns it with, say, ATAC-seq, we apply that transformation to the entire RNA-seq dataset, including cells that do *not* have ATAC-seq data.

Such full-dataset extensions of the pairwise syntheses can then be stacked together. Doing Leiden clustering on the result would enable us to infer cell types by integrating information from all modalities. As we will show below, Schema's synthesis helps improve the quality of cell type inference over what you could get just from RNA-seq. Similarly for feature selection, Schema's computed feature weights for each two-way synthesis can be averaged to get the genes important to the overall synthesis. In a completely automated fashion and without any knowledge of tissue's source or biology, we'll find that the genes Schema identifies as important turn out to be very relevant to neuronal function and disease. Ready for more?

First, you will need the data. The original is available on GEO (GSE152020) but the individual modalities are huge (e.g., the ATAC-seq peak-counts are in a 14,095 x 2,443,832 sparse matrix!). This is not unusual— epigenetic modalites are typically very sparse (we discuss why this matters in *Paired RNA-seq and ATAC-seq*). As a preprocessing step, we performed singular value decompositions (SVD) of these modalities and also reduced the RNA-seq data to its 4,000 highly variable genes. An AnnData object with this preprocessing is available here (please remember to also cite the original study if you use this dataset) :

```
wget http://cb.csail.mit.edu/cb/schema/adata_dimreduced_paired-tag.pkl
```

Let's load it in:

```
import schema, pickle, anndata, sklearn.metrics
import scanpy as sc

# you may need to change the file location as appopriate to your setup
adata = pickle.load(open("adata_dimreduced_paired-tag.pkl", "rb"))

print (adata.shape,
       [(c, adata.uns['SVD_'+c].shape) for c in adata.uns['sec_modalities']])
```

As you see, we have stored the 50-dimensional SVDs of the secondary modalities in the `.uns` slots of the anndata object. Also look at the `adata.obs` dataframe which has UMAP coordinates, ground-truth cell type names (as assigned by Zhu et al.) etc.

We now do Schema runs for the 6 two-way modality combinations, with RNA-seq as the primary in each run. Each run will also store the transformation on the entire 64,849-cell RNA-seq dataset and also store the gene importances.

```
d_rna = adata.X.todense()

desc2transforms = {}
for desc in adata.uns['sec_modalities']:
    print(desc)

    # we mostly stick with the default settings, explicitly listed here for clarity
    sqp = schema.SchemaQP(0.99, mode='affine', params= {"decomposition_model": 'pca',
                                                        "num_top_components":50,
                                                        "do_whiten": 0, # this is␣
↪different from default
                                                        "dist_npairs": 5000000})

    # extract the relevant subset
    idx1 = adata.obs['rowidx'][adata.uns["SVD_"+desc].index]
    prim_d = d_rna[idx1,:]
    sec_d = adata.uns["SVD_"+desc].values
    print(len(idx1), prim_d.shape, sec_d.shape)

    sqp.fit(prim_d, [sec_d], ['feature_vector'], [1]) # fit on the idx1 subset...
    dz = sqp.transform(d_rna)  # ...then transform the full RNA-seq dataset

    desc2transforms[desc] = (sqp, dz, idx1, sqp.feature_weights(k=3))
```

Cell type inference:: In each of the 6 runs above, `dz` is a 64,849 x 50 matrix. We can horizontally stack these matrices for a 64,849 x 300 matrix that represents the transformation of RNA-seq data informed simultaneously by all 6 secondary modalities.

```
a6Xpca = np.hstack([dz for  _,dz,_,_ in desc2transforms.values()])
adata_schema = anndata.AnnData(X=a6Xpca, obs=adata.obs)
print (adata_schema.shape)
```

We then perform Leiden clustering on the original and transformed data, computing the overlap with expert marker-gene-based annotation by Zhu et al.

```
# original
sc.pp.pca(adata)
sc.pp.neighbors(adata)
sc.tl.leiden(adata)

# Schema-transformed
```

(continues on next page)

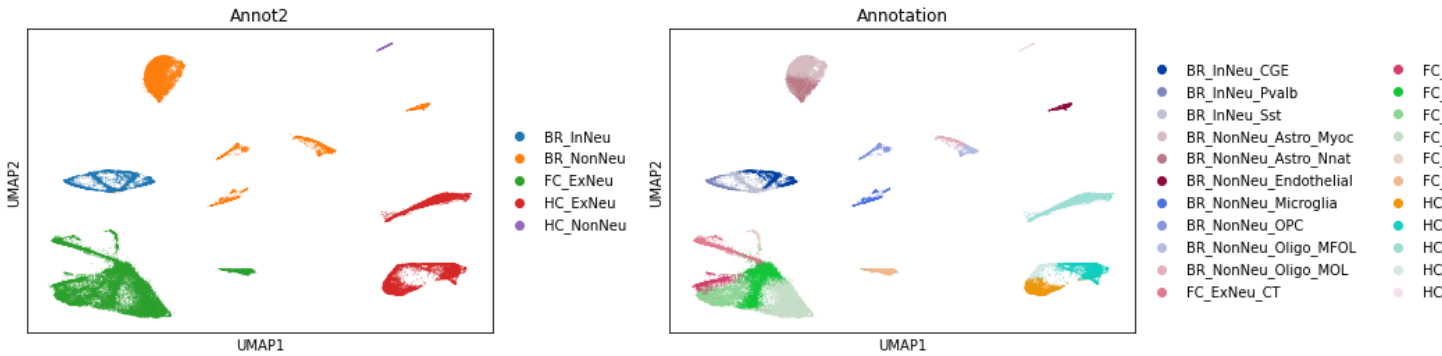
(continued from previous page)

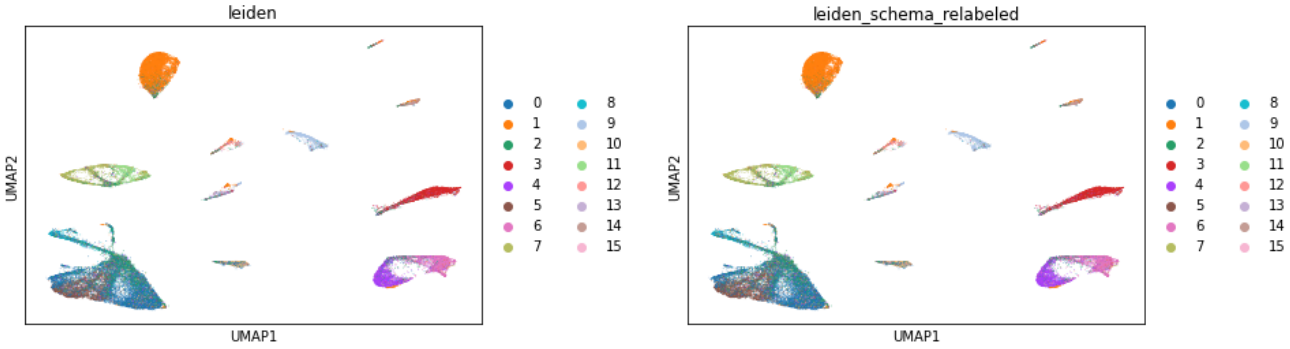
```
# since Schema had already done PCA before it transformed, let's stick with its raw␣
↪output
sc.pp.neighbors(adata_schema, use_rep='X')
sc.tl.leiden(adata_schema)

# we'll do plots etc. with the original AnnData object
adata.obs['leiden_schema'] = adata_schema.obs['leiden'].values

# compute overlap with manual cell type annotations
ari_orig  =  sklearn.metrics.adjusted_rand_score(adata.obs.Annotation, adata.obs.
↪leiden)
ari_schema=  sklearn.metrics.adjusted_rand_score(adata.obs.Annotation, adata.obs.
↪leiden_schema)

print ("ARI: Orig: {} With Schema: {}".format( ari_orig, ari_schema))
```

As you can see, the ARI with Schema improved from 0.437 (using only RNA-seq) to 0.446 (using all modalities). Single-cell epigenetic modalities are very sparse, making it difficult to distinguish signal from noise. However, Schema's constrained approach allows it to extract signal from these secondary modalities nonetheless, a task which has otherwise been challenging (see the related discussion in our paper or in *Paired RNA-seq and ATAC-seq*).

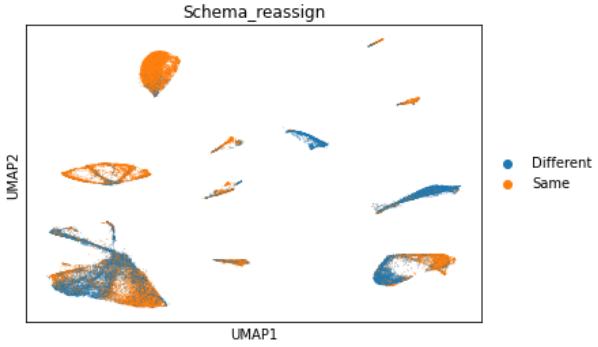
Before we plot these clusters, we'll relabel the Schema-based Leiden clusters to match the labeling of RNA-seq only Leiden clusters; this will make their color schemes consistent. You will need to install the Python package *munkres* (`pip install munkres`) for the related computation.

```
import munkres
list1 = adata.obs['leiden'].astype(int).tolist()
list2 = adata.obs['leiden_schema'].astype(int).tolist()

contmat = sklearn.metrics.cluster.contingency_matrix(list1, list2)
map21 = dict(munkres.Munkres().compute(contmat.max() - contmat))
adata.obs['leiden_schema_relabeled'] = [str(map21[a]) for a in list2]
adata.obs['Schema_reassign'] = [('Same' if (map21[a]==a) else 'Different') for a in␣
↪list2]

for c in ['Annotation','Annot2', 'leiden', 'leiden_schema_relabeled', 'Schema_reassign
↪']:
    sc.pl.umap(adata, color=c)
```



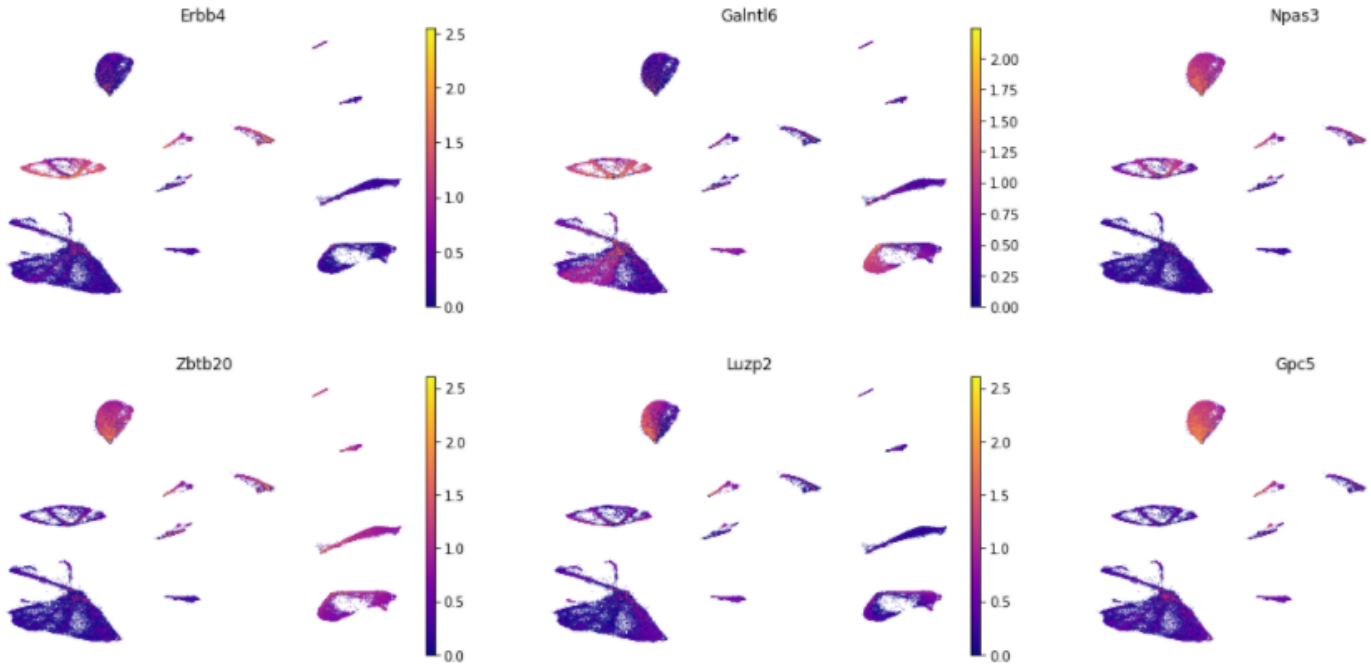


It's also interesting to identify cells where the cluster assignments changed after multi-modal synthesis. As you can see, it's only in certain cell types where the epigenetic data suggests a different clustering than the primary RNA-seq modality.



Gene set identification: The feature importances output by Schema here identify the genes whose expression variations best agree with epigenetic variations in these tissues. We first aggregate the feature importances across the 6 two-ways runs:

```
df_genes = pd.DataFrame({'gene': adata.var.symbol})
for desc, (_,_,_,wts) in desc2transforms.items():
    df_genes[desc] = wts
df_genes['avg_wt'] = df_genes.iloc[:,1:].mean(axis=1)
df_genes = df_genes.sort_values('avg_wt', ascending=False).reset_index(drop=True)

gene_list = df_genes.gene.values

sc.pl.umap(adata, color= gene_list[:6], gene_symbols='symbol', color_map='plasma',␣
↪frameon=False, ncols=3)
```



Many of the top genes identified by Schema (e.g., Erbb4, Npas3, Zbtb20, Luzp2) are known to be relevant to neuronal function or disease. Note that all of this fell out of the synthesis directly— we didn't do any differential expression analysis against an external background or provide the method some other indication that the data is from brain tissue.

We also did a GO enrichment analysis (via Gorilla) of the top 100 genes by Schema weight. Here are the significant hits (FDR q-val < 0.1). Again, most GO terms relate to neuronal development, activity, and communication:

Table 1: GO Enrichment of Top Schema-identified genes

GO:0007416	synapse assembly
GO:0050808	synapse organization
GO:0051960	regulation of nervous system development
GO:0007156	homophilic cell adhesion via plasma membrane adhesion molecules
GO:0032989	cellular component morphogenesis
GO:0007155	cell adhesion
GO:0051239	regulation of multicellular organismal process
GO:0022610	biological adhesion
GO:0099177	regulation of trans-synaptic signaling
GO:0050804	modulation of chemical synaptic transmission
GO:0007399	nervous system development
GO:0099536	synaptic signaling
GO:0006810	transport
GO:0042391	regulation of membrane potential
GO:0007610	behavior
GO:0098742	cell-cell adhesion via plasma-membrane adhesion molecules

CHAPTER 5

Visualization Examples

Popular tools like t-SNE and UMAP can produce intuitive and appealing visualizations. However, since they perform opaque non-linear transformations of the input data, it can be unclear how to "tweak" the visualization to accentuate a specific aspect of the input. Also, it can can sometimes be difficult to understand which features (e.g. genes) of the input were most important to getting the plot.

Schema can help with both of these issues. With scRNA-seq data as the primary modality, Schema can transform it by infusing additional information into it while preserving a high level of similarity with the original data. When t-SNE/UMAP are applied on the transformed data, we have found that the broad contours of the original plot are preserved while the new information is also reflected. Furthermore, the relative weight of the new data can be calibrated using the *min_desired_corr* parameter of Schema.

5.1 Ageing fly brain

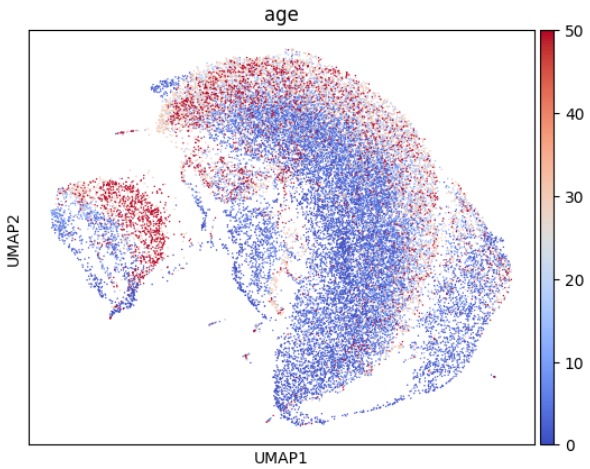
Here, we tweak the UMAP plot of Davie et al.'s ageing fly brain data to accentuate cell age.

First, let's get the data and do a regular UMAP plot.

```
import schema
import scanpy as sc
import anndata

def sc_umap_pipeline(bdata, fig_suffix):
    sc.pp.pca(bdata)
    sc.pp.neighbors(bdata, n_neighbors=15)
    sc.tl.umap(bdata)
    sc.pl.umap(bdata, color='age', color_map='coolwarm', save='_{}.png'.format(fig_
↪suffix) )
```

```
adata = schema.datasets.fly_brain()  # adata has scRNA-seq data & cell age
sc_umap_pipeline(adata, 'regular')
```

This should produce a plot like this, where cells are colored by age.



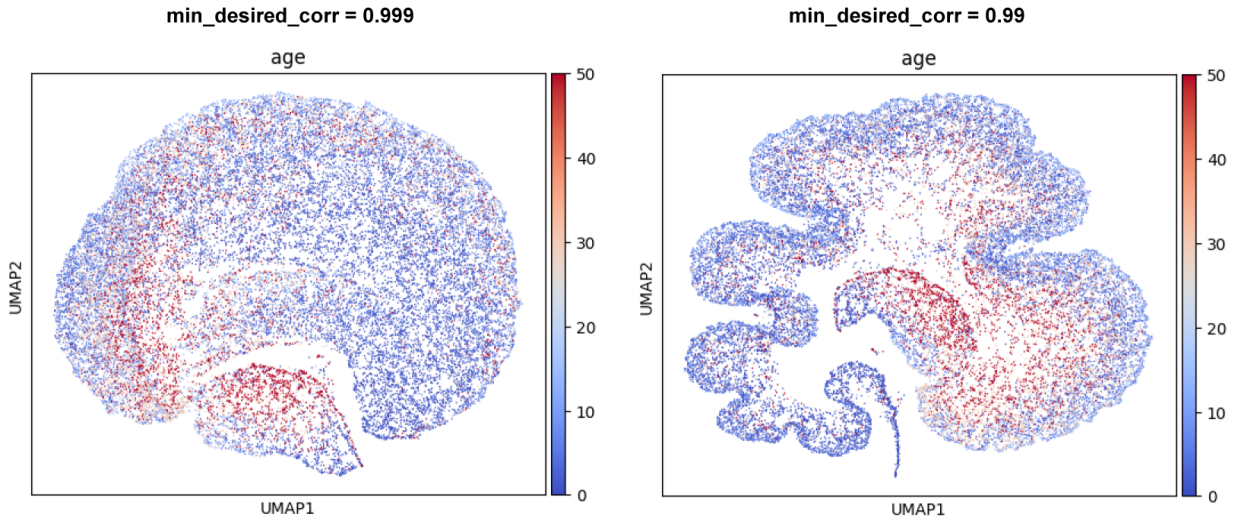
Next, we apply Schema to infuse cell age into the scRNA-seq data, while preserving a high level of correlation with the original scRNA-seq distances. We start by requiring a minimum 99.9% correlation with original scRNA-seq distances

```
sqp = schema.SchemaQP( min_desired_corr=0.999, # require 99.9% agreement with␣
↪original scRNA-seq distances
                       params= {'decomposition_model': 'nmf', 'num_top_components':␣
↪20} )

mod999_X = sqp.fit_transform( adata.X, [ adata.obs['age'] ], ['numeric'])  #␣
↪correlate gene expression with the age
sc_umap_pipeline( anndata.AnnData( mod999_X, obs=adata.obs), '0.999' )
```

We then loosen the *min_desired_corr* constraint a tiny bit, to 99%

```
sqp.reset_mincorr_param(0.99) # we can re-use the NMF transform (which takes more␣
↪time than the quadratic program)

mod990_X = sqp.fit_transform( adata.X, [ adata.obs['age'] ], ['numeric'])
sc_umap_pipeline( anndata.AnnData( mod990_X, obs=adata.obs), '0.990' )

diffexp_gene_wts = sqp.feature_weights() # get a ranking of genes important to the␣
↪alignment
```

These runs should produce a pair of plots like the ones shown below. Note how cell-age progressively stands out as a characteristic feature. We also encourage you to try out other choices of *min_desired_corr* (e.g., 0.90 or 0.7); these will show the effect of allowing greater distortions of the primary modality.



This example also illustrates Scehma's interpretability. The variable *diffexp_gene_wts* identifies the genes most important to aligning scRNA-seq with cell age. As we describe in our paper, these genes turn out to be differentially expressed between young cells and old cells.

CHAPTER 6

Datasets

6.1 Ageing *Drosophila* brain

This is sourced from Davie et al. (*Cell* 2018, GSE 107451) and contains scRNA-seq data from a collection of fly brain cells along with each cell's age (in days). It is a useful dataset for exploring a common scenario in multi-modal integration: scRNA-seq data aligned to a 1-dimensional secondary modality. Please see the example in Visualization where this dataset is used.

```
import schema
adata = schema.datasets.fly_brain()
```

6.2 Paired RNA-seq and ATAC-seq from mouse kidney cells

This is sourced from Cao et al. (*Science* 2018, GSE 117089) and contains paired RNA-seq and ATAC-seq data from a collection of mouse kidney cells. The AnnData object provided here has some additional processing done to remove very low count genes and peaks. This is a useful dataset for the case where one of the modalities is very sparse (here, ATAC-seq). Please see the example in Paired RNA-seq and ATAC-seq where this dataset is used.

```
import schema
adata = schema.datasets.scicar_mouse_kidney()
```

CHAPTER 7

References

Code: Github repo

Paper: If you use Schema, please consider citing *Schema: metric learning enables interpretable synthesis of heterogeneous single-cell modalities* (http://doi.org/10.1101/834549)

Project Website: http://schema.csail.mit.edu

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

schema, 9

Index

E

explore_param_mincorr() (*schema.SchemaQP method*), 10

F

feature_weights() (*schema.SchemaQP method*), 11
fit() (*schema.SchemaQP method*), 12
fit_transform() (*schema.SchemaQP method*), 13

G

get_start_end_dist_correlations() (*schema.SchemaQP method*), 13

R

reset_maxwt_param() (*schema.SchemaQP method*), 13
reset_mincorr_param() (*schema.SchemaQP method*), 14

S

schema (*module*), 9
SchemaQP (*class in schema*), 9

T

transform() (*schema.SchemaQP method*), 14