UniCell Deconvolve

Release 0.1.0

Daniel Charytonowicz

Apr 26, 2023

GETTING STARTED

UniCell Deconvolve (UCD) is a pre-trained deep learning model that provides context-free estimations of cell type fractions from whole transcriptome expression data for bulk, single-cell and spatial transcriptomics data. The model is trained on the world's largest fully-integrated scRNA-Seq training database, comprising 28M+ single cells spanning 840+ cell types from 899 studies to date. Extensive benchmarking shows UCD favors comperably when compared with reference-based deconvolution tools, without the need for pretraining. UCD demonstrates strong multi-task performance across a range of deconvolution challenges spanning several transcriptomic data modalities, disease types, and tissues.

- Read our paper at Nature Communications.
- Install via `pip install ucdeconvolve`.
- Discuss development on GitHub.

CHAPTER

ONE

INSTALLATION

1.1 From New Environment (Conda)

We highly recommend installing conda or miniconda and creating a new virtual environment to install ucdeconvolve as it reduces the likelihood of version conflicts. We suggest the following conda environment which will be compatible with jupyter notebooks.

```
conda create -n ucdenv python=3.8 pytables jupyter jupyterlab
conda activate ucdenv
pip install ucdeconvolve
```

Note: As ucdeconvolve uses HDF files as intermin datastore types, the `pytables` package **must** be installed with conda before installing ucdeconvolve to ensure that all underlying hdf5 dependencies are present.

1.2 From Existing Environment (Pip)

To install ucdeconvolve using pypi in an existing environment, first ensure that `pytables` has been installed if you are using an existing conda / virtualenv and that the `tables` package is present under `pip list`. Then, simply run the following command to install ucdeconvolve and any other required dependencies.

```
pip install ucdeconvolve
```

CHAPTER

TWO

REGISTRATION

2.1 Create a New Account

Registration for UCDeconvolve is straightforward and can be done either in a notebook environment or terminal. We offer dynamic registration with live user input, or programmatic registration with al fields as an input.

Load the ucdeconvolve package and run the 'ucd.api.register' command as shown below. Follow the instructions by inputting the required information at each step.

```
[ ]: import ucdeconvolve as ucd

ucd.api.register()
```

Alternatively, one can perform registration entirely programmatically by invoking the function as follows:

```
[ ]: username = "USERNAME"
password = "PASSWORD"
firstname = "FIRSTNAME"
lastname = "LASTNAME"
email = "EMAIL"
institution = "INSTITUTION"

ucd.api.register(
    username = username,
    password = password,
    firstname = firstname,
    lastname = lastname,
    email = email,
    institution=institution
    dynamic = False
)
```

2.2 Account Activation

Upon completion of the initial registration form, you will recieve an email at the address specified with an activation code. Copy the code and paste it back into the waiting input prompt in order to activate your account. Upon account activation, a followup email will be sent with your user API key. This key will also be automatically appended to your working 'ucd' module instance.

If you accidentally close the registration function instance prior to adding the activation code, you can still activate your account by invoking the `ucd.api.activate` command and passing the activation code directly there.

```
[ ]: activation_code = "ACTIVATION_CODE"
ucd.api.activate(activation_code)
```

2.3 New Session Authentication

When you start a new python instance, you can authenticate your API by simply calling the `ucd.tl.authenticate([token])` function and passing your user access token. It will be appended to your settings module at `ucd.settings.token`

CHAPTER

THREE

API OVERVIEW

Import the ucdeconvolve package using `import ucdeconvolve as ucd`. The package contains four main modules, described in detail below.

Note: Authenticate a new python session using `ucd.api.authenticate`

API: `api`

API functions allow for user registration, account activation and authentication for service invocation.

api.register([username, password, ...])	Registers a New User
api.activate([code])	Activate User Account
api.authenticate(token)	Authenticate

Tools: `tl`

Tools module contains the three primary prediction functions of ucdeconvolve.

tl.base(data[, token, split, sort, ...])	UniCell Deconvolve: Base
tl.explain(data, celltypes[, groupby, ...])	UniCell Deconvolve: Explain
tl.select(data, reference[, token, ...])	UniCell Deconvolve: Select

Plotting: `pl`

Plotting functions for embedding and spatial are designed to interface as wrappers around scanpy functions such as `sc.pl.embedding` and `sc.pl.spatial` with additional functionality to enable construciton of plots similar to those in the ucdeconolve paper.

pl.embedding(adata[, basis, color, key, ...])	Plot Deconvolution
pl.spatial(adata[, color, key, category, ...])	Plot Spatial
pl.base_clustermap(adata[, groupby, ...])	Plot Clustered heatmap of top celltype predictions grouped by a column in 'adata.obs'
pl.explain_boxplot(adata[, key, celltypes, ...])	Plot Boxplots of Feature Attributions By Gene
pl.explain_clustermap(adata[, key, n_top_genes])	Plot Explanation Results as Clustermap

Utilities: `utils`

Utilities module contains useful functions for interfacing with results of deconvolution functions and preparing prediction queries.

utils.read_results(adata[, key, category, ...])	Read deconvolution results from an annotated dataset and return a dataframe.
utils.assign_top_celltypes(adata[, key, ...])	Gets top deconvolution predictions by cluster.
utils.get_base_celltypes([root, category, ...])	Get UCDBase CellTypes
utils.get_prebuilt_reference(reference[, ...])	Get Prebuilt Reference
utils.list_prebuilt_references([token])	List Prebuilt References

3.1 API

UniCell Deconvolve - Cell Type Deconvolution For Transcriptomic Data.

ucdeconvolve.api.**activate**(*code: Optional[str] = None*) → None

Activate User Account

Activates account with an acitvation code recieved via email

Parameters
code – Activation code emailed to user upon registration

Return type
None

ucdeconvolve.api.**authenticate**(*token: str*) → None

Authenticate

Updates user access token credentials

Parameters
token – Valid user token

Return type
None

ucdeconvolve.api.**register**(*username: Optional[str] = None*, *password: Optional[str] = None*, *firstname: Optional[str] = None*, *lastname: Optional[str] = None*, *email: Optional[str] = None*, *institution: Optional[str] = None*, *dynamic: bool = True*) → None

Registers a New User

Parameters

- **username** – Username for new account
- **password** – Password for new account
- **firstname** – First name of new user
- **lastname** – Last name of new user
- **email** – Valid email address of new user. Note that an email will be sent for account activation.
- **institution** – The insitution, academic or private, the user is affiliated with.
- **dynamic** – Whether or not to prompt for inputs dynamically, default is True.

Return type
Either nothing or waits for user to complete.

3.2 Tools

UniCell Deconvolve - Cell Type Deconvolution For Transcriptomic Data.

ucdeconvolve.tl.**base**(*data: Union[AnnData, DataFrame]*, *token: Optional[str] = None*, *split: bool = True*, *sort: bool = True*, *propagate: bool = True*, *return_results: bool = False*, *key_added: str = 'ucdbase'*, *use_raw: Union[bool, Tuple[bool, bool]] = True*, *verbosity: Optional[int] = None*) → Optional[AnnData]

UniCell Deconvolve: Base

Predicts cell type fractions for provided transcriptomic data.

Parameters

- **data** – Transcriptomic data (obs x genes) to predict cell type fractions. Can be either a dataframe or annotated dataset. Note that in any case data will be converted to an annotated datset object before proceeding.
- **token** – UCDeconvolve API access token. If None, defaults to settings parameter.
- **split** – Whether or not to split underlying data into three categories, primary, cancer cell_line. Helps with interpretability downstream, default is True.
- **sort** – Sort columns of results by mean predictions. Default True.
- **propagate** – Whether or not to perform belief propagation and pass predictions up a cell-type heiarchy. helpful in interpreting overall deconvolution results. default is True.
- **return_results** – Whether or not to return the predictions dict from the function, default to false as all data is written to anndata object either passed in, or created when passing in a dataframe, which will in that case be returned by default. Also returns the underlying anndata if it is a view as copying can destroy context internally.
- **use_raw** – Use counts in 'adata.raw'. Default True, as by convention in single cell analysis, log1p scaled counts before HVG filter are kept here.
- **verbosity** – Level of verbosity for function information. Default is taken from package, set to 'logging.DEBUG' for more detailed information.

Returns

adata_mixture_orig – Results appended to anndata object if return_results or if original input was dataframe.

Return type

anndata.AnnData

ucdeconvolve.tl.**explain**(*data: AnnData*, *celltypes: Union[str, List[str], Dict[Union[int, str], str]]*, *groupby: Optional[str] = None*, *group_n: int = 16*, *group_frac: Optional[float] = None*, *token: Optional[str] = None*, *return_results: bool = False*, *key_added: str = 'ucdexplain'*, *use_raw: Union[bool, Tuple[bool, bool]] = True*, *verbosity: Optional[int] = None*) → Optional[AnnData]

UniCell Deconvolve: Explain

Explains cell type fraction prediction for provided transcriptomic data.

Parameters

- **data** – Transcriptomic data (obs x genes) to predict cell type fractions. Can be either a dataframe or annotated dataset. Note that in any case data will be converted to an annotated datset object before proceeding.

- **celltypes** – Name of cell type(s) to get explanations for. If a single string is passed, this celltype is used for all samples. If a list of strings is passed, the list must be the same length as the dataset and each entry corresponds to which celltype to get explanatons for in the whole dataset. If a dictionary is passed, the key should corresponding to an 'adata.obs' column defined by 'groupby', alliowing for celltype expalantions to be generated specific to different clusters or conditions.
- **groupby** – Groupby key in 'adata.obs' to arrange search for celltypes. If celltypes is given as a dict, this must be defined.
- **group_n** – The number of samples to subsample from each group for explanations, as this is an expensive operation and most cells in a cluster will yield similar results.
- **token** – UCDeconvolve API access token. If None, defaults to settings parameter.
- **return_results** – Whether or not to return the predictions dict from the function, default to false as all data is written to anndata object either passed in, or created when passing in a dataframe, which will in that case be returned by default. Also returns the underlying anndata if it is a view as copying can destroy context internally.
- **use_raw** – Use counts in 'adata.raw'. Default True, as by convention in single cell analysis, log1p scaled counts before HVG filter are kept here.
- **verbosity** – Level of verbosity for function information. Default is taken from package, set to 'logging.DEBUG' for more detailed information.

Returns

adata_mixture_orig – Results appended to anndata object if return_results or if original input was dataframe.

Return type

anndata.AnnData

ucdeconvolve.tl.**select**(*data: Union[AnnData, DataFrame]*, *reference: Union[AnnData, DataFrame, List[str], str]*, *token: Optional[str] = None*, *reference_key: str = 'celltype'*, *ignore_categories: Optional[Iterable[str]] = None*, *method: str = 'both'*, *return_results: bool = False*, *key_added: str = 'ucdselect'*, *use_raw: Union[bool, Tuple[bool, bool]] = True*, *verbosity: Optional[int] = None*) → Optional[AnnData]

UniCell Deconvolve: Select

Predicts cell type fractions for provided transcriptomic data using a user-specified reference. Leverages transfer learning from base UCD model embeddings.

Parameters

- **data** – Transcriptomic data (obs x genes) to predict cell type fractions. Can be either a dataframe or annotated dataset. Note that in any case data will be converted to an annotated datset object before proceeding.
- **reference** – Transcriptomic data (obs x genes) to be used as a reference. Can be either a dataframe or annotated dataset. Note that if a dataframe is passed, row indices should correspond to categories for reference. If a list of strings is passed, these strings should correspond to reference profiles from the unicell cell type registry as any other names will throw an error. If a string alone is passed, we look for a pre-built reference in the ucd backend.

  Currently valid prebuilt references include:

  allen-mouse-cortex : Mouse whole-brain cortex (44 cell types) enge2017-human-pancreas : Human pancreas (6 cell types) lee-human-pbmc-covid : Human PBMC (24 cell types)
- **token** – UCDeconvolve API access token. If None, defaults to settings parameter.

- **reference_key** – The key in reference.obs or index if reference is a dataframe to use to perform the grouping operation.
- **method** – The method used for building a reference matrix. Must be one of two strings, either "embeddings" or "features". If "embeddings", the UCD base model is queried to return an embedding vector to represent celltype mixtures, and is used to generated representations for transfer learning. If "features", model defaults to using features in the reference matrix, similar to other available methods. Reccomended to use "both" in all cases.
- **return_results** – Whether or not to return the predictions dict from the function, default to false as all data is written to anndata object either passed in, or created when passing in a dataframe, which will in that case be returned by default. Also returns the underlying anndata if it is a view as copying can destroy context internally.
- **ignore_categories** – Categories in 'reference.obs['reference_key']' to ignore. Default is None.
- **use_raw** – Use counts in 'adata.raw'. Default True, as by convention in single cell analysis, log1p scaled counts before HVG filter are kept here. Note that if a tuple is passed, it will selectively apply use_raw to DATA and then REF in that order.
- **verbosity** – Logging verbosity, if None defaults to settings value.

Returns

adata_mixture_orig – Results appended to anndata object if return_results or if original input was dataframe.

Return type

anndata.AnnData

3.3 Plotting

UniCell Deconvolve - Cell Type Deconvolution For Transcriptomic Data.

ucdeconvolve.pl.**base_clustermap**(*adata: AnnData*, *groupby: str = 'leiden'*, *category: Optional[str] = None*, *key: str = 'ucdbase'*, *n_top_celltypes: int = 30*, *max_filter: float = 0.1*, ***kwargs*) → Optional

Plot Clustered heatmap of top celltype predictions grouped by a column in 'adata.obs'

Parameters

- **adata** – The annotated dataset with deconvolution data
- **groupby** – What column in 'adata.obs' to group celltype predictions by (i.e. 'leiden').
- **category** – Which category of prediction data to use if split, or all of not split.
- **key** – Key for deconvolution results, default is 'ucdbase'
- **n_top_celltypes** – Number of top celltypes per category to take and plot. Smaller means only the most common types.
- **kwargs** – Keyword attributes for clustermap. See seaborn.clustermap for details.

Return type

A clustermap

ucdeconvolve.pl.**embedding**(*adata: AnnData*, *basis: str = 'X_umap'*, *color: Optional[Union[str, List[str]]] = None*, *key: str = 'ucdbase'*, *category: Optional[str] = None*, ***kwargs*) → Optional[object]

Plot Deconvolution

Wrapper for scanpy function 'sc.pl.embedding' to help plot deconvolution results. Follows the parameter conventions of its wrapped function with some exceptions noted below.

Functions to read the results from the deconvolution run given by key, subset to category and then appends them to the 'adata.obs' dataframe of a copy of the passed adata object, allowing standard plotting module to the visualize the results.

Parameters

- **adata** – anndata object to plot
- **basis** – The embedding to plot using, for example 'X_pca' or 'X_umap' if calculated and present.
- **color** – Refers to the cell type we want to plot contained within the category of split and result specificed by key. Can be one or more.
- **key** – location of data in obsm and uns to plot containing numerical data and headers, respectively. Can be either 'ucdbase' or 'ucdselect'.
- **category** – if the data results are split, indicate which split to use for plotting. defaults to 'all' assuming that we did not split the output. valid categories are 'all', 'primary', 'cell_lines', and 'cancer'.
- **kwargs** – attributes to pass along to 'sc.pl.embedding', see documentation for details.

Return type
Plot(s)

ucdeconvolve.pl.**explain_boxplot**(*adata: AnnData*, *key: str = 'ucdexplain'*, *celltypes: Optional[Union[str, List[str]]] = None*, *n_top_genes: int = 16*, *ncols: int = 5*, *figsize: Tuple[int, int] = (3, 3)*, *dpi: int = 150*, *titlewidth: int = 24*, *barcolor: str = 'lightblue'*, *ax: Optional[Axes] = None*, *return_fig: bool = False*) → Optional[Axes]

Plot Boxplots of Feature Attributions By Gene

Parameters

- **adata** – Annotated dataset with ucdexplain results.
- **key** – UCDExplain results key, default is 'ucdexplain'
- **celltypes** – The celltypes from the given run to plot. if none then plots all.
- **n_top_genes** – Number of top attribution genes to plot.
- **ncols** – Number of columns to plot for multiple celltypes before creating a new row
- **figsize** – Size of individual subplot figure
- **dpi** – Pixel density of plot
- **titlewidth** – Width of subplot title before newline
- **barcolor** – Color of bars
- **ax** – Optional axes to plot on.
- **return_fig** – Return figure or not

Returns
fig – Figure with underlying subplots

Return type
plt.Figure

ucdeconvolve.pl.**explain_clustermap**(*adata: AnnData*, *key: Union[str, List[str]] = 'ucdexplain'*, *n_top_genes: int = 64*, ***kwargs*) → Optional[Axes]

Plot Explanation Results as Clustermap

Plot Clustered heatmap of top feature attribution predictions grouped by the celltypes passed to the `ucd.tl.explain` function.

Parameters

- **adata** – The annotated dataset with deconvolution data
- **key** – Key for deconvolution results, default is 'ucdexplain'.
- **n_top_genes** – Number of top feature attributes (genes) per celltype
- **kwargs** – Keyword attributes for clustermap. See seaborn.clustermap for details.

Return type
A clustermap

ucdeconvolve.pl.**spatial**(*adata: AnnData*, *color: Optional[Union[str, List[str]]] = None*, *key: str = 'ucdbase'*, *category: Optional[str] = None*, *labels: Optional[List[str]] = None*, *colormaps: Optional[List[ListedColormap]] = None*, *cbar_nrows: int = 4*, *title: str = ''*, ***kwargs*) → Optional[object]

Plot Spatial

Wrapper for scanpy function 'sc.pl.spatial' to help plot deconvolution results on spatial data. Follows parameter conventions of wrapped function with some exceptions.

Functions to read the results from the deconvolution run given by key, subset to category and then appends them to the 'adata.obs' dataframe of a copy of the passed adata object, allowing standard plotting module to the visualize the results.

Parameters

- **adata** – anndata object to plot
- **color** – Refers to the cell type(s) we want to plot contained within the category of split and result specificed by key. Can be one or more. If more than one string is passed we try to plot an overlapped plot.
- **key** – location of data in obsm and uns to plot containing numerical data and headers, respectively. Can be either 'ucdbase' or 'ucdselect'.
- **category** – if the data results are split, indicate which split to use for plotting. defaults to 'all' assuming that we did not split the output. valid categories are 'all', 'primary', 'cell_lines', and 'cancer'.
- **labels** – Labels for each color being plotted when using the overlapping colormap spatial function.
- **colormaps** – Optional custom colormaps to use for each color.
- **cbar_nrows** – Number of rows to spread cbars across, default is 3.
- **kwargs** – attributes to pass along to 'sc.pl.spatial', see documentation for details.

Return type
Plot(s)

3.4 Utilities

UniCell Deconvolve - Cell Type Deconvolution For Transcriptomic Data.

ucdeconvolve.utils.**assign_top_celltypes**(*adata: AnnData*, *key: str = 'ucdbase'*, *category: Optional[str] = None*, *groupby: Optional[str] = None*, *inplace: bool = True*, *key_added: str = 'pred_celltype'*, *knnsmooth_neighbors: Optional[int] = None*, *knnsmooth_cycles: int = 1*) → Union[List[str], Dict[str, str]]

Gets top deconvolution predictions by cluster.

Parameters

- **adata** – Annotated dataset with deconvolution results stored
- **groupby** – Optional variable, if not passed then the top celltype is predicted for each individual sample by row.
- **category** – Which split category to use, defaults if none.
- **key** – Key for deconvolution results, default key is 'ucdbase'
- **inplace** – Whether or not to attach the result to the anndata object 'obs' as a column
- **key_added** – The key to add as a column.
- **knnsmooth_neighbors** – Optional smoothing for predictions, uses neighbors graph calculated using sc.tl.neighbors. If not none, passed integer referring to number of nearest neighbors to consider for smoothing, reccomended 3.
- **knnsmooth_cycles** – Number of cycles to repeat smoothing, default 1.

Returns

celltypes – If no groupby is passed return a list of strings corresponding to the top celltype that can be used for annotation. If a group is passed, returns a dicitonary mapping group label to celltype.

Return type

Union[List[str], Dict[str, str]]

ucdeconvolve.utils.**get_base_celltypes**(*root: Optional[str] = None*, *category: Optional[str] = None*, *as_digraph: bool = False*) → List[str]

Get UCDBase CellTypes

Return a list of UCDbase celltypes.

Parameters

- **root** – Optional root cell type, if set, returns all decendants of that celltype along the UCD celltype hierarchy.
- **category** – If category is set, overrides root behavior and returns a list of all celltypes in the subset category. Must be either 'primary', 'lines', or 'cancer'.
- **as_digraph** – If true, return the underlying networkX graph that can be used to visualize result directly when calling a root node. Call root node "cell" to return all nodes in graph.

Returns

celltypes – A list of celltypes, either all or subsets.

Return type

List[str]

ucdeconvolve.utils.**get_prebuilt_reference**(*reference: str*, *token: Optional[str] = None*, *cache: bool = True*) → AnnData

Get Prebuilt Reference

Downloads a pre-made reference dataset.

Parameters

- **reference** – String name of prebuilt reference to get.
- **token** – Optional access token, if not passed default token is retrieved from settings.
- **cache** – If true, save a loaded file into cachedir and try to reload it before downloading again.

Returns
adata – Annotated dataset object containing the prebuilt reference.

Return type
anndata.AnnData

ucdeconvolve.utils.**list_prebuilt_references**(*token: Optional[str] = None*) → AnnData

List Prebuilt References

Lists available pre-built references

Parameters
token – Optional access token, if not passed default token is retrieved from settings.

Returns
adata – Annotated dataset object containing the prebuilt reference.

Return type
anndata.AnnData

ucdeconvolve.utils.**read_results**(*adata: AnnData*, *key: Optional[str] = None*, *category: Optional[str] = None*, *celltypes: Optional[Union[str, List[str]]] = None*, *explain_n_top_genes: Optional[int] = None*) → DataFrame

Read deconvolution results from an annotated dataset and return a dataframe.

Parameters

- **adata** – Annotated dataset with deconvolution results stored.
- **key** – Key for deconvolution results. If not passed, will default to searching for 'ucdbase', then 'ucdselect', and lastly 'ucdexplain'.
- **category** – Which split category to use. Defaults to 'all' if running non-base model. If run base and split was made, uses 'primary'.
- **celltypes** – The celltypes specifically to read results from, if None then all celltypes from explanation are read and combined in a multi-index dataframe. For ucdselect and ucdbase celltypes will subset the returned dataframe.
- **explain_n_top_genes** – Number of genes to extract when reading explanations, if None then return all genes.

Compatability: `compat`

The compatability module allows for users of earlier builds of ucdeconvolve who have existing workflows to continue leveraging legacy code with minimal required changes.

Warning: This module will be removed in the near future.

`compat.deconvolve`(data, token[, split, ...])	UniCell Deconvolve
`compat.read_results`(adata[, category, key])	Read deconvolution results from an annotated dataset and return a dataframe.

CHAPTER

FOUR

BASIC TUTORIALS

Browse some basic tutorials for how to get started with using UCD for your projects using the navbar.

4.1 Single Cell RNA-Seq

In this tutorial we are going to run through how UCDeconvolve can be used to aid in the analysis and annotation of single-cell RNA-Sequencing data. For this tutorial, we are going to use the 'pbmc3k' dataset provided in the scanpy datasets model at `sc.datasets.pbmc3k()`.

4.1.1 Loading Packages & Authenticating

The first step in this analysis will be to load scanpy and ucdeconvolve after following the installation and registration instructions, and authenticate our API. In this tutorial we saved our user access token in the variable `TOKEN`.

[80]:
```
import scanpy as sc
import ucdeconvolve as ucd

ucd.api.authenticate(TOKEN)
```

```
2023-04-25 15:57:57,357|[UCD]|INFO: Updated valid user access token.
```

| .. note:: By default the logging level is set to `DEBUG`. To change logging levels you can import `logging` and set `ucd.settings.verbosity` directly. To reduce logs, change `verbosity` to `logging.INFO`. In general we recommend keeping logging to `DEBUG` to provide status updates on a running deconvolution job.

4.1.2 Loading & Preprocessing Data

We will now begin by loading our pbmc dataset.

[3]:
```
adata = sc.datasets.pbmc3k()
```

We will save raw counts data into adata, which can serve as an input to ucd functions. Unicell will detect non-logarithmized counts data and automatically normalize our data. We will run a quick built-in preprocssing functions using scanpy to obtain some clustered data. This step will take a minute or two to complete.

[ ]:
```
adata.raw = adata

sc.pp.recipe_seurat(adata)
```

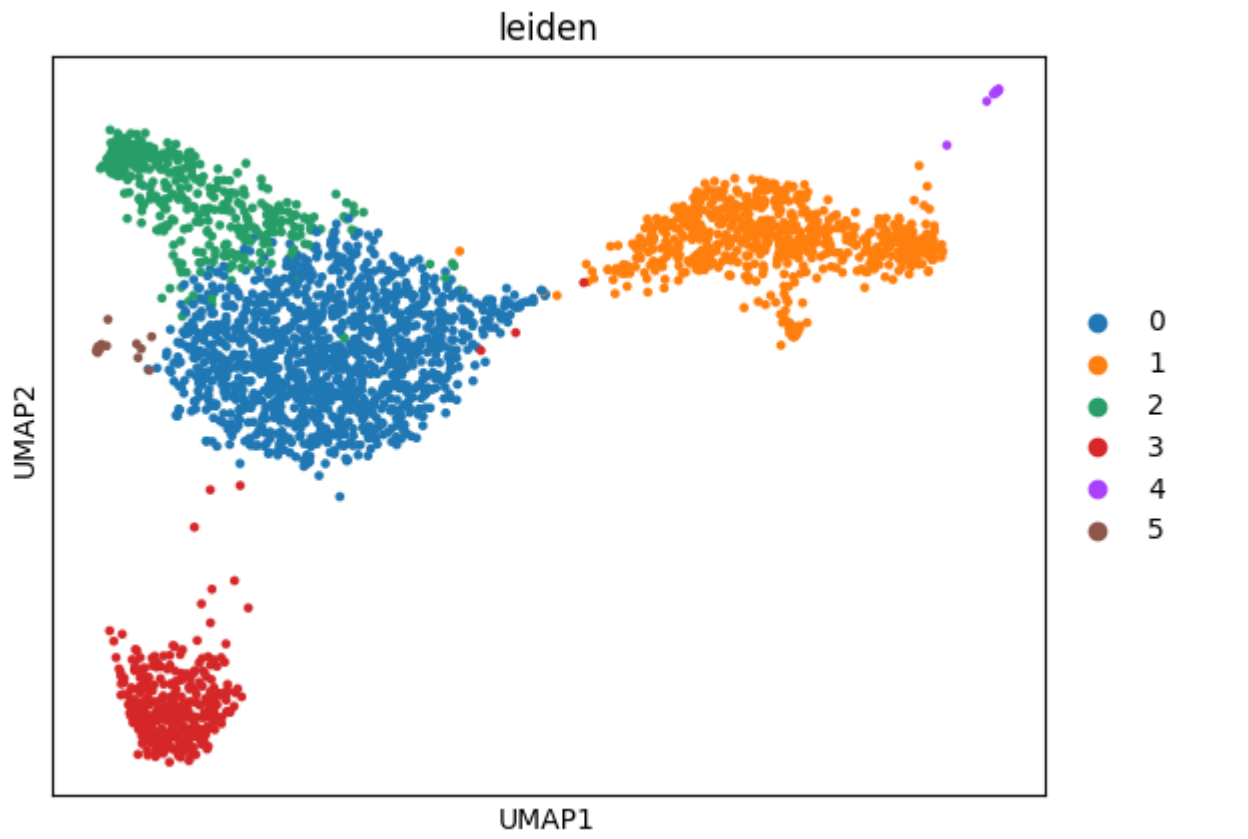
(continues on next page)

(continued from previous page)

```
sc.tl.pca(adata)
sc.pp.neighbors(adata, n_neighbors = 30)
sc.tl.umap(adata, min_dist = 0.1)
sc.tl.leiden(adata, resolution = 0.75)
```

We plot the UMAP of our dataset using leiden clusters as an overlay and see the following image:

[29]:
```
sc.pl.umap(adata, color = 'leiden')
```



4.1.3 Initial Cluster Identification Using UCDBase

To get a general sense of the celltypes most likely present in this dataset, we want to first run `ucd.tl.base` which will return context-free deconvolutions of cell type states.

[7]:
```
ucd.tl.base(adata)
```

```
2023-04-25 13:10:28,425|[UCD]|INFO: Starting UCDeconvolveBASE Run. | Timer Started.
Preprocessing Dataset | 100% (11 of 11) || Elapsed Time: 0:00:01 Time:  0:00:01
2023-04-25 13:10:30,501|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 13:10:31,339|[UCD]|INFO: Upload Complete | Elapsed Time: 0.838 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | / |#| 0 Elapsed Time: 0:00:00
Waiting For Submission : QUEUED | Queue Size : 1 | | |#| 3 Elapsed Time: 0:00:04
Waiting For Submission : RUNNING | Queue Size : 1 | / |#| 3 Elapsed Time: 0:00:04
Waiting For Completion | 100% (2700 of 2700) || Elapsed Time: 0:00:28 Time:  0:00:28
```

(continues on next page)

(continued from previous page)

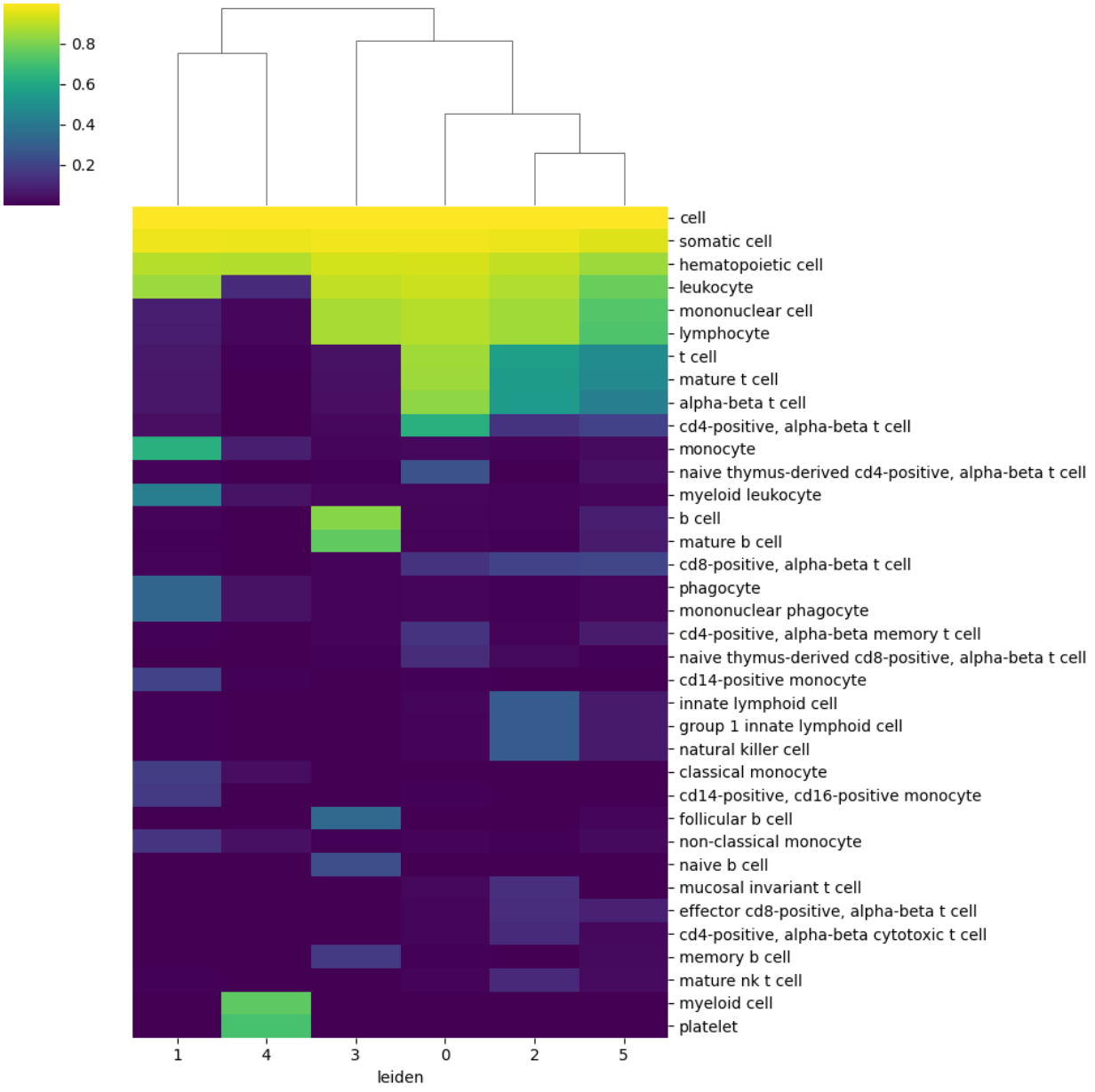
```
2023-04-25 13:11:07,163|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 13:11:08,505|[UCD]|INFO: Download Complete | Elapsed Time: 1.342 (s)
2023-04-25 13:11:09,238|[UCD]|INFO: Run Complete | Elapsed Time: 40.812 (s)
```

4.1.3.1 Plotting Clustermap

To get a general sense of the deconvolution results, let's plot a clustermap that aggregates base predictions on the basis of leiden cluster using the function `ucd.pl.base_clustermap`

[58]:
```
ucd.pl.base_clustermap(adata, groupby = 'leiden', n_top_celltypes=75)
```

[58]:
```
<seaborn.matrix.ClusterGrid at 0x16b06d430>
```

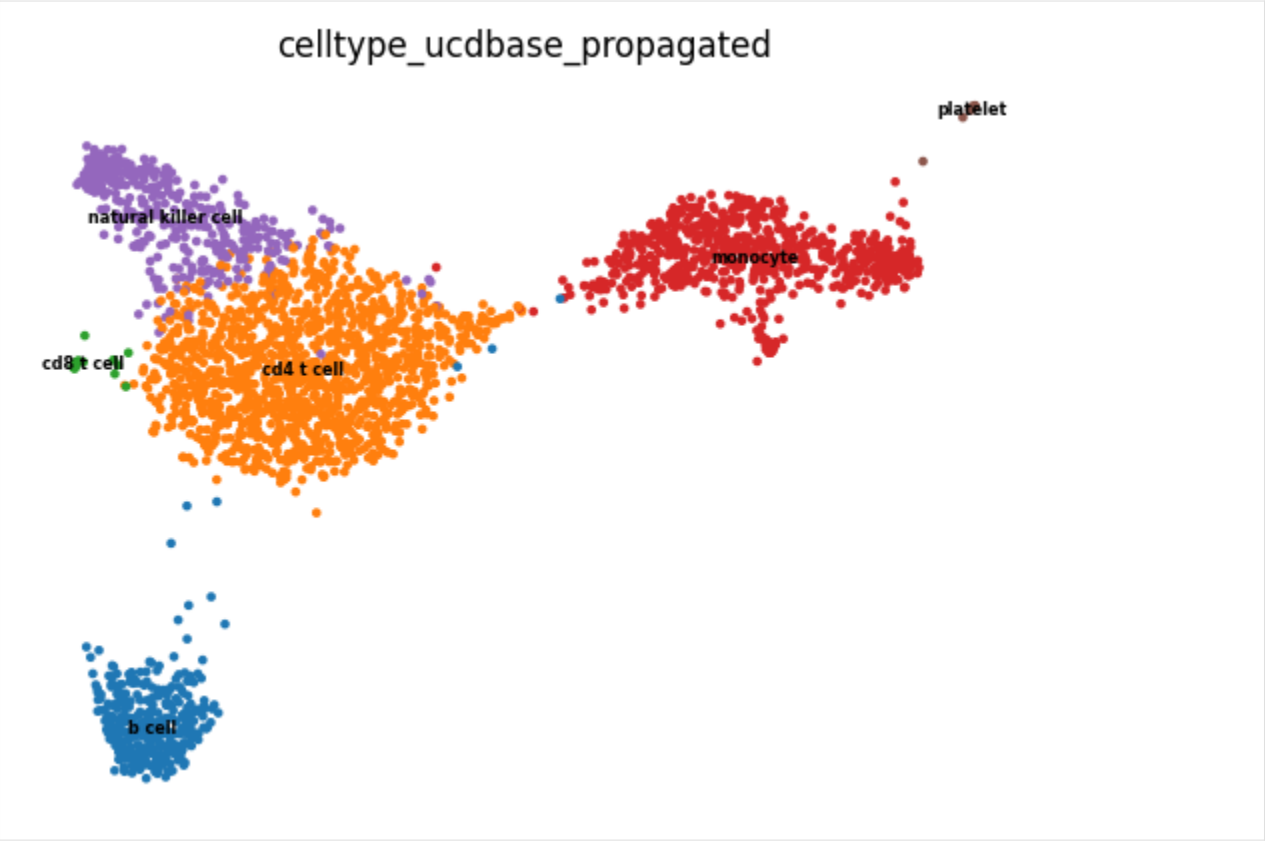


We can see that the predictions shown in the clustermap are hierarchecal. By default, ucdeconvolve base performs belief propagation, which takes flattened predictions and aggregates them up a cell type heirarchy. This flag can be set in the `ucd.tl.base` function as `propagate = False`. For most cases we reccomend peforming belief propagation, as it accounts for uncertainty in ground-truth labels used during training.

In either case, we can use this clustering information to label our dataset by selecting the most likely detailed cell subtype, using ucdbase to guide us to an answer faster than performing manual curation.

[59]:
```
label = "celltype_ucdbase_propagated"
adata.obs[label] = 'unknown'
adata.obs.loc[adata.obs.leiden.isin(("1",)), label] = "monocyte"
adata.obs.loc[adata.obs.leiden.isin(("4",)), label] = "platelet"
adata.obs.loc[adata.obs.leiden.isin(("3",)), label] = "b cell"
adata.obs.loc[adata.obs.leiden.isin(("0",)), label] = "cd4 t cell"
adata.obs.loc[adata.obs.leiden.isin(("2",)), label] = "natural killer cell"
adata.obs.loc[adata.obs.leiden.isin(("5",)), label] = "cd8 t cell"

sc.pl.umap(adata, color = 'celltype_ucdbase_propagated', legend_loc = 'on data',
           legend_fontsize = "xx-small", frameon = False)
```



4.1.4 Examining Feature Attributions with UCDExplain

To gain some additional insight into the cell types being predicted for each cluster, we can leveraged **integrated gradients** which is implemented in the `ucd.tl.explain` module. This method takes a target output for the ucdbase model and computes attribution scores for all input genes. A positive attribution score indicates that a given gene's expression is positively associated with the prediction of that given cell type (i.e. canonical marker genes tend to have high feature attribution scores for their corresponding cell types) while a negative attribution score indicates that a given gene's expression is negatively associated with the prediction of that given cell type (i.e. it may be a canonical marker of another cell type). We can use feature attributions to validate some of our predictions by confirming that the top genes associated with a given cell type are concordant with biological phenomena.

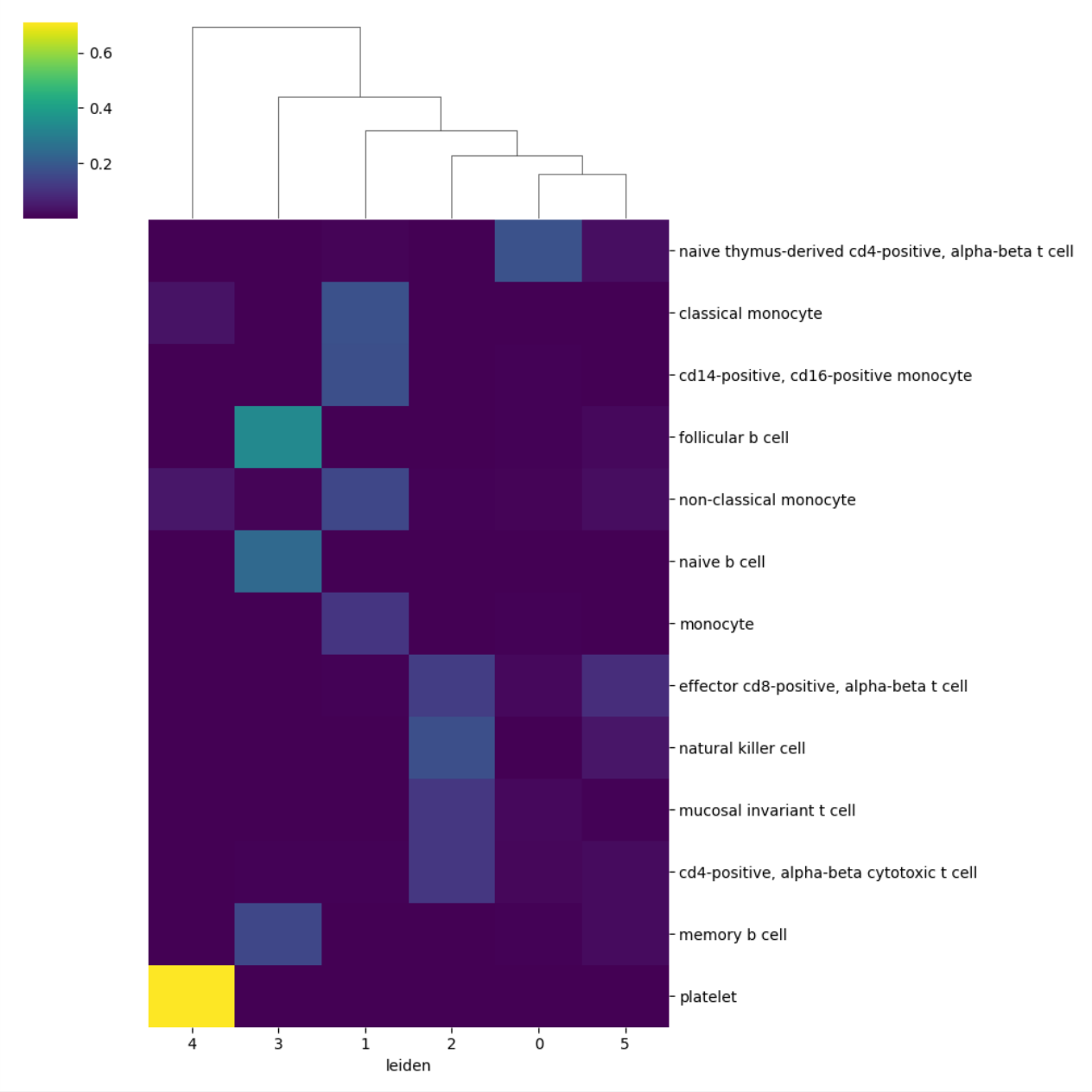
4.1.4.1 Examining Raw Predictions

To do this, we first must examine the models raw, non-propagated predictiosn. As feature attributions using integrated gradients relies on the core wieghts underpinning the ucdbase deep learning model, it does not consider belief propagation which is a post-processing function. Therefore, we ned to first get a sense of the "raw" cell type predictions made for each cluster.

We can plot raw cell type predictions using the same clustermap function above, but this time adding an additional parameter.

[60]:
```
ucd.pl.base_clustermap(adata, groupby = 'leiden', category = 'raw', n_top_celltypes = 75)
```

[60]:
```
<seaborn.matrix.ClusterGrid at 0x16b0c27f0>
```

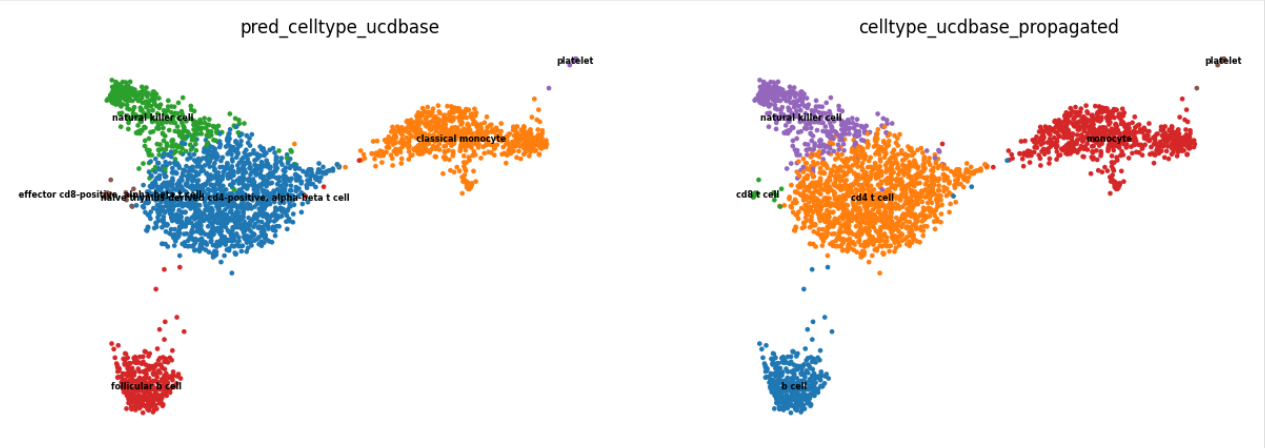


We can immediately see that these predictions are alot more specific for each cluster. We include a utility function to assign target cell types from predictions to each cluster, and can use this 'raw' prediction information to collect feature attributions for each cluster individually. This functions creates a new column in `adata.obs` entitled `pred_celltype_{key}` which in our case key, which represents the name of the run call, defaults to `ucdbase`, therefore our column is called `pred_celltype_ucdbase`.

[61]:
```
ucd.utils.assign_top_celltypes(adata, category = "raw", groupby = "leiden")
```

Let's plot the resulting 'raw' assigned celltypes over our UMAP and compre them with our aggregated propagation assignments we performed semi-manually.

[62]:
```
sc.pl.umap(adata, color = ["pred_celltype_ucdbase", "celltype_ucdbase_propagated"],
           legend_loc = 'on data', legend_fontsize = 'xx-small', frameon = False)
```



4.1.4.2 Running UCDExplain

Warning: As feature attributions is a highly computationally intensive operation, for scRNA-Seq data where each cell is considered a sample, we **highly recommend utilizing the subsampling capabilities** built into `ucd.tl.explain` to speed up peformance. When run at the cluster-level, we find that subsampling a sufficient number of cells from each cluster provides the same level of feature attribution granularity as one would gain running all cells.

Let's start by retrieving a dictionary mapping our groups in `leiden` to our raw celltypes. The `assign_top_celltypes` function has a parameter that can be set `inplace = False` which will return the dictionary directly.

[63]:
```
celltypes = ucd.utils.assign_top_celltypes(adata, category = "raw", groupby = "leiden",␣
↪inplace = False)
```

Now let's go ahead and run `ucd.tl.explain` `withour group set toleiden``. We will take as many as 64 cells per group to subsample with.

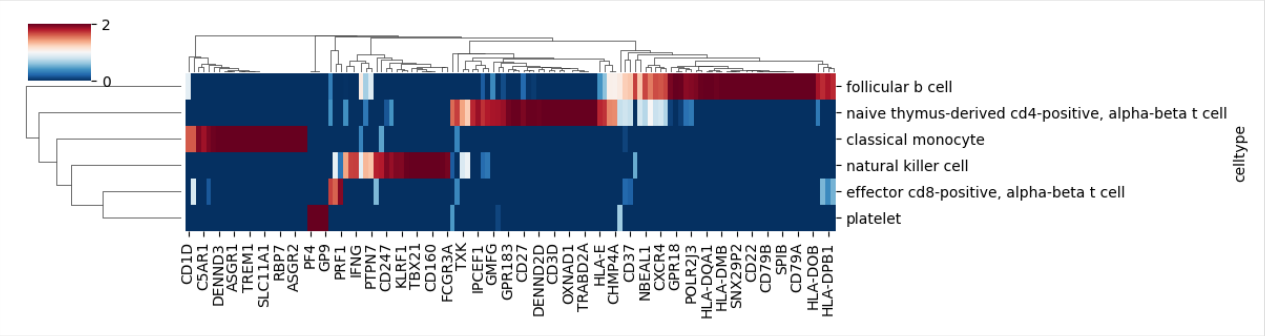
[37]:
```
ucd.tl.explain(adata, celltypes = celltypes, groupby = "leiden", group_n = 64)
```

```
2023-04-25 14:52:06,134|[UCD]|INFO: Starting UCDeconvolveEXPLAIN Run. | Timer Started.
Preprocessing Dataset | 100% (2 of 2) |##| Elapsed Time: 0:00:00 Time:  0:00:00
2023-04-25 14:52:06,884|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 14:52:07,714|[UCD]|INFO: Upload Complete | Elapsed Time: 0.829 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | / |#| 0 Elapsed Time: 0:00:00
Waiting For Submission : QUEUED | Queue Size : 1 | - |#| 1 Elapsed Time: 0:00:01
Waiting For Submission : RUNNING | Queue Size : 1 | \ |#| 1 Elapsed Time: 0:00:01
Waiting For Completion | 100% (283 of 283) || Elapsed Time: 0:00:57 Time:  0:00:57
2023-04-25 14:53:09,083|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 14:53:09,311|[UCD]|INFO: Download Complete | Elapsed Time: 0.227 (s)
2023-04-25 14:53:09,932|[UCD]|INFO: Run Complete | Elapsed Time: 63.798 (s)
```

Let's get a sense of the marker genes being used to classify each of the cell types. We can quickly obtain a decent visualization using the `ucd.pl.explain_clustermap` function.

[73]:
```
ucd.pl.explain_clustermap(adata, n_top_genes= 128)
```

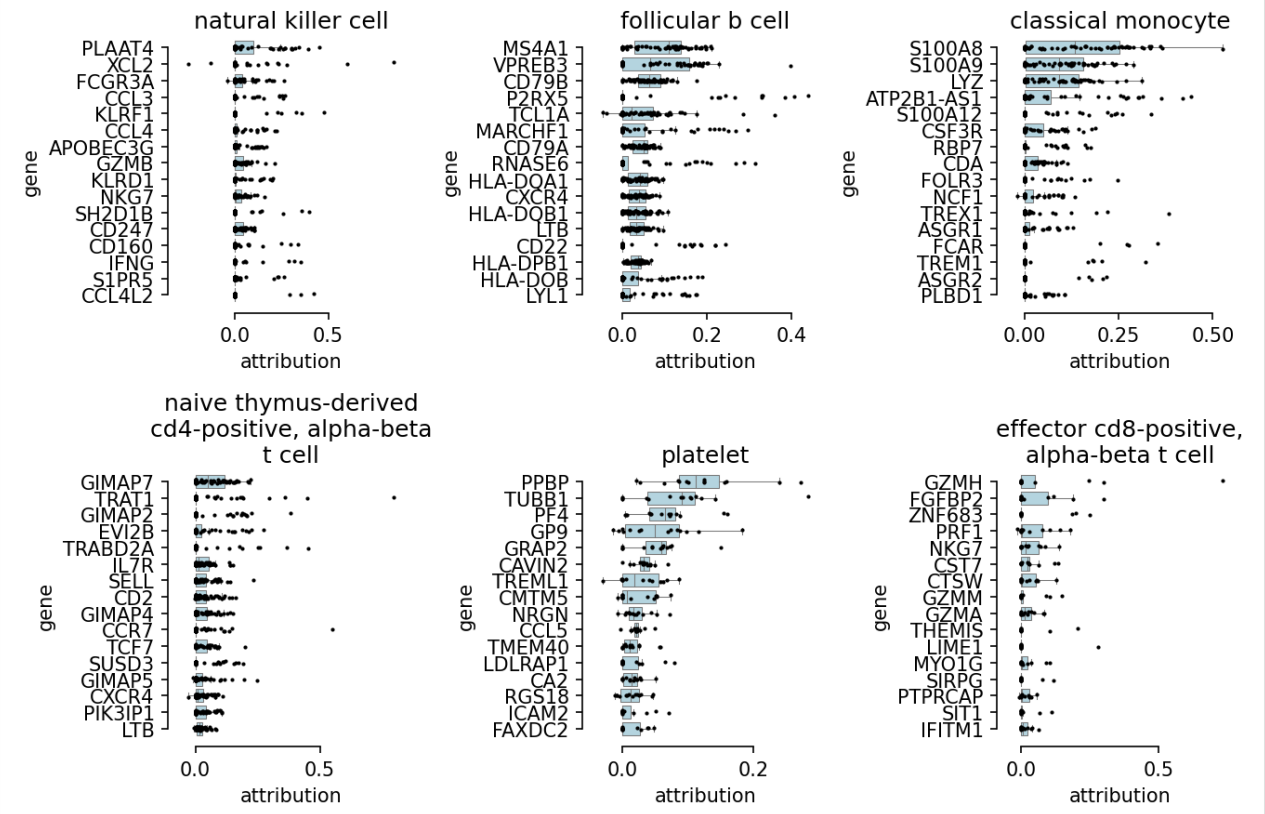
[73]:
```
<seaborn.matrix.ClusterGrid at 0x16c5659a0>
```



We can see from these results that UCDbase associates specific gene sets with a particular cell type annotations. They can also be used to verify the annotations being given by UCD by comparing them with known marker genes for varius cell types. For example we see *CD79B* as a strong attribution towards follicular b cell annotation, which is a well known b cell marker.

We can also plot boxplots for each celltype showing only the top N feature attributions for each type using the `ucd.pl.explain_boxplot` function.

[82]:
```
ucd.pl.explain_boxplot(adata, key = "ucdexplain", n_top_genes = 16, ncols = 3)
```



4.1.4.2.1 Compare Attribution Signatures for Different B Cell Subtypes

Something to note was that for our b cell cluster, the ucd raw annotations showed a distribution of probabilities across three differen subtypes, 'follicular b cells', 'naive b cell', and 'memory b cell'. Let's generate explanations for all three cell types and compare results.

Note: At this moment, ucdeconvolve only supports passing one cell type per feature attribution call per sample, so we will simply repeat the function call and append different key batches as results. In the future it will be possible to request predictions for multiple celltypes at once per sample. In the meantime, the plotting function `ucd.pl.explain_clustermap` supports viewing multiple keys simultaneously by passing a list of keys corresponding to different ucdexplain runs, presumably

---

for different cell types.

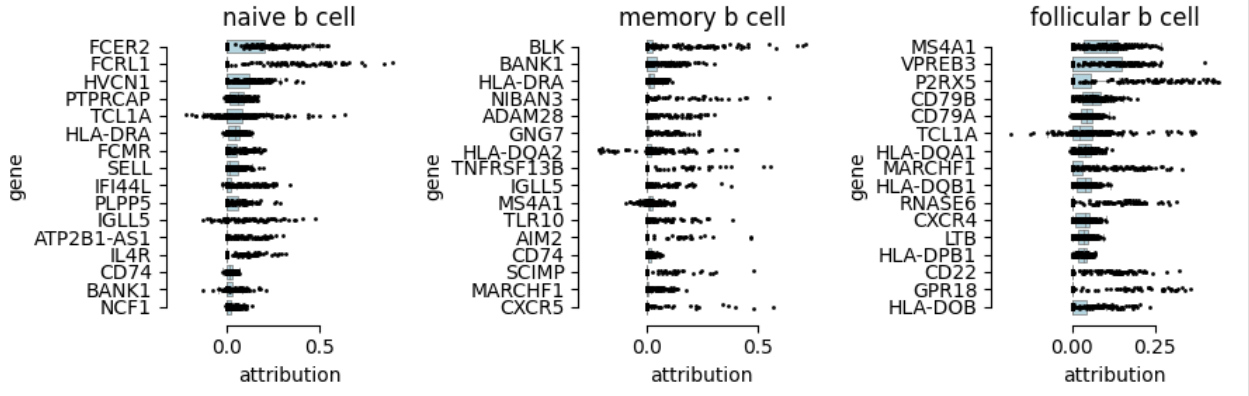
[65]:
```
adata_bcells = adata[adata.obs.leiden.eq("3")]
```

[ ]:
```
ucd.tl.explain(adata_bcells, celltypes = 'naive b cell', key_added="ucdexplain_naive_b")
ucd.tl.explain(adata_bcells, celltypes = 'memory b cell', key_added="ucdexplain_memory_b
↪")
ucd.tl.explain(adata_bcells, celltypes = 'follicular b cell', key_added="ucdexplain_
↪follicular_b")
```

Now let's go ahead and plot to view which genes are being used to drive different subtype predictions. These gene sets may be useful downstream in developing scoring signature or can be used for other applications such as gene set enrichment analysis or determining more fine-grained subclusters.

[76]:
```
import matplotlib.pyplot as plt
fig, axes = plt.subplots(ncols = 3, figsize = (9,3))

ucd.pl.explain_boxplot(adata_bcells, key = "ucdexplain_naive_b", ax = axes[0])
ucd.pl.explain_boxplot(adata_bcells, key = "ucdexplain_memory_b", ax = axes[1])
ucd.pl.explain_boxplot(adata_bcells, key = "ucdexplain_follicular_b", ax = axes[2])
```



4.1.5 Generating Contextualized Predictions with UCDSelect

Once we obtain a general overview of our dataset and understand what cell type categories different clusters belong to, we may want to perform higher-resolution, contextualized annotation. We can use UCDSelect to do this, which leverages a transfer learning regime utilizing UCDBase as a feature extraciton engine to calculate cell-type features for an input target dataset and an annotated reference dataset.

UCDSelect comes with pre-built reference datasets for common tissue types. To view datasets available as prebuilt references, run the utility function `ucd.utils.list_prebuilt_references()`.

Note: Would you like to have a particular study incorporated as a prebuilt reference? Email us at `ucdeconvolve@gmail.com` and let us know!

---

[83]:
```
ucd.utils.list_prebuilt_references()
```

[83]:
```
['allen-mouse-cortex', 'enge2017-human-pancreas', 'lee-human-pbmc-covid']
```

4.1.5.1 Running UCDSelect

Let's go ahead and run ucdselect using the *lee-human-pbmc-covid* reference, as both our target and this reference are PBMCs.

[85]:
```
ucd.tl.select(adata, "lee-human-pbmc-covid")
```

```
2023-04-25 16:05:15,042|[UCD]|INFO: Starting UCDeconvolveSELECT Run. | Timer Started.
Preprocessing Mix | 100% (11 of 11) |####| Elapsed Time: 0:00:01 Time:  0:00:01
Preprocessing Ref | 100% (1 of 1) |######| Elapsed Time: 0:00:00 Time:  0:00:00
2023-04-25 16:05:18,864|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 16:05:19,885|[UCD]|INFO: Upload Complete | Elapsed Time: 1.021 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | \ |#| 2 Elapsed Time: 0:00:03
Waiting For Submission : QUEUED | Queue Size : 1 | | |#| 3 Elapsed Time: 0:00:04
Waiting For Submission : RUNNING | Queue Size : 1 | | |#| 3 Elapsed Time: 0:00:04
Waiting For Completion | 100% (2700 of 2700) || Elapsed Time: 0:00:54 Time:  0:00:54
2023-04-25 16:06:52,212|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 16:06:52,779|[UCD]|INFO: Download Complete | Elapsed Time: 0.566 (s)
2023-04-25 16:06:53,483|[UCD]|INFO: Run Complete | Elapsed Time: 98.44 (s)
```

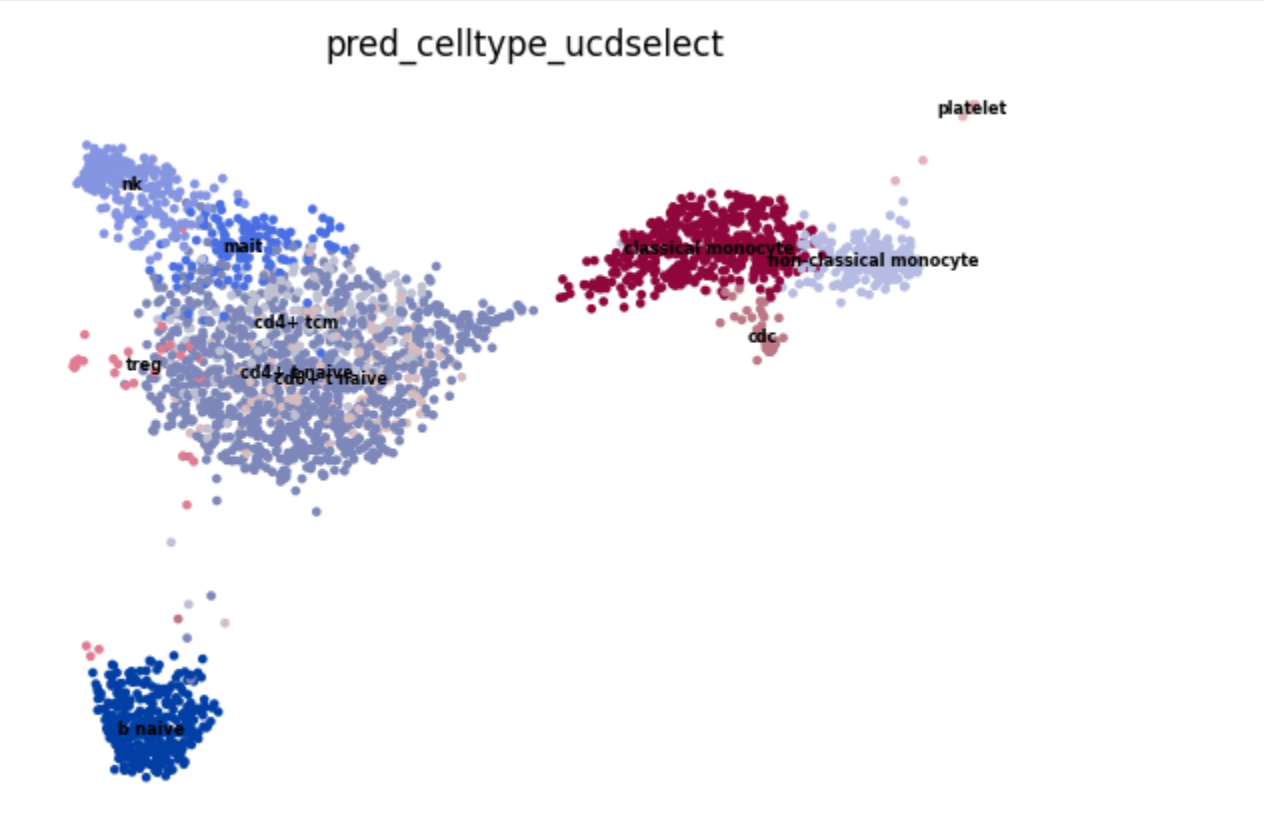
Now let's go ahead and assign these predictions to our cells. We will use a similar approach as we did with ucdbase, but this time we pass "ucdselect" as our results key to indicate this is the run we want to assign from. Additionally, we are going to increase our clustering resolution as we are working with a contextualized, high-resolution reference dataset.

Note: Cell types are best regarded as phenotypic "states" and as such exhibit a spectrum of variation. Assigning on the basis of a cluster ID represents an approximation that serves to reduces noise in annotations. Care should be taken when selecting a degree of clustering to use for cell type assignment.

---

[129]:
```
sc.tl.leiden(adata, resolution=3.0, key_added="leiden_hires")
ucd.utils.assign_top_celltypes(adata, "ucdselect", groupby = "leiden_hires")
```

[131]:
```
sc.pl.umap(adata, color = "pred_celltype_ucdselect", legend_loc = 'on data',
           legend_fontsize = 'xx-small', frameon = False)
```



[ ]:

[ ]:

4.2 Spatial Transcriptomics

In this tutorial we are going to run through how UCDeconvolve can be used to aid in the analysis and annotation of visium spatial transcriptomics data. For this tutorial, we will perform a cell type deconvolution of a spatial gene expression section of the human lymph node, made available by 10X Genomics. We will utilize scanpy to quickly load the dataset, and then pass it into ucdeconvolve to obtain cell type predictions.

4.2.1 Loading Packages & Authenticating

The first step in this analysis will be to load scanpy and ucdeconvolve after following the installation and registration instructions, and authenticate our API. In this tutorial we saved our user access token in the variable `TOKEN`.

[25]:
```
import scanpy as sc
import ucdeconvolve as ucd

ucd.api.authenticate(TOKEN)
```

```
2023-04-25 16:55:04,945|[UCD]|INFO: Updated valid user access token.
```

4.2.2 Read Lymph Node Dataset

We begin by loading the *V1_Human_Lymph_Node* dataset from *10X Genomics* made available in the `sc.datasets.visium_sge` utility function.

[ ]:
```
adata = sc.datasets.visium_sge("V1_Human_Lymph_Node")
```

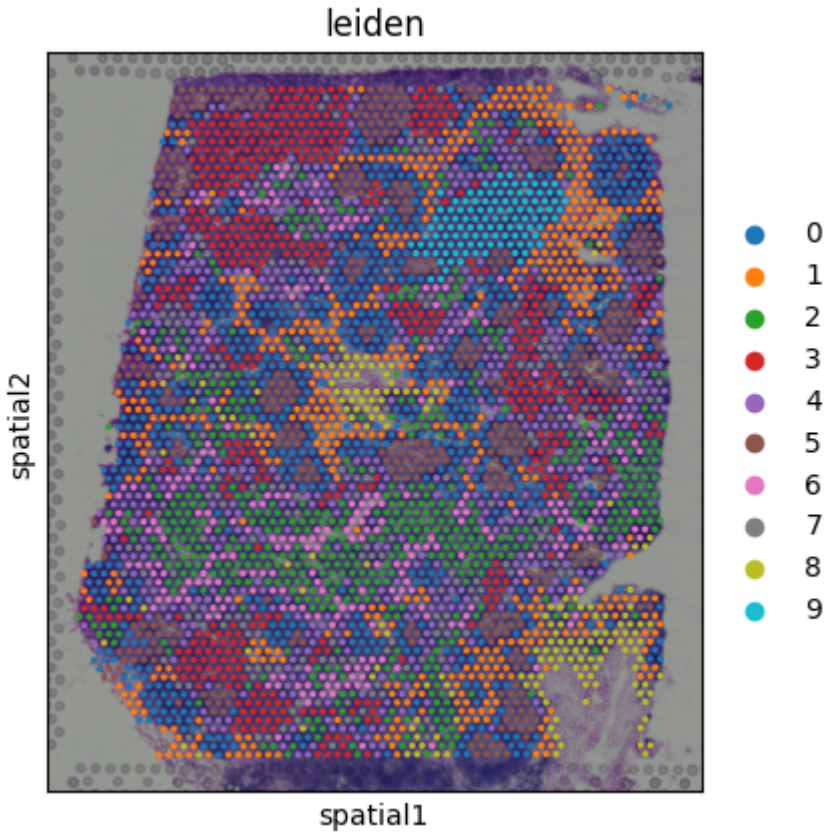
Let's perform some basic preprocessing on this dataset so we have some expression clusters to compare our downstream results with.

[9]:
```
adata.raw = adata

sc.pp.recipe_seurat(adata)

sc.pp.pca(adata)
sc.pp.neighbors(adata)
sc.tl.leiden(adata)
sc.tl.umap(adata)
```

[15]:
```
sc.pl.spatial(adata, color = 'leiden')
```



4.2.3 Run UCDBase

We begin by obtaining context-free cell type predictions using UCDBase.

[5]:
```
ucd.tl.base(adata)
```

```
2023-04-25 16:27:40,012|[UCD]|INFO: Starting UCDeconvolveBASE Run. | Timer Started.
Preprocessing Dataset | 100% (16 of 16) || Elapsed Time: 0:00:02 Time:  0:00:02
2023-04-25 16:27:43,509|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 16:27:49,367|[UCD]|INFO: Upload Complete | Elapsed Time: 5.857 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | \ |#| 2 Elapsed Time: 0:00:03
Waiting For Submission : QUEUED | Queue Size : 1 | | |#| 3 Elapsed Time: 0:00:04
Waiting For Submission : RUNNING | Queue Size : 1 | | |#| 3 Elapsed Time: 0:00:04
Waiting For Completion | 100% (4035 of 4035) || Elapsed Time: 0:00:45 Time:  0:00:45
2023-04-25 16:28:42,073|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 16:28:42,817|[UCD]|INFO: Download Complete | Elapsed Time: 0.743 (s)
2023-04-25 16:28:43,466|[UCD]|INFO: Run Complete | Elapsed Time: 63.453 (s)
```

4.2.4 Visualizing Results

We can print our adata object to see what new information has been added to it. UCD appends the results of each deconvolution run into 'adata.obsm' along with column names (i.e. celltypes) and run information into 'adata.uns' under the default results stem 'ucdbase'. Depending on whether or not the split parameter was set to True or False, you will either see a single new entry into 'adata.obsm' or three entries. By default, split = True so predictions will be split into primary (non-malignat), cell lines, and primary cancer (malignant). raw unsplit predictions are also saved.

[7]:
```
adata
```

[7]:
```
AnnData object with n_obs × n_vars = 4035 × 36601
    obs: 'in_tissue', 'array_row', 'array_col'
    var: 'gene_ids', 'feature_types', 'genome'
    uns: 'spatial', 'ucdbase'
    obsm: 'spatial', 'ucdbase_cancer', 'ucdbase_lines', 'ucdbase_primary', 'ucdbase_raw'
```

Let's start by reading our results in their raw, unpropagated form and see what top cell type predictions exist in this sample. Note that since this is a visium sample, these deconvolution represent actual predicted cell type mixtures and not individual cell phenotypes.

[35]:
```
ucd.utils.read_results(adata, category = 'raw').head(5)
```

[35]:
```
                    germinal center b cell  igg memory b cell  naive t cell
AAACAAGTATCTCCCA-1                0.107994           0.118136      0.079384  \
AAACAATCTACTAGCA-1                0.210373           0.019785      0.241911
AAACACCAATAACTGC-1                0.114059           0.096963      0.018223
AAACAGAGCGACTCCT-1                0.348411           0.028202      0.009421
AAACAGCTTTCAGAAG-1                0.100210           0.118197      0.040606

                    common dendritic progenitor  follicular b cell
AAACAAGTATCTCCCA-1                     0.094625           0.046563  \
AAACAATCTACTAGCA-1                     0.081217           0.008128
AAACACCAATAACTGC-1                     0.063723           0.031963
AAACAGAGCGACTCCT-1                     0.019920           0.069834
AAACAGCTTTCAGAAG-1                     0.136658           0.181446
```

(continues on next page)

(continued from previous page)

```
                    immature b cell  effector cd4-positive, alpha-beta t cell
AAACAAGTATCTCCCA-1        0.059231                                    0.031747  \
AAACAATCTACTAGCA-1        0.007391                                    0.027828
AAACACCAATAACTGC-1        0.034795                                    0.007577
AAACAGAGCGACTCCT-1        0.010529                                    0.000306
AAACAGCTTTCAGAAG-1        0.041983                                    0.018380

                    endothelial cell of lymphatic vessel    t cell
AAACAAGTATCTCCCA-1                              0.020969  0.022492  \
AAACAATCTACTAGCA-1                              0.007141  0.004119
AAACACCAATAACTGC-1                              0.110919  0.042955
AAACAGAGCGACTCCT-1                              0.003792  0.289740
AAACAGCTTTCAGAAG-1                              0.011223  0.005778

                    naive thymus-derived cd4-positive, alpha-beta t cell  ...
AAACAAGTATCTCCCA-1                                           0.006955     ...  \
AAACAATCTACTAGCA-1                                           0.214080     ...
AAACACCAATAACTGC-1                                           0.000502     ...
AAACAGAGCGACTCCT-1                                           0.000523     ...
AAACAGCTTTCAGAAG-1                                           0.005383     ...

                    oogonial cell  contractile cell  polar body
AAACAAGTATCTCCCA-1            0.0                0.0         0.0  \
AAACAATCTACTAGCA-1            0.0                0.0         0.0
AAACACCAATAACTGC-1            0.0                0.0         0.0
AAACAGAGCGACTCCT-1            0.0                0.0         0.0
AAACAGCTTTCAGAAG-1            0.0                0.0         0.0

                    natural killer cell cell  b-1a b cell
AAACAAGTATCTCCCA-1                       0.0          0.0  \
AAACAATCTACTAGCA-1                       0.0          0.0
AAACACCAATAACTGC-1                       0.0          0.0
AAACAGAGCGACTCCT-1                       0.0          0.0
AAACAGCTTTCAGAAG-1                       0.0          0.0

                    supporting cell of cochlea  bone marrow cell
AAACAAGTATCTCCCA-1                         0.0               0.0  \
AAACAATCTACTAGCA-1                         0.0               0.0
AAACACCAATAACTGC-1                         0.0               0.0
AAACAGAGCGACTCCT-1                         0.0               0.0
AAACAGCTTTCAGAAG-1                         0.0               0.0

                    pre-natural killer cell  kidney granular cell
AAACAAGTATCTCCCA-1                      0.0                   0.0  \
AAACAATCTACTAGCA-1                      0.0                   0.0
AAACACCAATAACTGC-1                      0.0                   0.0
AAACAGAGCGACTCCT-1                      0.0                   0.0
AAACAGCTTTCAGAAG-1                      0.0                   0.0

                    splenic red pulp macrophagec
AAACAAGTATCTCCCA-1                           0.0
AAACAATCTACTAGCA-1                           0.0
```

(continues on next page)

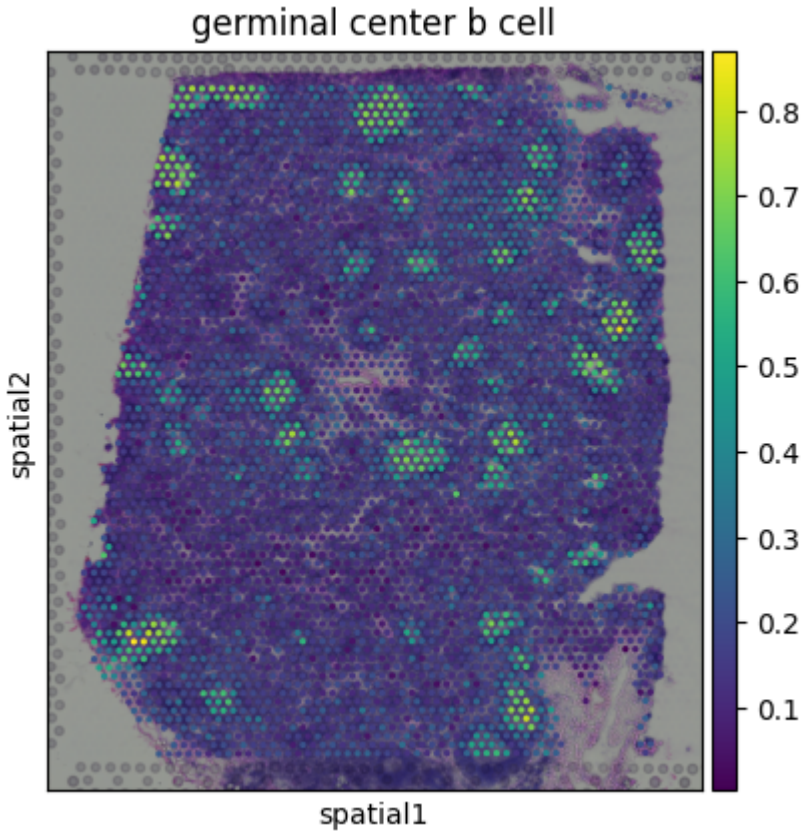
(continued from previous page)

```
AAACACCAATAACTGC-1                    0.0
AAACAGAGCGACTCCT-1                    0.0
AAACAGCTTTCAGAAG-1                    0.0

[5 rows x 841 columns]
```

We can visualize our results by using one of the built-in plotting functions in UCD, which wrap scanpy's plotting API.

[28]:
```
ucd.pl.spatial(adata, color = "germinal center b cell")
```

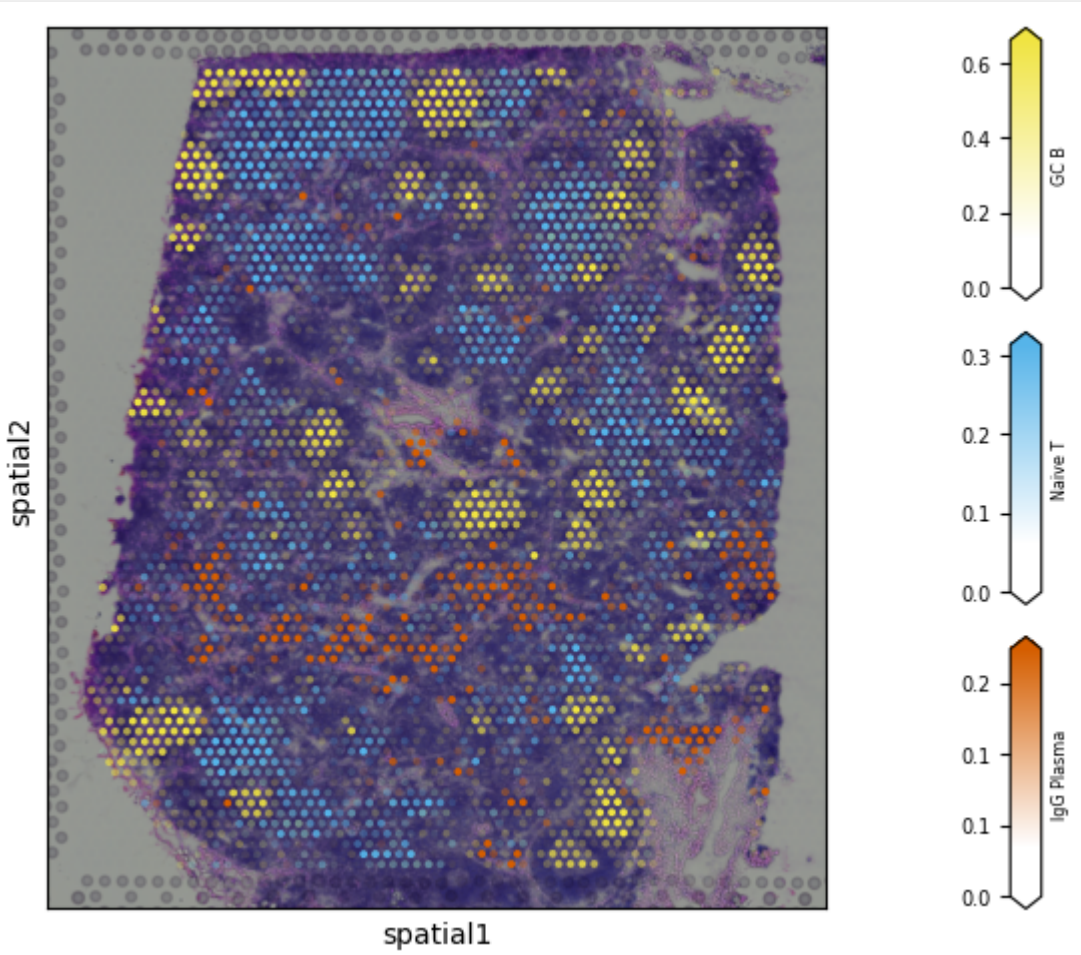


4.2.4.1 Advanced Visualization

Many times it can be useful to plot multiple cell type densities on the same spatial plot. To do this, `ucd.pl.spatial` extends the functionality of scanpy's spatial plotting functions to allow us to plot multiple cell type predictions. We offer a set of colormaps under `ucd.pl.CM` inspired by cell2location [Kleshchevnikov et. al. 2022] that can be used as overlay colors.

[34]:
```
ucd.pl.spatial(adata,
    color = ["germinal center b cell", "naive t cell", "igg plasma cell"],
    labels = ['GC B', "Naive T", "IgG Plasma"],
    colormaps = [ucd.pl.CM.Yellow, ucd.pl.CM.Blue, ucd.pl.CM.Orange],
    cbar_nrows=3
)
```

[34]:
```
<Axes: xlabel='spatial1', ylabel='spatial2'>
```



[ ]:

4.3 Bulk RNA-Seq

In this tutorial we are going to run through how UCDeconvolve can be used to aid in the analysis and annotation of bulk RNA-Sequencing data. For this tutorial, we are going to recreate part of the analysis in Figure 5 of the unicell deconvolve manuscript, where we analyze bulk-RNA-Seq data from patients at various stages of Type 2 Diabetes from a prior study `GSE50244`.

4.3.1 Loading Packages & Authenticating

The first step in this analysis will be to load scanpy and ucdeconvolve after following the installation and registration instructions, and authenticate our API. In this tutorial we saved our user access token in the variable `TOKEN`.

[17]:
```
import matplotlib.pyplot as plt
import scanpy as sc
import seaborn as sns
import requests
import io

import ucdeconvolve as ucd
ucd.api.authenticate(TOKEN)
```

```
2023-04-25 17:23:16,217|[UCD]|INFO: Updated valid user access token.
```

4.3.2 Download and Pre-Process Data

We will download a preprocessed version of the target dataset from an online repository. This dataset contains expression data from pancreatic islet biopsies from ~100 patients at varying stages of T2D.

[3]:
```
url = "https://github.com/dchary/ucdeconvolve_paper/raw/main/figure5/adata_t2d_GSE50244.
↪h5ad"
adata = sc.read_h5ad(io.BytesIO(requests.get(url).content))
```

4.3.3 Run UCDBase For Context-Free Prediction

We begin by obtaining context-free cell type predictions using UCDBase.

[5]:
```
ucd.tl.base(adata)
```

```
2023-04-25 17:19:09,360|[UCD]|INFO: Starting UCDeconvolveBASE Run. | Timer Started.
Preprocessing Dataset | 100% (1 of 1) |##| Elapsed Time: 0:00:00 Time:  0:00:00
2023-04-25 17:19:10,062|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 17:19:10,917|[UCD]|INFO: Upload Complete | Elapsed Time: 0.854 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | \ |#| 2 Elapsed Time: 0:00:03
Waiting For Submission : QUEUED | Queue Size : 1 | / |#| 4 Elapsed Time: 0:00:06
Waiting For Submission : RUNNING | Queue Size : 1 | | |#| 4 Elapsed Time: 0:00:06
Waiting For Completion | 100% (77 of 77) || Elapsed Time: 0:00:14 Time:  0:00:14
2023-04-25 17:19:33,178|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 17:19:33,417|[UCD]|INFO: Download Complete | Elapsed Time: 0.239 (s)
2023-04-25 17:19:34,084|[UCD]|INFO: Run Complete | Elapsed Time: 24.723 (s)
```

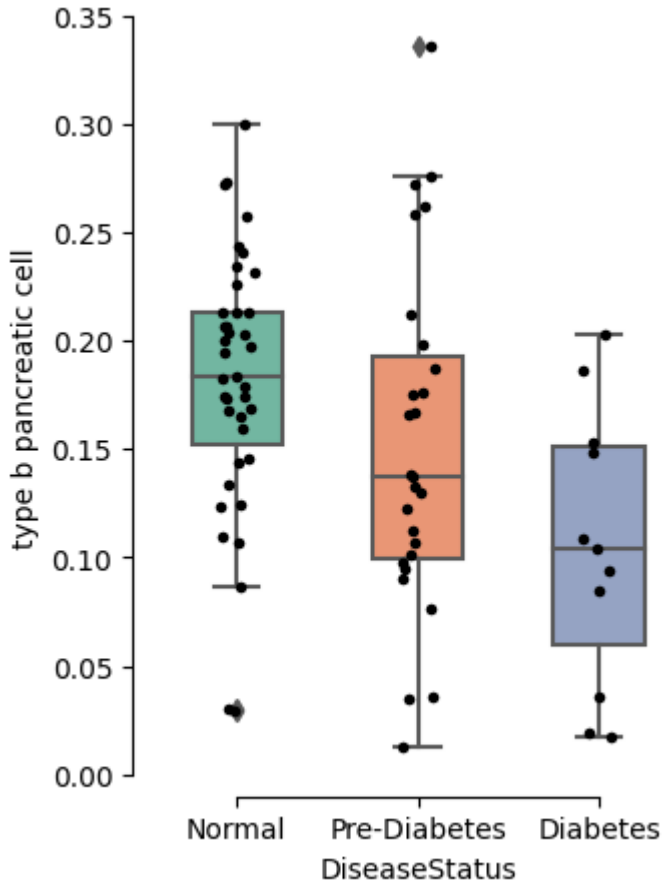
4.3.3.1 Examine Differences in Pancreatic Beta Cell Fractions Across Disease States

We want to look at differences in cell type fractions between patients stratified by disease state. Late-stage T2D is characterized by a loss of beta cells. Let's see if this dataset is concordant with this hypothesis.

[34]:
```
# Load only beta cell predictions
preds = ucd.utils.read_results(adata, celltypes = ['type b pancreatic cell'])

# Append to adata as a column
adata.obs['type b pancreatic cell'] = preds['type b pancreatic cell']

# Plot boxplots
fig, ax = plt.subplots(figsize = (3.5,5))
sns.boxplot(data = adata.obs, x = 'DiseaseStatus', y = 'type b pancreatic cell',
            order = ['Normal', 'Pre-Diabetes', 'Diabetes'], ax = ax,
            palette="Set2", width = 0.5)
sns.stripplot(data = adata.obs, x = 'DiseaseStatus', y = 'type b pancreatic cell',
            order = ['Normal', 'Pre-Diabetes', 'Diabetes'], ax = ax, color = 'k', size =␣
↪4)
sns.despine(ax = ax, trim = True, offset = 5)
```



4.3.4 Run UCDSelect For Contextualized Predictions

We may want to leverage a pancreas-specific reference dataset to explore more detail subtypes, but also to confirm the findings suggested by our context-free approach. We can use UCDSelect to do this, which leverages a transfer learning regime utilizing UCDBase as a feature extraciton engine to calculate cell-type features for an input target dataset and an annotated reference dataset.

UCDSelect comes with pre-built reference datasets for common tissue types. To view datasets available as prebuilt references, run the utility function `ucd.utils.list_prebuilt_references()`.

Note: Would you like to have a particular study incorporated as a prebuilt reference? Email us at `ucdeconvolve@gmail.com` and let us know!

---

[35]:
```
ucd.utils.list_prebuilt_references()
```

[35]:
```
['allen-mouse-cortex', 'enge2017-human-pancreas', 'lee-human-pbmc-covid']
```

4.3.4.1 Running UCDSelect

Let's go ahead and run ucdselect using the *enge2017-human-pancreas* reference.

[36]:
```
ucd.tl.select(adata, "enge2017-human-pancreas")
```

```
2023-04-25 17:28:26,171|[UCD]|INFO: Starting UCDeconvolveSELECT Run. | Timer Started.
Preprocessing Mix | 100% (1 of 1) |######| Elapsed Time: 0:00:00 Time:  0:00:00
Preprocessing Ref | 100% (1 of 1) |######| Elapsed Time: 0:00:00 Time:  0:00:00
2023-04-25 17:28:27,651|[UCD]|INFO: Uploading Data | Timer Started.
2023-04-25 17:28:28,784|[UCD]|INFO: Upload Complete | Elapsed Time: 1.132 (s)
Waiting For Submission : UNKNOWN | Queue Size : 0 | / |#| 0 Elapsed Time: 0:00:00
Waiting For Submission : QUEUED | Queue Size : 1 | - |#| 1 Elapsed Time: 0:00:01
Waiting For Submission : RUNNING | Queue Size : 1 | | |#| 1 Elapsed Time: 0:00:01
Waiting For Completion | 100% (77 of 77) || Elapsed Time: 0:00:22 Time:  0:00:22
2023-04-25 17:29:12,832|[UCD]|INFO: Download Results | Timer Started.
2023-04-25 17:29:12,958|[UCD]|INFO: Download Complete | Elapsed Time: 0.125 (s)
2023-04-25 17:29:13,621|[UCD]|INFO: Run Complete | Elapsed Time: 47.449 (s)
```

Let's go ahead and read our results, where we see only pancreas-specific cell types being deconvoled which were annotated in the original reference.

[38]:
```
ucd.utils.read_results(adata, key = 'ucdselect').head(5)
```

[38]:

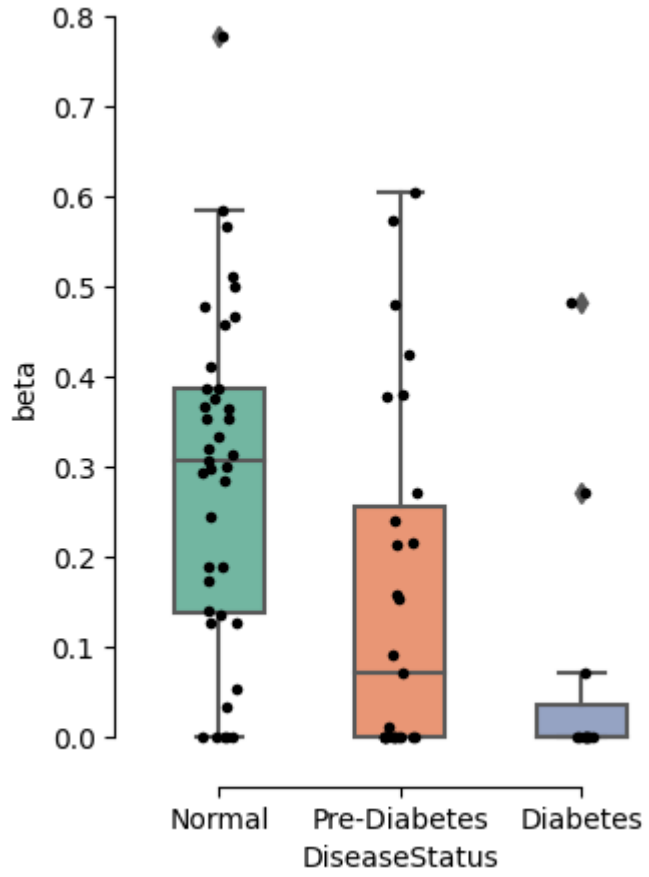
sample_id	beta	alpha	ductal	delta	acinar	mesenchymal
GSM1216753	0.000859	0.048488	0.704008	0.119697	0.126949	0.000000
GSM1216755	0.126114	0.124226	0.000000	0.000000	0.710946	0.038713
GSM1216758	0.365763	0.102278	0.000000	0.000000	0.000000	0.531958
GSM1216760	0.566138	0.015259	0.000000	0.000000	0.207783	0.210820
GSM1216763	0.000000	0.066785	0.125175	0.000000	0.396808	0.411232

Let's repeat our boxplot plotting and look at results for type b pancreatic cells when using a contextualized reference for deconvolution with fine-tuning.

[58]:
```
# Load only beta cell predictions
preds = ucd.utils.read_results(adata, key = 'ucdselect', celltypes = ['beta'])

# Append to adata as a column
adata.obs['beta'] = preds['beta']

# Plot boxplots
fig, ax = plt.subplots(figsize = (3.5,5))
sns.boxplot(data = adata.obs, x = 'DiseaseStatus', y = 'beta',
            order = ['Normal', 'Pre-Diabetes', 'Diabetes'], ax = ax,
            palette="Set2", width = 0.5)
sns.stripplot(data = adata.obs, x = 'DiseaseStatus', y = 'beta',
            order = ['Normal', 'Pre-Diabetes', 'Diabetes'], ax = ax, color = 'k', size =␣
↪4)
sns.despine(ax = ax, trim = True, offset = 5)
```



We can see that islet samples in the Diabetes cohort exhibit almost no detectable beta cells, which patients who are otherwise healthy report a predicted beta cell median fraction of ~30%. Note that while the trend of relative proportions between ucdbase and ucdselect is the same, the absolute proportions were different. Healthy islets are often ~70% beta cells, however these bulk islet samples likely represent imperfect dissections, with ~50% of the tissue predicted to represent pancreatical acinar / ductal epithelial cells and mesencyhmal stroma.

CHAPTER

FIVE

RELEASE NOTES

5.1 0.1.0 - 2023-04-17

5.1.1 Overview

This is a major update to the UCDeconvolve API. This update features an overhauled user registration system that enables independent user registration, verification, and API key management. Core API services have been expanded to include, in addition to UCDBase, UCDExplain, and UCDSelect fine-tuning services. The API is now designated as feature-complete, and has been upgraded to beta status.

Note: Please note that significant development work is still underway, and aspects of this software may be changed at any time without notice.

See details on major updates below:

5.1.2 Central API Server

All user management, job requests, progress, and results are managed by a central API server that serves as authentication and communication layer between end-users and backend prediction services.

5.1.3 User Registration / Authentication

Users are now free to register to receive an API key independently and will immediately receive API keys upon activation of their accounts. The entire registration process is now integrated into the core API and can be done programmatically. See tutorials for how registration works in additional detail.

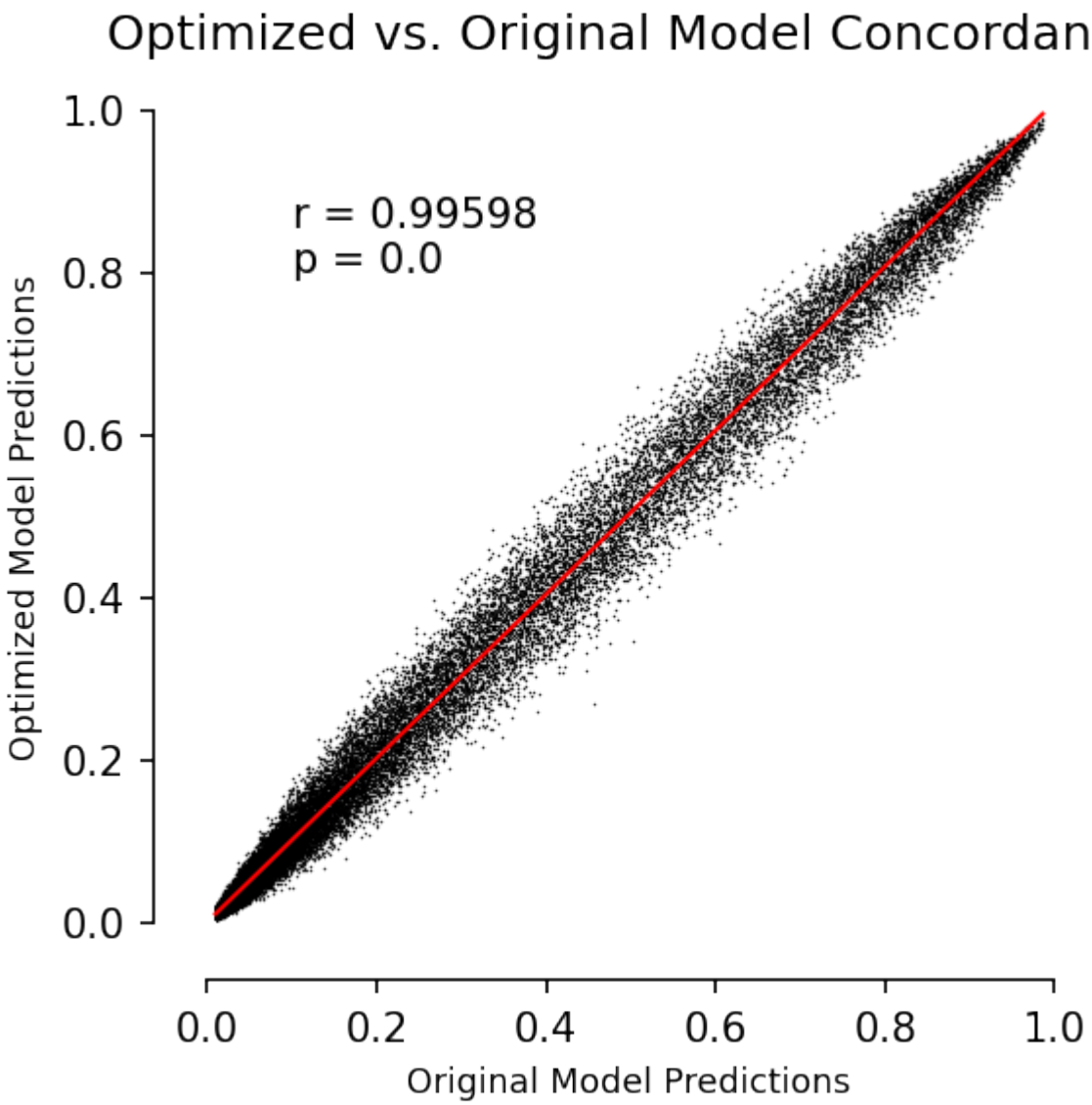
5.1.4 Integration of UCDExplain and UCDSelect

Full functionality of UCDExplain and UDCSelect fine-tuning is now available. UCDExplain leverages integrated gradients to return feature attributions associated with prediction of a given cell type from the UCDBase model with a given input (i.e. gene). UCDSelect allows transfer learning of UCDBase embedding weights to enable contextualized deconvolution results when combined with an appropriate reference dataset. Prebuilt references have been made available for access, and dozens more are planned to be added in due time. For details on using UCDExplain and UCDSelect, see tutorials.

5.1.5 Improvements to UCDBase

Performance of the existing UCDBase context-free deconvolution function has been improved with two key changes.

1. Data submitted for deconvolution is preprocessed client-side and uploaded to the central API server prior to initiation of prediction, which is in contrast to prior approach which involve streaming batch chunks directly and waiting for batch prediction responses. This approach offers improvements in performance and reliability. No user data is kept, and all submitted data is deleted immediately following completion of a prediction job.
2. We have redeployed UCDBase using the ONNX ML platform. The base model used for prediction is a performance-optimized derivative of the original UCDBase model, which offers superior performance for large datasets. We show a comparison of correlations for original vs. optimized model performance and note high concordance:



5.1.6 Improvements to Utilities / Plotting

Numerous new plotting functions and utilities have been added to streamline the UCDeconvolve prediction process. These include new methods for generating visualizations of spatial deconvolution and feature-attribution plots as shown in the original manuscript. For more details, see updated tutorials section.

5.1.7 Under the Hood Improvements

Many more improvements were made to improve the usability, stability, and performance of the underlying UCD package, that lay a foundation for continued improvement over time.

5.2 0.0.1 - 2022-08-04

5.2.1 Overview

Initial release.

CHAPTER

SIX

METHOD OVERVIEW

A little about UCD.

PYTHON MODULE INDEX

u

INDEX