CRISPR Jamboree 2024: Inference

Gene Katsevich

January 26, 2024

1 Overview of datasets, inputs, and outputs

I propose the following for the inference module:

- **Datasets:** We should test on at least one high-MOI and one low-MOI dataset. I propose we try subsets of the Gasperini (high-MOI) and Papalexi (low-MOI) datasets, which are built into the `sceptre` package. These datasets are available on GitHub.
- **Input:** The input will be a `MuData` object with a certain set of minimum required fields (see more below), the sidedness of the test, and any additional arguments to the analysis method.
- **Output:** The output will be an updated `MuData` object, with the *p*-values and log fold-change estimates added in the `uns` field.

2 `MuData` structure

We will need to specify precisely how the `MuData` object should be formatted. The current `MuData` formatting guidelines do not specify naming conventions precisely enough, and which fields are mandatory versus which fields are optional. Lucas's sample Gasperini `MuData` object is a good starting point, but it has the following limitations:

- It does not include negative control gRNAs.
- It does not include the gRNA-to-cell assignments.
- It does not include the data frame for the results of an analysis method.
- Certain naming choices conflict with the `MuData` formatting guidelines document.

In what follows, I propose an example of a `MuData` object structure that addresses some of the above limitations. To illustrate, I will use a subset of the Gasperini dataset. Note that this subset is different from Lucas's. For example, it contains non-targeting gRNAs. The `MuData` structure itself is inspired by Lucas's but includes some additions, such as the gRNA-to-cell assignments and the data frame for the results of an analysis method.

```
import mudata as md
import pandas as pd
gasperini_data = md.read_h5mu("data/gasperini_subset.h5mu")
gasperini_data
```

```
## MuData object with n_obs × n_vars = 44308 × 716
##   obs:   'batch'
##   uns:   'inference_results', 'moi'
##   3 modalities
##     gene:    44308 x 526
##       var:   'symbol'
##     guide:   44308 x 95
##       var:   'targeting', 'guide_target', 'guide_chr', 'guide_start', 'guide_end'
##     guide_assignment:    44308 x 95
```

We can see that this object has three modalities, named `gene` (for gene expression), `guide` (for gRNA expression), and `guide_assignment` (for binary gRNA-to-cell assignments). We need to make sure these names are standardized.

2.1 Gene expression modality

The gene expression modality (`gene`) should contain the raw RNA UMI counts; we will leave any normalization to each of the individual analysis methods. The variable names are the ENSEMBL gene IDs:

```
gasperini_data['gene'].var_names[:5].tolist()
```

```
## ['ENSG00000069275', 'ENSG00000117222', 'ENSG00000117266', 'ENSG00000117280', 'ENSG00000133059']
```

The gene names can be optionally be provided in a variable called `symbol`:

```
gasperini_data['gene'].var
```

```
##                     symbol
## ENSG00000069275   NUCKS1
## ENSG00000117222    RBBP5
## ENSG00000117266    CDK18
## ENSG00000117280    RAB29
## ENSG00000133059    DSTYK
## ...                  ...
## ENSG00000155380  SLC16A1
## ENSG00000196683    TOMM7
## ENSG00000176890     TYMS
## ENSG00000198786   MT-ND5
## ENSG00000198840   MT-ND3
##
## [526 rows x 1 columns]
```

We need to choose a standardized name for the variable storing the gene names; I chose `symbol` because this is what DACC appears to use.

2.2 gRNA expression modality

The gRNA expression modality (`guide`) should contain the raw gRNA UMI counts; we will leave any normalization to each of the individual analysis methods. Within the `guide` modality, the variable names are the gRNA IDs:

```
gasperini_data['guide'].var_names[:5].tolist()
```

```
## ['grna_CCGGGCG', 'grna_TGGCGGC', 'grna_AAGGCCG', 'grna_GACGCCG', 'grna_CACACCC']
```

The variables in this modality are similar to the per-guide metadata format we developed:

```
gasperini_data['guide'].var
```

```
##               targeting     guide_target guide_chr  guide_start   guide_end
## grna_CCGGGCG          1  ENSG00000069482     chr11   68451943.0  68451958.0
## grna_TGGCGGC          1  ENSG00000069482     chr11   68451958.0  68451974.0
## grna_AAGGCCG          1  ENSG00000100316     chr22   39715775.0  39715790.0
## grna_GACGCCG          1  ENSG00000100316     chr22   39715790.0  39715806.0
## grna_CACACCC          1  ENSG00000104131     chr15   44829255.0  44829270.0
## ...                 ...              ...       ...          ...         ...
## grna_ATTAGCA          0    non-targeting                   -9.0        -9.0
## grna_AGATACC          0    non-targeting                   -9.0        -9.0
## grna_ATATGTA          0    non-targeting                   -9.0        -9.0
## grna_GTAGCCT          0    non-targeting                        -9.0        -9.0
## grna_TTAGCTT          0    non-targeting                        -9.0        -9.0
##
## [95 rows x 5 columns]
```

First, we must specify for each gRNA whether it is `targeting`. (Here this boolean variable shows up as 0/1; probably we should change this.) Next, we must specify for each gRNA what exactly it targets (which gene, which putative enhancer, etc). This is probably similar to `intended_target_name` in the per-guide metadata format but I wasn't sure, so I named it `guide_target` (we should come to a consensus on this). The reason this is required is because we it is most meaningful to test for associations between *targeted elements* and gene expression rather than between *individual gRNAs* and gene expression. Therefore, we must know which guides target the same element. We should have a reserved string for the `guide_target` for non-targeting guides; I propose `non-targeting`. Optionally, we can specify the genomic coordinates of the region targeted by the guide. We should have standard placeholder values genomic coordinates of non-targeting guides. I propose the empty string for chromosome and `-9` for start and end coordinates.

2.3 gRNA assignment modality

The gRNA assignment modality (`guide_assignment`) should contain the binary gRNA-to-cell assignments. The variable names are again the gRNA IDs:

```
gasperini_data['guide_assignment'].var_names[:5].tolist()
```

```
## ['grna_CCGGGCG', 'grna_TGGCGGC', 'grna_AAGGCCG', 'grna_GACGCCG', 'grna_CACACCC']
```

There are no required `vars` for this modality, because the relevant metadata are already in the `guide` modality.

2.4 Other fields

There are two other fields of the `gasperini_data` object, `obs` and `uns`. The `obs` field contains cell metadata not specific to any modality. The most important such metadata is `batch`.

```
gasperini_data.obs
```

```
##            batch
## cell_1        b1
## cell_2        b1
## cell_3        b1
## cell_4        b1
## cell_5        b1
## ...          ...
## cell_44304    b2
## cell_44305    b2
## cell_44306    b2
## cell_44307    b2
## cell_44308    b2
##
## [44308 rows x 1 columns]
```

I propose we have a required variable called `batch` that specifies the batch for each cell. Even if the data only has one batch, we can have a variable with just one value. The other field is `uns`, which contains unstructured metadata. I have included two fields in `uns`: `moi` and `inference_results`. The `moi` field specifies the MOI of the experiment (`high` or `low`):

```
gasperini_data.uns['moi'][0]
```

```
## 'high'
```

I propose for this to be a mandatory field. The `inference_results` field is where the results of the inference will be stored:

```
pd.set_option('display.max_columns', None)
pd.DataFrame(gasperini_data.uns['inference_results'])
```

```
##             gene_id      grna_target  log_2_FC  p_value        pair_type
## 0    ENSG00000069482  ENSG00000069482      -9.0     -9.0  positive_control
## 1    ENSG00000100316  ENSG00000100316      -9.0     -9.0  positive_control
## 2    ENSG00000104131  ENSG00000104131      -9.0     -9.0  positive_control
## 3    ENSG00000122026  ENSG00000122026      -9.0     -9.0  positive_control
## 4    ENSG00000135821  ENSG00000135821      -9.0     -9.0  positive_control
## ..               ...              ...       ...      ...               ...
## 615  ENSG00000131094  candidate_enh_8      -9.0     -9.0         discovery
## 616  ENSG00000136448  candidate_enh_8      -9.0     -9.0         discovery
## 617  ENSG00000172992  candidate_enh_8      -9.0     -9.0         discovery
## 618  ENSG00000181513  candidate_enh_8      -9.0     -9.0         discovery
## 619  ENSG00000161714  candidate_enh_8      -9.0     -9.0         discovery
##
## [620 rows x 5 columns]
```

The columns `gene_id` and `grna_target` specify what pairs of gene and targeted element to test for association. The columns `log_2_FC` and `p_value` will be filled in by the inference method, and should be initialized with placeholders such as `-9`. Finally, the optional column `pair_type` specifies the type of pair being tested, e.g. positive control or discovery pair.

3 Items for discussion

1. We need to work together to settle on the precise specification of the `MuData` format, not just for the inference task but also for upstream tasks like gRNA assignment. We might want to write Python and/or R functions that check whether a given `MuData` object conforms to whatever specification we end up deciding on.

2. Do we agree that we should test for association between *targeted elements* and gene expression rather than between *individual gRNAs* and gene expression? Should we also have an option to test the latter? I think the latter is adding unnecessary complication at this stage.

3. The current `MuData` structure does not include any cell-wise covariates or QC metrics. We have discussed that there are certain cell-wise covariates that most methods would want to use, like library size. However, adding covariates to the `MuData` will require us to standardize more field names, and different methods might want to use different covariates. Therefore, I thought we could just leave it up to the individual methods to compute whichever covariates they require.