scedar Documentation

Release 0.2.0

Yuanchao Zhang

Mar 16, 2020

Contents

Scedar (Single-cell exploratory data analysis for RNA-Seq) is a reliable and easy-to-use Python package for efficient visualization, imputation of gene dropouts, detection of rare transcriptomic profiles, and clustering of large-scale single cell RNA-seq (scRNA-seq) datasets.

CHAPTER 1

Demo

Preparation in 12s

```
import numpy as np
import pandas as pd
import scedar as sce
count_df = pd.read_csv(
  'zeisel_counts.csv', index_col=0)
cell_types = pd.read_csv(
  'zeisel_ctype.csv', index_col=0)
sdm = sce.eda.SampleDistanceMatrix(
  count_df.values.T,
  metric='cosine', nprocs=25)
```

Find cluster separating genes in 3min

```
cmp_labs = [1, 15, 22]
```

Workflow of using `scedar` to analyze an scRNA-seq dataset with 3005 mouse brain cells and 19,972 genes generated using the STRT-Seq UMI protocol by Zeisel et al. (2015). Procedures and parameters that are not directly related to data analysis are omitted. The full version of the demo is available at the *Tutorial* section.

Data sources:

- Zeisel, A., Muñoz-Manchado, A. B., Codeluppi, S., Lönnerberg, P., La Manno, G., Juréus, A., Marques, S., Munguba, H., He, L., Betsholtz, C., Rolny, C., Castelo-Branco, G., Hjerling-Leffler, J., and Linnarsson, S. (2015). Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq . *Science*, **347** (6226), 1138–1142.
- Hemberg Group scRNA-seq datasets

CHAPTER 2

Install

Use PyPI:

```
pip install scedar
```

2.1 Install

Use PyPI:

```
pip install scedar
```

Installation has been tested on Linux and MacOS.

Installing nmslib from source may be able to optimize its performance on your computer. Run `pip install --no-binary :all: nmslib` prior to installing scedar.

2.2 Tutorial

[1]:
```
import scedar as sce
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import os
from collections import namedtuple
from collections import Counter
```

[2]:
```
# plt.style.use('dark_background')
plt.style.use('default')
%matplotlib inline
```

2.2.1 Read data

Zeisel: STRT-Seq. Mouse brain.

[3]:
```
RealDataset = namedtuple('RealDataset',
                         ['id', 'x', 'coldata'])
```

[4]:
```
%%time
zeisel = RealDataset(
    'Zeisel',
    pd.read_csv('data/real_csvs/zeisel_counts.csv', index_col=0),
    pd.read_csv('data/real_csvs/zeisel_coldata.csv', index_col=0))
```

```
CPU times: user 7.3 s, sys: 873 ms, total: 8.17 s
Wall time: 8.21 s
```

2.2.2 Exploratory Data Analysis

Check dimensions and data

[5]:
```
zeisel.x.shape
```

[5]:
```
(19972, 3005)
```

[6]:
```
zeisel.x.iloc[:3, :3]
```

[6]:
```
          X1  X1.1  X1.2
Tspan12    0     0     0
Tshz1      3     1     0
Fnbp1l     3     1     6
```

[7]:
```
zeisel.coldata.head()
```

[7]:
```
      clust_id    cell_type1  total_features  log10_total_features  \
X1           1  interneurons            4848              3.685652
X1.1         1  interneurons            4685              3.670802
X1.2         1  interneurons            6028              3.780245
X1.3         1  interneurons            5824              3.765296
X1.4         1  interneurons            4701              3.672283

      total_counts  log10_total_counts  pct_counts_top_50_features  \
X1           21580            4.334072                   19.819277
X1.1         21748            4.337439                   19.293728
X1.2         31642            4.500278                   17.423045
X1.3         32914            4.517394                   19.593486
X1.4         21530            4.333064                   15.397120

      pct_counts_top_100_features  pct_counts_top_200_features  \
X1                      25.949954                    34.147359
X1.1                    26.011587                    34.715836
X1.2                    23.487769                    31.369699
X1.3                    26.122623                    34.687367
X1.4                    21.634928                    30.380864

      pct_counts_top_500_features  ...  log10_total_features_feature_control  \
X1                      48.535681  ...                                     0
```

(continues on next page)

(continued from previous page)

```
X1.1                    50.455214  ...                                          0
X1.2                    45.970545  ...                                          0
X1.3                    49.222215  ...                                          0
X1.4                    46.339991  ...                                          0

      total_counts_feature_control  log10_total_counts_feature_control  \
X1                               0                                    0
X1.1                             0                                    0
X1.2                             0                                    0
X1.3                             0                                    0
X1.4                             0                                    0

      pct_counts_feature_control  total_features_ERCC  \
X1                             0                    0
X1.1                           0                    0
X1.2                           0                    0
X1.3                           0                    0
X1.4                           0                    0

      log10_total_features_ERCC  total_counts_ERCC  log10_total_counts_ERCC  \
X1                            0                  0                        0
X1.1                          0                  0                        0
X1.2                          0                  0                        0
X1.3                          0                  0                        0
X1.4                          0                  0                        0

      pct_counts_ERCC  is_cell_control
X1                  0            False
X1.1                0            False
X1.2                0            False
X1.3                0            False
X1.4                0            False

[5 rows x 30 columns]
```

[8]:
```
labs = zeisel.coldata['cell_type1'].values.tolist()
```

[9]:
```
Counter(labs).most_common()
```

[9]:
```
[('ca1pyramidal', 948),
 ('oligodendrocytes', 820),
 ('s1pyramidal', 390),
 ('interneurons', 290),
 ('astrocytes', 198),
 ('endothelial', 175),
 ('microglia', 98),
 ('mural', 60),
 ('ependymal', 26)]
```

Create an instance of scedar data structure

The `sce.eda.SampleDistanceMatrix` class stores the data matrix and distance matrix of a scRNA-seq dataset without cell type label. The distance matrix is computed only when necessary, which could also be directly provided by the user. The `nprocs` parameter specifies the number of CPU cores to use for methods that support parallel computation.

[10]:
```
%%time
sdm = sce.eda.SampleDistanceMatrix(
    zeisel.x.values.T,
    fids=zeisel.x.index.values.tolist(),
    metric='cosine', nprocs=16)
```

```
CPU times: user 158 ms, sys: 246 ms, total: 404 ms
Wall time: 403 ms
```

t-SNE

Label text is added to randomly selected points in each category. Change `random_state` value to select differet set of points.

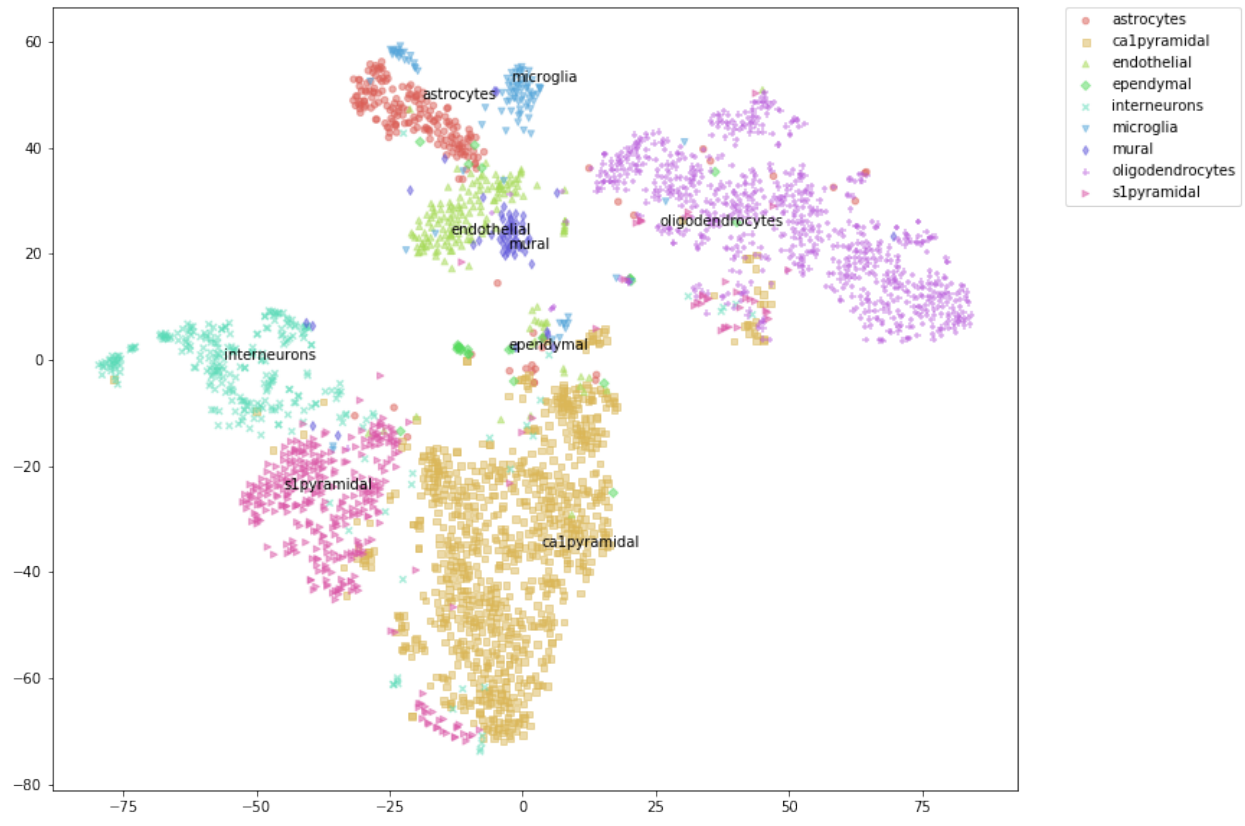
[11]:
```
%%time
tsne_x = sdm.tsne(perplexity=30, n_iter=3000, random_state=111)
```

```
CPU times: user 1min 23s, sys: 24.6 s, total: 1min 48s
Wall time: 1min 5s
```

[12]:
```
%%time
sdm.tsne_plot(labels=labs, figsize=(15, 10), alpha=0.5, s=20,
              n_txt_per_cluster=1, plot_different_markers=True,
              shuffle_label_colors=False, random_state=17)
```

```
CPU times: user 94.2 ms, sys: 74.3 ms, total: 169 ms
Wall time: 88.2 ms
```

[12]:



UMAP

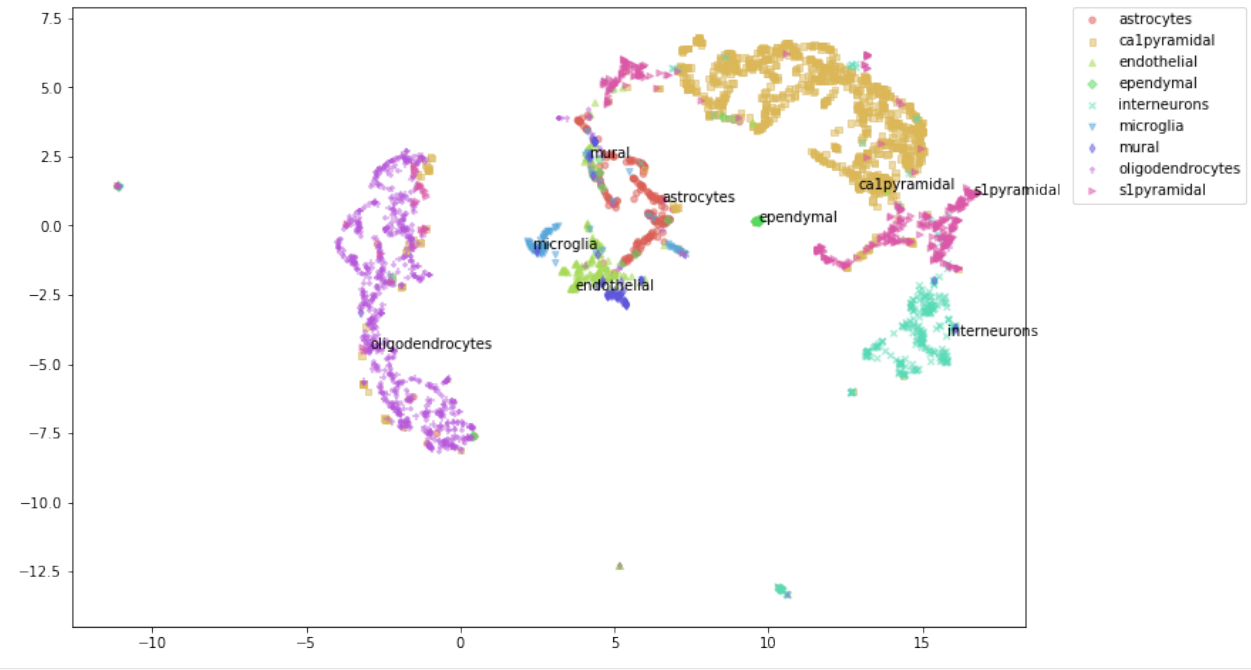
Label text is added to randomly selected points in each category. Change `random_state` value to select differet set of points.

[13]:
```
%%time
fig = sdm.umap_plot(labels=labs, alpha=0.5, s=20,
                    plot_different_markers=True,
                    n_txt_per_cluster=1,
                    figsize=(15, 8), random_state=99)
fig
```

```
/mnt/isilon/cbmi/variome/yuanchao/miniconda3/envs/py36/lib/python3.6/site-packages/
↪umap/spectral.py:229: UserWarning: Embedding a total of 2 separate connected␣
↪components using meta-embedding (experimental)
  n_components
```

```
CPU times: user 1min 20s, sys: 2min 19s, total: 3min 40s
Wall time: 17.3 s
```

[13]:



2.2.3 Clustering

[14]:
```
%%time
mirac_res = sce.cluster.MIRAC(
    sdm._last_tsne, metric='euclidean',
    linkage='ward', min_cl_n=25,
    min_split_mdl_red_ratio=0.00,
    optimal_ordering=False,
    cl_mdl_scale_factor=0.80, verbose=False)
```

```
CPU times: user 6.83 s, sys: 5.13 s, total: 12 s
Wall time: 5.45 s
```

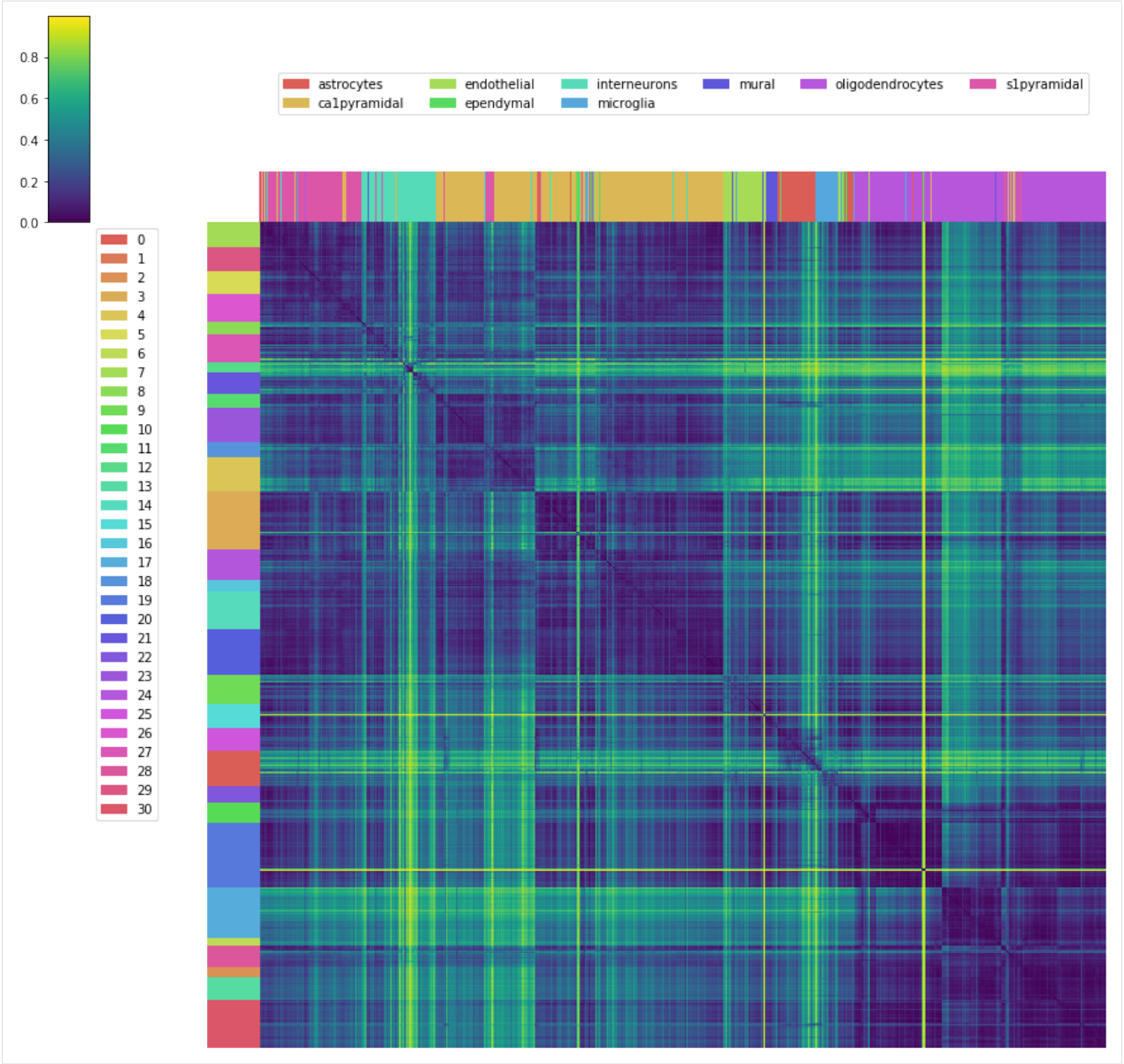
Visualize pairwise distance matrix

[15]:
```
# setup for pairwise distance matrix visualization
# this procedure will be simplified in future
# versions of scedar

# convert labels to random integers, in order
# to better visualize the pairwise distance
# matrix
np.random.seed(123)
uniq_labs = sorted(set(mirac_res.labs))
rand_lab_lut = dict(zip(uniq_labs,
                        np.random.choice(len(uniq_labs),
                                         size=len(uniq_labs),
                                         replace=False).tolist()))
olabs = mirac_res._labs
mirac_res._labs = [rand_lab_lut[l] for l in olabs]

# visualize the original cosine pairwise
# distance matrix rather than the t-SNE
# euclidean pairwise distance
tsne_euc_d = mirac_res._sdm._d
mirac_res._sdm._lazy_load_d = sdm._d

# order orignial cell types according to hac
# optimal ordering
mirac_res_ord_cell_types = [labs[i] for i in mirac_res._hac_tree.leaf_ids()]
```

[16]:
```
%%time
mirac_res.dmat_heatmap(col_labels=mirac_res_ord_cell_types,
                       figsize=(15, 15), cmap='viridis')
```

```
CPU times: user 487 ms, sys: 163 ms, total: 650 ms
Wall time: 649 ms
```

[16]:



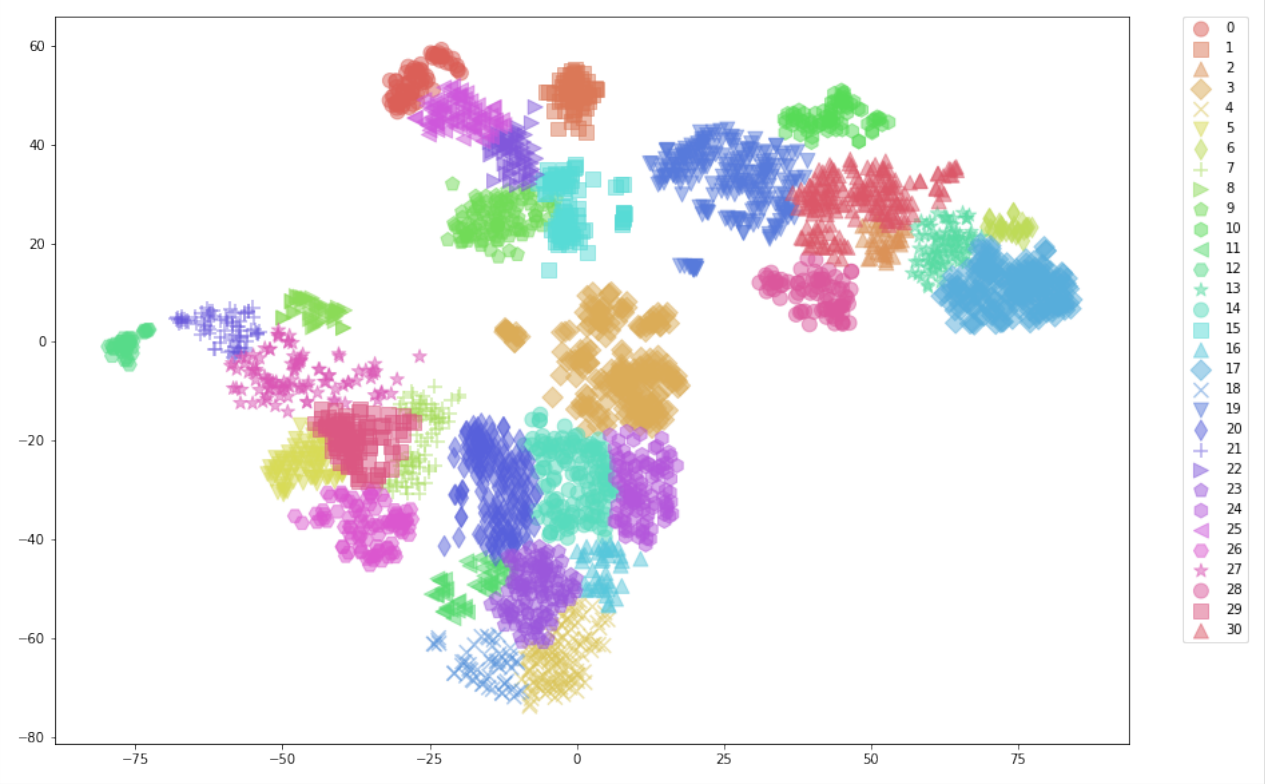
Visualize t-SNE

[17]:
```
slcs = sdm.to_classified(mirac_res.labs)
```

[18]:
```
%%time
slcs.tsne_plot(figsize=(18, 10), alpha=0.5, s=120,
               n_txt_per_cluster=0, plot_different_markers=True,
               shuffle_label_colors=False, random_state=15)
```

```
CPU times: user 211 ms, sys: 127 ms, total: 338 ms
Wall time: 143 ms
```

[18]:



Identify cluster separating genes

[19]:
```
cmp_labs = [1, 15, 22]
```

[20]:
```
xgb_params = {"eta": 0.3,
              "max_depth": 1,
              "silent": 1,
              "nthread": 20,
              "alpha": 1,
              "lambda": 0,
              "seed": 123}
```

[21]:
```
%%time
cmp_labs_sgs = slcs.feature_importance_across_labs(
    cmp_labs, nprocs=20, random_state=123, xgb_params=xgb_params,
    num_bootstrap_round=500, shuffle_features=True)
```

```
test rmse: mean 0.29898961799999996, std 0.04622533702208873
train rmse: mean 0.224621238, std 0.0222938908375076
CPU times: user 20min 26s, sys: 38.8 s, total: 21min 5s
Wall time: 3min 45s
```

[22]:
```
sorted_cmp_labs_sgs = sorted(cmp_labs_sgs[0], key=lambda t: t[1], reverse=True)
```

[23]:
```
[t[:4] for t in sorted_cmp_labs_sgs[:15]]
```

[23]:
```
[('Lyz2', 2.107142857142857, 1.0120448082360538, 28),
 ('Pf4', 2.017857142857143, 1.026280931821501, 56),
```

(continues on next page)

(continued from previous page)

```
('Ccl24', 2.0, 1.0954451150103321, 5),
('Tyrobp', 2.0, 0.7071067811865476, 4),
('Ccr1', 2.0, 0.0, 1),
('C1qb', 1.9127906976744187, 1.039039864448649, 344),
('F13a1', 1.8901734104046244, 0.9152317208646171, 173),
('Fcgr3', 1.8878504672897196, 0.9102439110699772, 107),
('Mrc1', 1.7959183673469388, 0.8801574960346051, 49),
('Ms4a7', 1.6666666666666667, 0.4714045207910317, 3),
('C1qa', 1.65, 0.852936105461599, 20),
('Cbr2', 1.3529411764705883, 0.8360394355030527, 34),
('Fcer1g', 1.3, 0.45825756949558405, 10),
('Clu', 1.2907608695652173, 0.4942291619843629, 368),
('Stab1', 1.2876712328767124, 0.6077006304881872, 73)]
```

The fields in each tuple are:

- Gene symbol.
- Mean of gene importance across 500 bootstrapping rounds.
- Standard deviation of gene importance across 500 bootstrapping rounds.
- Number of times of the gene is used in all bootstrapping rounds.

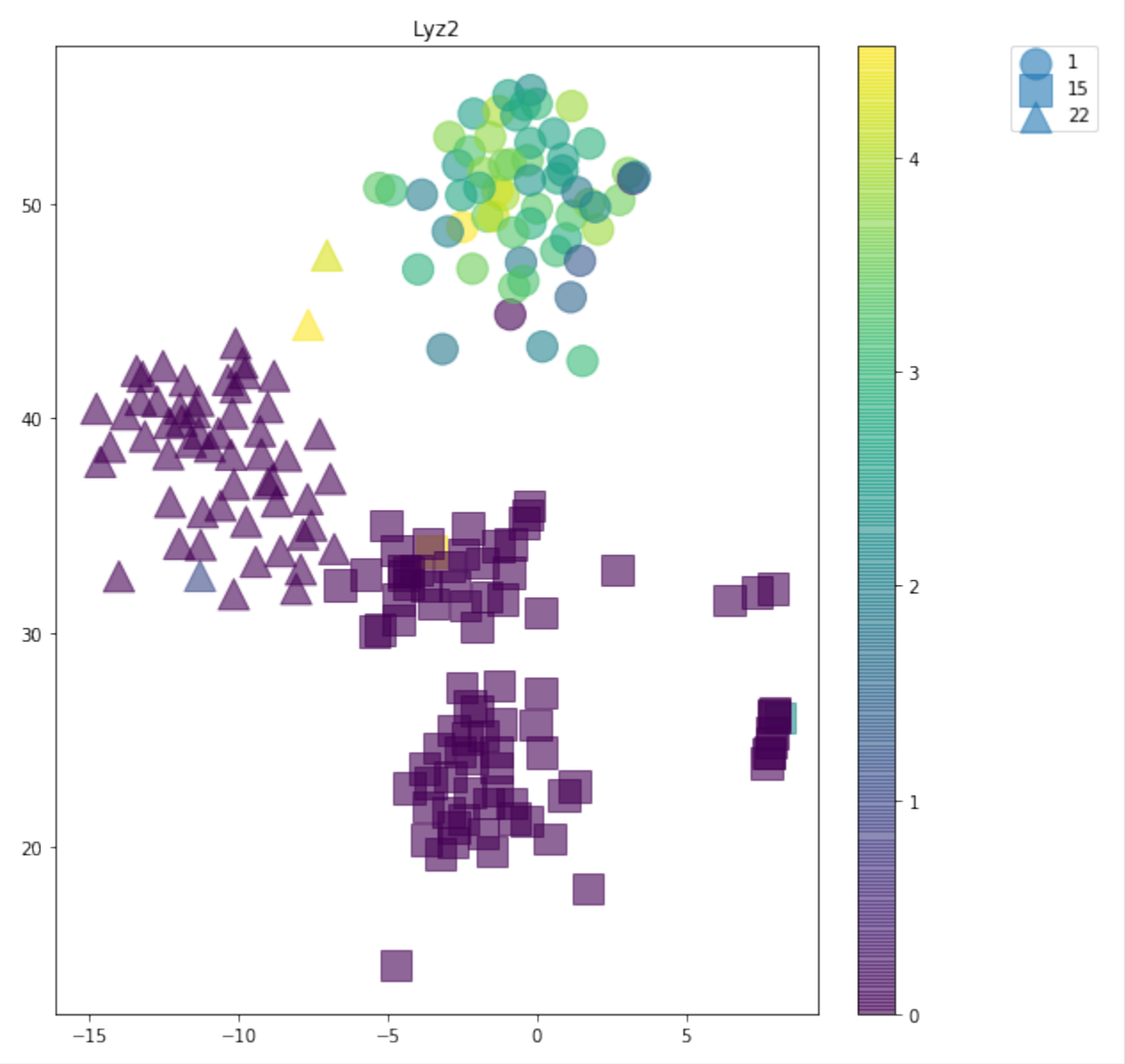
The importance of a gene in the trained classifier is the number of times it has been used as a inner node of the decision trees.

[24]:
```
len(sorted_cmp_labs_sgs)
```

[24]:
```
203
```

[25]:
```
sgid = sorted_cmp_labs_sgs[0][0]
slcs.tsne_feature_gradient_plot(sgid, figsize=(11, 10), alpha=0.6, s=300,
                                selected_labels=cmp_labs,
                                transform=lambda x: np.log2(x+1),
                                title=sgid, n_txt_per_cluster=0,
                                plot_different_markers=True,
                                shuffle_label_colors=True, random_state=15)
```

[25]:



2.2.4 K-nearest neighbors

Impute gene dropouts

[26]:

```
%%time
kfi = sce.knn.FeatureImputation(sdm)
kfi_sdm = kfi.impute_features(
    k=[50], n_do=[15], min_present_val=[2],
    n_iter=[1], nprocs=1)[0]
```

```
CPU times: user 45.7 s, sys: 2.11 s, total: 47.8 s
Wall time: 47.9 s
```

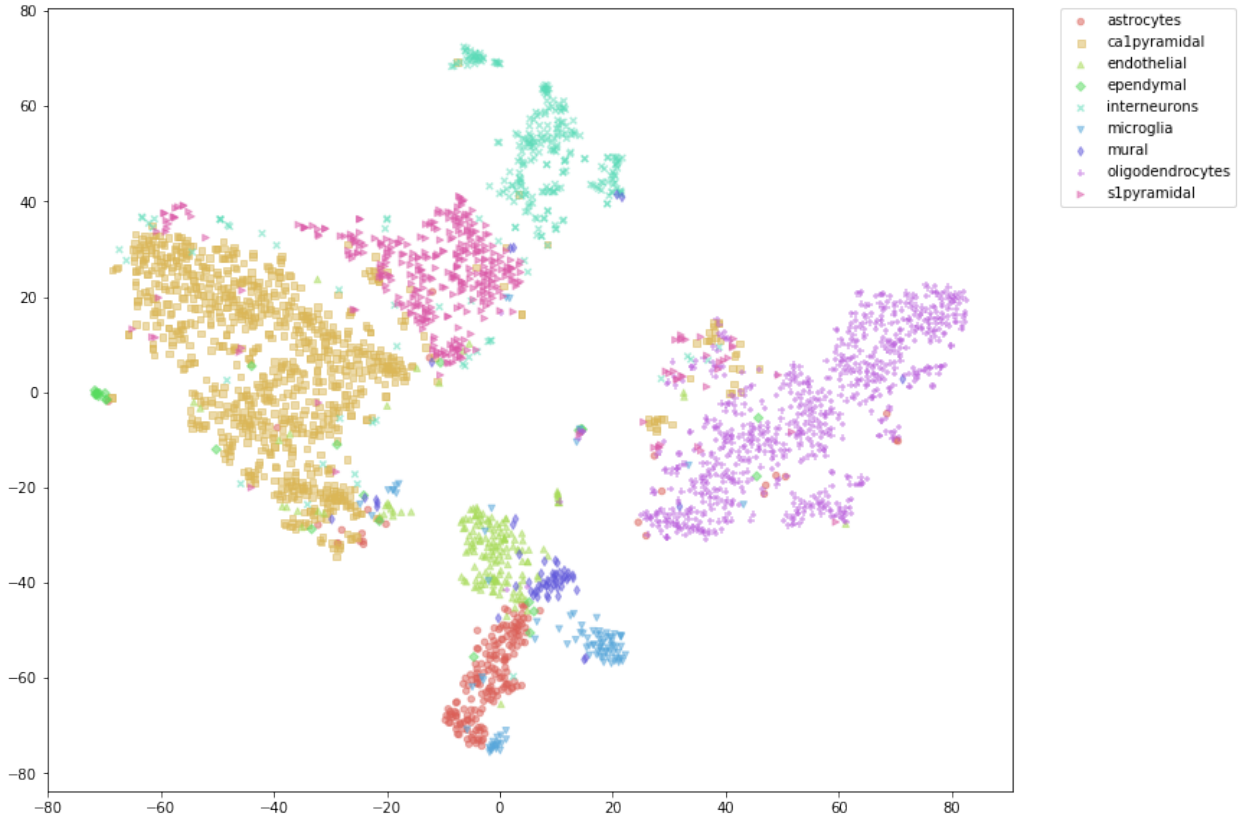
[27]:
```
%%time
kfi_tsne_x = kfi_sdm.tsne(perplexity=30, n_iter=3000, random_state=111)
```

```
CPU times: user 1min 19s, sys: 25.2 s, total: 1min 44s
Wall time: 1min 2s
```

[28]:
```
%%time
# t-SNE plot after imputation
kfi_sdm.tsne_plot(labels=labs, figsize=(15, 10), alpha=0.5, s=20,
                  n_txt_per_cluster=0, plot_different_markers=True,
                  shuffle_label_colors=False, random_state=15)
```

```
CPU times: user 192 ms, sys: 193 ms, total: 385 ms
Wall time: 152 ms
```

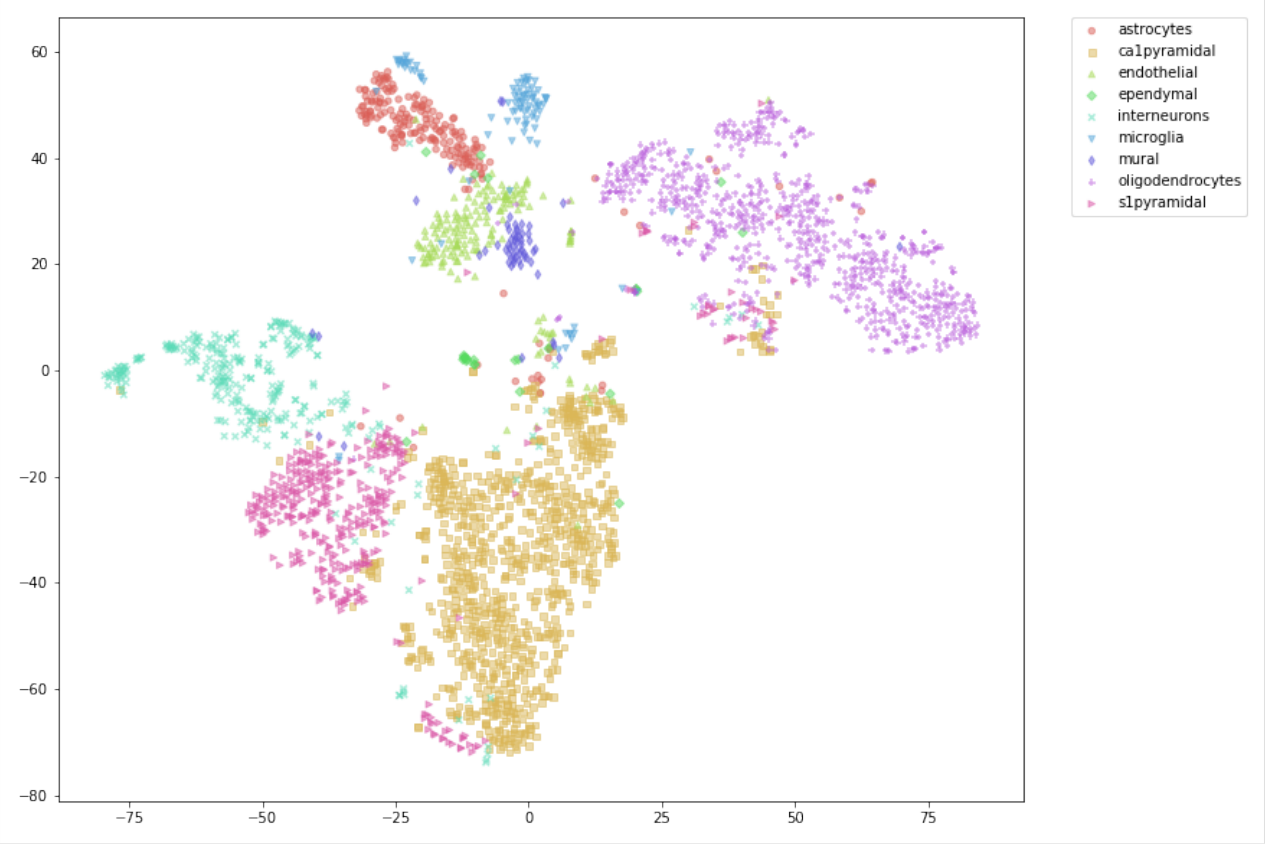
[28]:



[29]:
```
%%time
# t-SNE plot before imputation
sdm.tsne_plot(labels=labs, figsize=(15, 10), alpha=0.5, s=20,
              n_txt_per_cluster=0, plot_different_markers=True,
              shuffle_label_colors=False, random_state=15)
```

```
CPU times: user 85.4 ms, sys: 49.2 ms, total: 135 ms
Wall time: 65.3 ms
```

[29]:



Detect rare samples

[30]:
```
rsd = sce.knn.RareSampleDetection(sdm)
non_rare_s_inds = rsd.detect_rare_samples(50, 0.3, 20)[0]
```

[31]:
```
# number of rare samples
len(sdm.sids) - len(non_rare_s_inds)
```

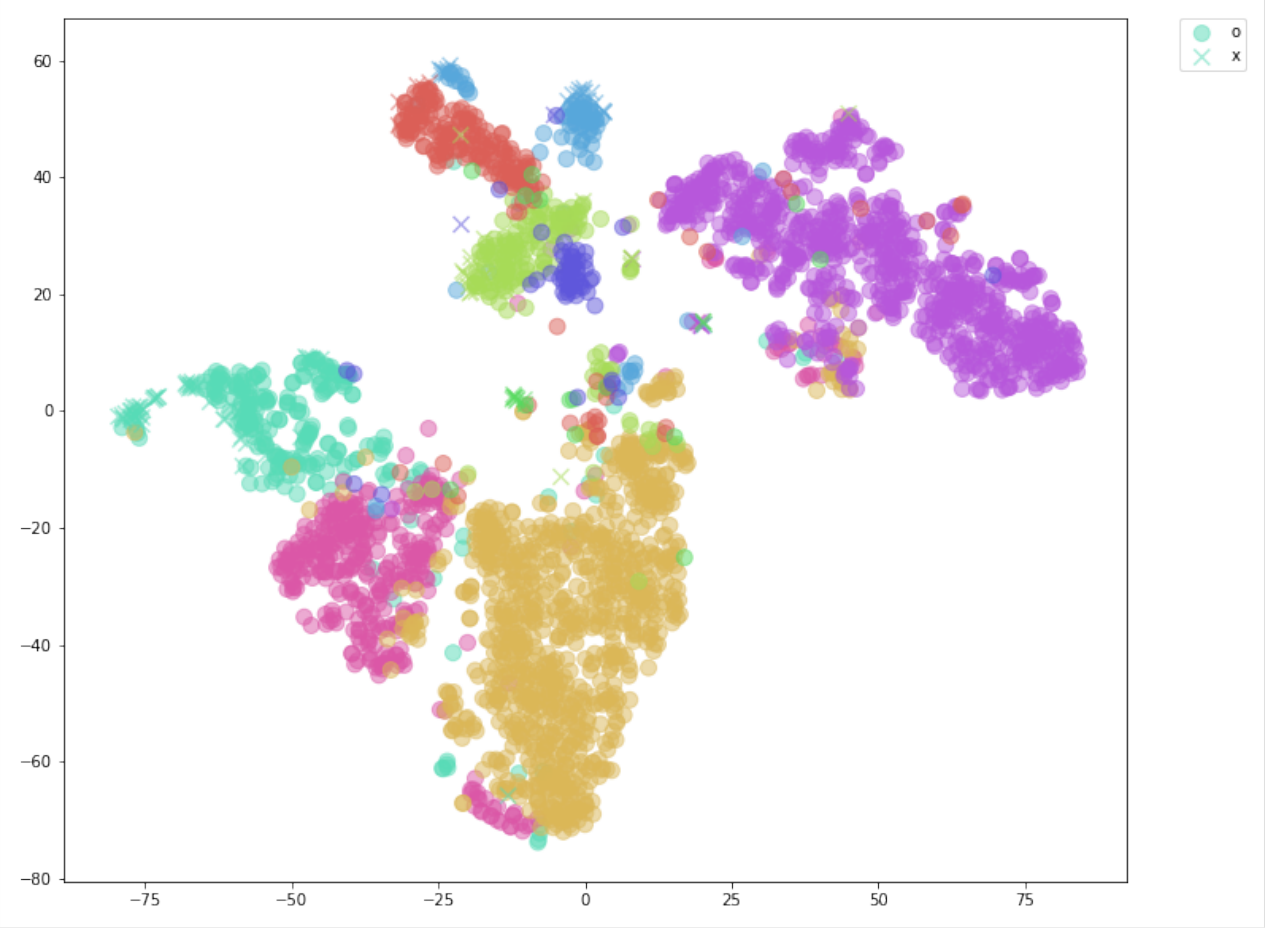
[31]:
```
133
```

[32]:
```
# ratio of non-rare samples
len(non_rare_s_inds) / len(sdm.sids)
```

[32]:
```
0.9557404326123128
```

[33]:
```
sdm.tsne_plot(labels=labs, s=100, figsize=(15, 10), alpha=0.5,
              n_txt_per_cluster=0, plot_different_markers=True,
              label_markers=['o' if i in non_rare_s_inds else 'x'
                             for i in range(len(sdm.sids))])
```

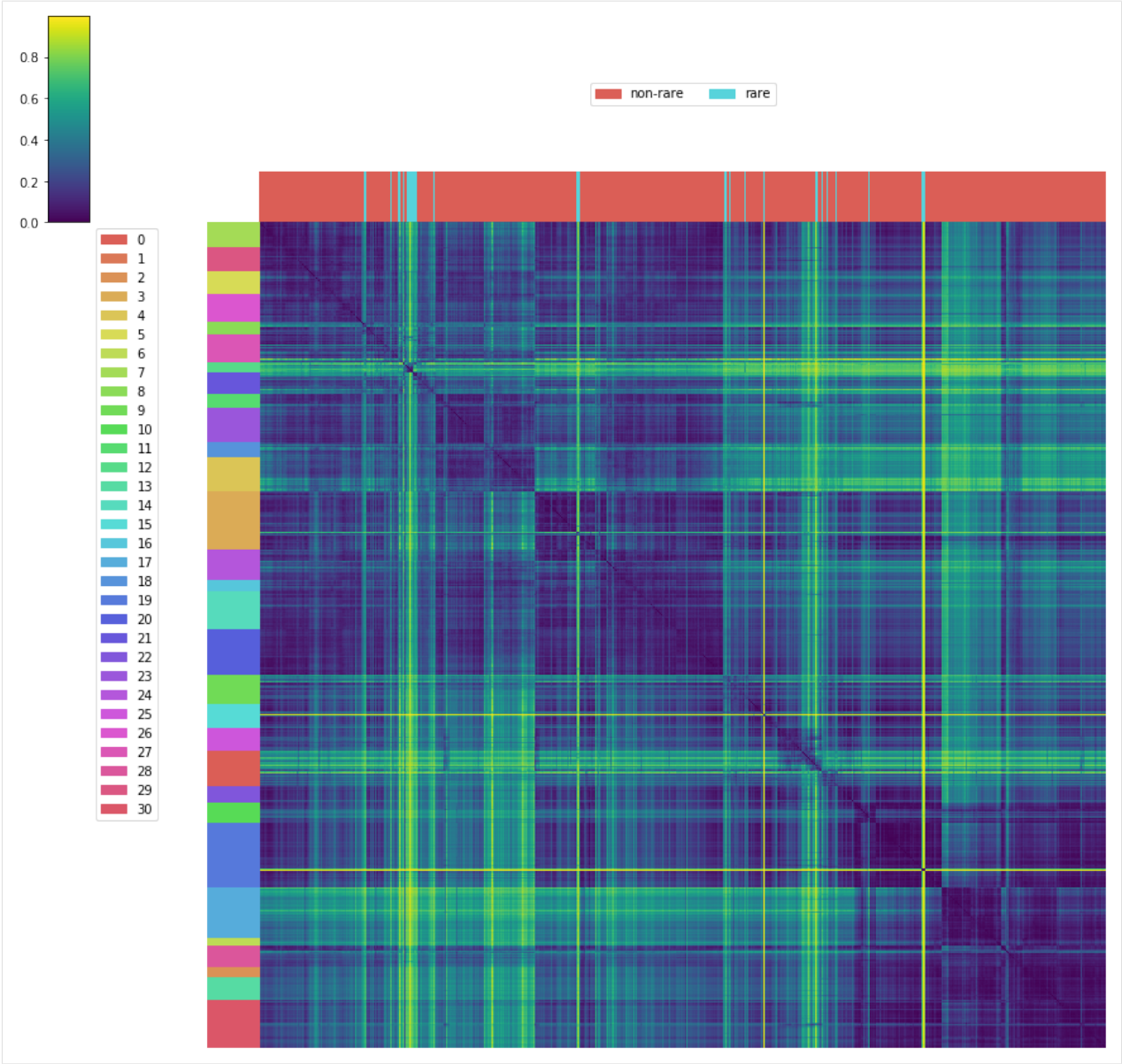
[33]:



[34]:
```
rsd_labs = np.array(['non-rare' if i in non_rare_s_inds else 'rare'
                     for i in range(len(sdm.sids))])[mirac_res._hac_tree.leaf_ids()].
↪tolist()
```

[35]:
```
mirac_res.dmat_heatmap(col_labels=rsd_labs,
                       cmap='viridis', figsize=(15, 15))
```

[35]:



[ ]:

2.3 API Reference Manual

2.3.1 scedar

scedar.cluster

scedar.cluster.mirac

class scedar.cluster.mirac.**MIRAC**(*x*, *d=None*, *metric='cosine'*, *sids=None*, *fids=None*, *hac_tree=None*, *nprocs=1*, *cl_mdl_scale_factor=1*, *min_cl_n=25*, *encode_type='auto'*, *mdl_method=None*, *min_split_mdl_red_ratio=0.2*, *soft_min_subtree_size=1*, *linkage='complete'*, *optimal_ordering=False*, *dim_reduct_method=None*, *verbose=False*)

Bases: `object`

MIRAC: MDL iteratively regularized agglomerative clustering.

Parameters

- **x** (*float array*) – Data matrix.
- **d** (*float array*) – Distance matrix.
- **metric** (*str*) – Type of distance metric.
- **sids** (*sid list*) – List of sample ids.
- **fids** (*fid list*) – List of feature ids.
- **hac_tree** (*HCTree*) – Hierarchical tree built by agglomerative clustering to divide in MIRAC. If provided, distance matrix will not be used for building another tree.
- **nprocs** (*int*) – Number of processes to run MIRAC parallely.
- **cl_mdl_scale_factor** (*float*) – Scale factor of cluster overhead mdl.
- **min_cl_n** (*int*) – Minimum # samples in a cluster.
- **encode_type** (*{"auto", "data", or "distance"}*) – Type of values to encode. If "auto", encode data when n_features <= 100.
- **mdl_method** (mdl.Mdl) – If None, use ZeroIGKdeMdl for encoded values with >= 50% zeros, and use GKdeMdl otherwise.
- **linkage** (*str*) – Linkage type for generating the hierarchy.
- **optimal_ordering** (*bool*) – To require hierarchical clustering tree with optimal ordering. Default value is False.
- **dim_reduct_method** (*{"PCA", "t-SNE", "UMAP", None}*) – If None, no dimensionality reduction before clustering.
- **verbose** (*bool*) – Print stats for each iteration.

_sdm
Data and distance matrices.

Type *SampleDistanceMatrix*

_min_cl_n
Stored parameter.

Type int

_encode_type
Encode type. If "auto" provided, this attribute will store the determined encode type.

Type str

_mdl_method
Mdl method. If None is provided, this attribute will store the determined mdl method.

Type *mdl.Mdl*

labs
Labels of clustered samples. 1-to-1 matching to from first to last.

Type label list

_hac_tree
Root node of the hierarchical agglomerative clustering tree.

Type eda.hct.HClustTree

_run_log
String containing the log of the MIRAC run.

Type str

TODO

*** Dendrogram representation of the splitting process.**

*** Take HCTree as parameter. Computing it is non-trivial when n is large.**

*** Simplify splitting criteria.**

dmat_heatmap(*selected_labels=None*, *col_labels=None*, *transform=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(10*, *10)*, ***kwargs*)

labs

tune_parameters(*cl_mdl_scale_factor=1*, *min_cl_n=25*, *min_split_mdl_red_ratio=0.2*, *soft_min_subtree_size=1*, *verbose=False*)

scedar.cluster.community

class scedar.cluster.community.**Community**(*x*, *d=None*, *graph=None*, *metric='cosine'*, *sids=None*, *fids=None*, *use_pdist=False*, *k=15*, *use_pca=True*, *use_hnsw=True*, *index_params=None*, *query_params=None*, *aff_scale=1*, *partition_method='RBConfigurationVertexPartition'*, *resolution=1*, *random_state=None*, *n_iter=2*, *nprocs=1*, *verbose=False*)

Bases: `object`

Community clustering

Parameters

- **x** (*float array*) – Data matrix.
- **d** (*float array*) – Distance matrix.
- **graph** (*igraph.Graph*) – Need to have a weight attribute as affinity. If this argument is not None, the graph will directly be used for community clustering.
- **metric** (*{'cosine', 'euclidean'}*) – Metric used for nearest neighbor computation.
- **sids** (*sid list*) – List of sample ids.
- **fids** (*fid list*) – List of feature ids.

- **use_pdist** (*boolean*) – To use the pairwise distance matrix or not. The pairwise distance matrix may be too large to save for datasets with a large number of cells.
- **k** (*int*) – The number of nearest neighbors.
- **use_pca** (*bool*) – Use PCA for nearest neighbors or not.
- **use_hnsw** (*bool*) – Use Hierarchical Navigable Small World graph to compute approximate nearest neighbor.
- **index_params** (*dict*) – Parameters used by HNSW in indexing.

  efConstruction [int] Default 100. Higher value improves the quality of a constructed graph and leads to higher accuracy of search. However this also leads to longer indexing times. The reasonable range of values is 100-2000.

  M [int] Default 5. Higher value leads to better recall and shorter retrieval times, at the expense of longer indexing time. The reasonable range of values is 5-100.

  delaunay_type [{0, 1, 2, 3}] Default 2. Pruning heuristic, which affects the trade-off between retrieval performance and indexing time. The default is usually quite good.

  post [{0, 1, 2}] Default 0. The amount and type of postprocessing applied to the constructed graph. 0 means no processing. 2 means more processing.

  indexThreadQty [int] Default self._nprocs. The number of threads used.
- **query_params** (*dict*) – Parameters used by HNSW in querying.

  efSearch [int] Default 100. Higher value improves recall at the expense of longer retrieval time. The reasonable range of values is 100-2000.
- **aff_scale** (*float > 0*) – Scaling factor used for converting distance to affinity. Affinity = (max(distance) - distance) * aff_scale.
- **partition_method** (*str*) – Following methods are implemented in leidenalg package:
  - RBConfigurationVertexPartition: only well-defined for positive edge weights.
  - RBERVertexPartition: well-defined only for positive edge weights.
  - CPMVertexPartition: well-defined for both positive and negative edge weights.
  - SignificanceVertexPartition: well-defined only for unweighted graphs.
  - SurpriseVertexPartition: well-defined only for positive edge weights.
- **resolution** (*float > 0*) – Resolution used for community clustering. Higer value produces more clusters.
- **random_state** (*int*) – Random number generator seed used for community clustering.
- **n_iter** (*int*) – Number of iterations used for community clustering.
- **nprocs** (*int > 0*) – The number of processes/cores used for community clustering.
- **verbose** (*bool*) – Print progress or not.

labs
Labels of clustered samples. 1-to-1 matching to from first to last.

Type label list

_sdm
Data and distance matrices.

Type *SampleDistanceMatrix*

_graph
Graph used for clustering.

Type igraph.Graph

_la_res
Partition results computed by leidenalg.

Type leidenalg.VertexPartition

_k

_use_pca

_use_hnsw

_index_params

_query_params

_aff_scale

labs

scedar.cluster.community_mirac

class scedar.cluster.community_mirac.**CommunityMIRAC**(*x*, *d=None*, *sids=None*, *fids=None*, *nprocs=1*, *verbose=False*)

Bases: object

CommunityMIRAC: Community + MIRAC clustering

Run community clustering with high resolution to get a large number of clusters. Then, run MIRAC on the community clusters.

Parameters

- **x** (*float array*) – Data matrix.
- **d** (*float array*) – Distance matrix.
- **sids** (*sid list*) – List of sample ids.
- **fids** (*fid list*) – List of feature ids.
- **nprocs** (*int > 0*) – The number of processes/cores used for community clustering.
- **verbose** (*bool*) – Print progress or not.

_x
Data matrix.

Type float array

_d
Distance matrix.

Type float array

_sids
List of sample ids.

Type sid list

_fids
List of feature ids.

Type fid list

_nprocs
The number of processes/cores used for community clustering.

Type int > 0

_verbose
Print progress or not.

Type bool

_cm_res
Community clustering result.

Type cluster.Community

_cm_clp_x
Data array with samples collapsed by community clustering labels. For each cluster, the mean of all samples is a row in this array.

Type array

_mirac_res
MIRAC clustering results on _cm_clp_x

Type cluster.MIRAC

labs
list of labels

Type list

static collapse_clusters(*data_x*, *cluster_labs*)

labs

run(*graph=None*, *metric='cosine'*, *use_pdist=False*, *k=15*, *use_pca=True*, *use_hnsw=True*, *index_params=None*, *query_params=None*, *aff_scale=1*, *partition_method='RBConfigurationVertexPartition'*, *resolution=100*, *random_state=None*, *n_iter=2*, *hac_tree=None*, *cl_mdl_scale_factor=1*, *min_cl_n=25*, *encode_type='auto'*, *mdl_method=None*, *min_split_mdl_red_ratio=0.2*, *soft_min_subtree_size=1*, *linkage='complete'*, *optimal_ordering=False*, *nprocs=None*)

run_community(*graph=None*, *metric='cosine'*, *use_pdist=False*, *k=15*, *use_pca=True*, *use_hnsw=True*, *index_params=None*, *query_params=None*, *aff_scale=1*, *partition_method='RBConfigurationVertexPartition'*, *resolution=100*, *random_state=None*, *n_iter=2*, *nprocs=None*)

run_mirac(*metric='cosine'*, *hac_tree=None*, *cl_mdl_scale_factor=1*, *min_cl_n=25*, *encode_type='auto'*, *mdl_method=None*, *min_split_mdl_red_ratio=0.2*, *soft_min_subtree_size=1*, *linkage='complete'*, *optimal_ordering=False*, *dim_reduct_method=None*, *nprocs=None*)

tune_mirac(*cl_mdl_scale_factor=1*, *min_cl_n=25*, *min_split_mdl_red_ratio=0.2*, *soft_min_subtree_size=1*, *verbose=False*)

scedar.eda

scedar.eda.mdl

class scedar.eda.mdl.**GKdeMdl**(*x*, *kde_bw_method='scott'*, *dtype=dtype('float64')*, *copy=True*)

Bases: *scedar.eda.mdl.Mdl*

Use Gaussian kernel density estimation to compute mdl

Parameters

- **x** (*1D np.number array*) – data used for fit mdl
- **bandwidth_method** – string KDE bandwidth estimation method bing passed to *scipy.stats.gaussian_kde*. Types: * *"scott"*: Scott's rule of thumb. * *"silverman"*: Silverman"s rule of thumb. * *constant*: constant will be timed by x.std(ddof=1) internally, because scipy times bw_method value by std. "Scipy weights its bandwidth by the ovariance of the input data" [3]. * *callable*: scipy calls the function on self
- **dtype** (*np.dtype*) – default to 64-bit float
- **copy** (*bool*) – passed to np.array()

_x

data to fit

Type 1d float array

_n

number of elements in data

Type int

_bw_method

bandwidth method

Type str

_kde

Type scipy kde

_logdens

log density

Type 1d float array

bandwidth

encode(*qx*, *mdl_scale_factor=1*)

Encode query data using fitted KDE code

Parameters

- **qx** (*1d float array*) –
- **mdl_scale_factor** (*number*) – times mdl by this number

Returns mdl

Return type float

static gaussian_kde_logdens(*x*, *bandwidth_method='scott'*, *ret_kernel=False*)

Estimate Gaussian kernel density estimation bandwidth for input *x*.

Parameters

- **x** (float array of shape *(n_samples)* or *(n_samples, n_features)*) – Data points for KDE estimation.

- **bandwidth_method** (*string*) – KDE bandwidth estimation method bing passed to *scipy.stats.gaussian_kde*.

kde

mdl

class scedar.eda.mdl.**Mdl**(*x*, *dtype=dtype('float64')*, *copy=True*)
Bases: abc.ABC

Minimum description length abstract base class

Interface of various mdl schemas. Subclasses must implement mdl property and encode method.

_x
data used for fit mdl

Type 1D np.number array

_n
number of points in x

Type np.int

encode(*x*)
Encode another 1D number array with fitted code :param x: data to encode :type x: 1D np.number array

mdl

x

class scedar.eda.mdl.**MultinomialMdl**(*x*, *dtype=dtype('float64')*, *copy=True*)
Bases: *scedar.eda.mdl.Mdl*

Encode discrete values using multinomial distribution

Parameters

- **x** (*1D np.number array*) – data used for fit mdl
- **dtype** (*np.dtype*) – default to 64-bit float
- **copy** (*bool*) – passed to np.array()

Note: When x only has 1 uniq value. Encode the the number of values only.

---

encode(*qx*, *use_adjescent_when_absent=False*)
Encode another 1D float array with fitted code

Parameters

- **qx** (*1d float array*) – query data
- **use_adjescent_when_absent** (*bool*) – whether to use adjascent value to compute query mdl. If not, uniform mdl is used. If adjascent values have same distance to query value, choose the one with smaller mdl.

Returns qmdl (float)

mdl

class scedar.eda.mdl.**ZeroIGKdeMdl**(*x*, *kde_bw_method='scott'*, *dtype=dtype('float64')*, *copy=True*)
Bases: *scedar.eda.mdl.Mdl*

Zero indicator Gaussian KDE MDL

Encode the 0s and non-0s using bernoulli distribution. Then, encode non-0s using gaussian kde. Finally, one ternary val indicates all 0s, all non-0s, or otherwise

Parameters

- **x** (*1D np.number array*) – data used for fit mdl
- **bandwidth_method** – string KDE bandwidth estimation method bing passed to *scipy.stats.gaussian_kde*. Types: * *"scott"*: Scott's rule of thumb. * *"silverman"*: Silverman"s rule of thumb. * *constant*: constant will be timed by x.std(ddof=1) internally, because scipy times bw_method value by std. "Scipy weights its bandwidth by the ovariance of the input data" [3]. * *callable*: scipy calls the function on self
- **dtype** (*np.dtype*) – default to 64-bit float
- **copy** (*bool*) – passed to np.array()

References

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html

[2] https://en.wikipedia.org/wiki/Kernel_density_estimation

[3] https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/

[4] https://github.com/scipy/scipy/blob/v1.0.0/scipy/stats/kde.py#L42-L564

bandwidth

encode(*qx*)
Encode qx

Parameters **qx** (*1d np number array*) –

Returns mdl (float)

kde

kde_mdl

mdl

x_nonzero

zi_mdl

class scedar.eda.mdl.**ZeroIMdl**(*x*, *dtype=dtype('float64')*, *copy=True*)
Bases: *scedar.eda.mdl.Mdl*

Encode an indicator vector of 0s and non-0s

encode(*qx*)
Encode another 1D number array with fitted code :param x: data to encode :type x: 1D np.number array

mdl

class scedar.eda.mdl.**ZeroIMultinomialMdl**(*x*, *dtype=dtype('float64')*, *copy=True*)
Bases: *scedar.eda.mdl.Mdl*

encode(*qx*, *use_adjescent_when_absent=False*)
Encode another 1D number array with fitted code :param x: data to encode :type x: 1D np.number array

mdl

scedar.eda.mdl.**np_number_1d**(*x*, *dtype=dtype('float64')*, *copy=True*)
Convert x to 1d np number array

Parameters

- **x** (*1d sequence of values convertable to np.number*) –
- **dtype** (*np number type*) – default to 64-bit float
- **copy** (*bool*) – passed to np.array()

Returns xarr (1d np.number array)

Raises `ValueError` – If x is not convertable to provided dtype or non-1d. If dtype is not subdtype of np number.

scedar.eda.mtype

scedar.eda.mtype.**check_is_valid_labs**(*labs*)

scedar.eda.mtype.**check_is_valid_sfids**(*sfids*)

scedar.eda.mtype.**is_uniq_np1darr**(*x*)
Test whether x is a 1D np array that only contains unique values.

scedar.eda.mtype.**is_valid_full_cut_tree_mat**(*cmat*)
Validate scipy hierarchical clustering cut tree Number of clusters should decrease from n to 1

scedar.eda.mtype.**is_valid_lab**(*lab*)

scedar.eda.mtype.**is_valid_sfid**(*sfid*)

scedar.eda.plot

scedar.eda.plot.**cluster_scatter**(*projection2d*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *gradient=None*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Scatter plot for clustering illustration

Parameters

- **projection2d** (*2 col numeric array*) – (n, 2) matrix to plot
- **labels** (*list of labels*) – labels of n samples
- **selected_labels** (*list of labels*) – selected labels to plot
- **plot_different_markers** (*bool*) – plot different markers for samples with different labels
- **label_markers** (*list of marker shapes*) – passed to matplotlib plot
- **shuffle_label_colors** (*bool*) – shuffle the color of labels to avoid similar colors show up in close clusters
- **gradient** (*list of number*) – color gradient of n samples
- **title** (*str*) –
- **xlab** (*str*) – x axis label
- **ylab** (*str*) – y axis label

- **figsize** (*tuple of two number*) – (width, height)
- **add_legend** (*bool*) –
- **n_txt_per_cluster** (*number*) – the number of text to plot per cluster. Could be 0.
- **alpha** (*number*) –
- **s** (*number*) – size of the points
- **random_state** (*int*) – random seed to shuffle features
- ****kwards** – passed to matplotlib plot

Returns matplotlib figure of the created scatter plot

scedar.eda.plot.**heatmap**(*x*, *row_labels=None*, *col_labels=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *transform=None*, *shuffle_row_colors=False*, *shuffle_col_colors=False*, *random_state=None*, *row_label_order=None*, *col_label_order=None*, ***kwargs*)

scedar.eda.plot.**hist_dens_plot**(*x*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)

Plot histogram and density plot of x.

scedar.eda.plot.**labs_to_cmap**(*labels*, *return_lut=False*, *shuffle_colors=False*, *random_state=None*)

scedar.eda.plot.**networkx_graph**(*ng*, *pos=None*, *alpha=0.05*, *figsize=(20, 20)*, *gradient=None*, *labels=None*, *different_label_markers=True*, *node_size=30*, *node_with_labels=False*, *nx_draw_kwargs=None*)

scedar.eda.plot.**regression_scatter**(*x*, *y*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(5, 5)*, *alpha=1*, *s=0.5*, *ax=None*, ***kwargs*)

Paired vector scatter plot.

scedar.eda.plot.**swarm**(*x*, *labels=None*, *selected_labels=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(10, 10)*, *ax=None*, ***kwargs*)

scedar.eda.sdm

class scedar.eda.sdm.**HClustTree**(*node*, *prev=None*)

Bases: object

Hierarchical clustering tree.

Implement simple tree operation routines. HCT is binary unbalanced tree.

node

current node

Type scipy.cluster.hierarchy.ClusterNode

prev

parent of current node

Type *HClustTree*

bi_partition(*soft_min_subtree_size=1*, *return_subtrees=False*)

soft_min_subtree_size: when curr tree size < 2 * soft_min_subtree_size, it is impossible to have a bi-partition with a minimum sub tree size bigger than soft_min_subtree_size. In this case, return the first partition.

When soft_min_subtree_size = 1, the performance is the same as taking the first bipartition.

When curr size = 1, the first bipartition gives (1, 0). Because curr size < 2 * soft_min_subtree_size, it goes directly to return.

When curr size = 2, the first bipartition guarantees to give (1, 1), with the invariant that parent nodes of leaves always have 2 child nodes. This also goes directly to return.

When curr size >= 3, the first bipartition guarantees to give two subtrees with size >= 1, with the same invariant in size = 2.

static cluster_id_to_lab_list(*cl_sid_list*, *sid_list*)

Convert nested clustered ID list into cluster label list.

For example, convert *[[0, 1, 2], [3,4]]* to *[0, 0, 0, 1, 1]* according to id_arr *[0, 1, 2, 3, 4]*

Parameters

cl_sid_list: list[list[id]] Nested list with each sublist as a sert of IDs from a cluster.

sid_list: list[id] Flat list of lists.

count()

static hclust_linkage(*dmat*, *linkage='complete'*, *n_eval_rounds=None*, *is_euc_dist=False*, *optimal_ordering=False*, *verbose=False*)

static hclust_tree(*dmat*, *linkage='complete'*, *n_eval_rounds=None*, *is_euc_dist=False*, *optimal_ordering=False*, *verbose=False*)

static hct_from_lkg(*hac_z*)

leaf_ids()

Returns the list of leaf IDs from left to right

Returns `list` of leaf IDs

left()

left_count()

left_leaf_ids()

n_round_bipar_cnt(*n*)

prev

right()

right_count()

right_leaf_ids()

static sort_x_by_d(*x*, *dmat=None*, *metric='cosine'*, *linkage='auto'*, *n_eval_rounds=None*, *optimal_ordering=False*, *nprocs=None*, *verbose=False*)

class scedar.eda.sdm.**SampleDistanceMatrix**(*x*, *d=None*, *metric='cosine'*, *use_pdist=True*, *sids=None*, *fids=None*, *nprocs=None*)

Bases: *scedar.eda.sfm.SampleFeatureMatrix*

SampleDistanceMatrix: data with pairwise distance matrix

Parameters

- **x** (*ndarray or list*) – data matrix (n_samples, n_features)
- **d** (*ndarray or list or None*) – distance matrix (n_samples, n_samples) If is None, d will be computed with x, metric, and nprocs.
- **metric** (*string*) – distance metric

- **use_pdist** (*boolean*) – to use the pairwise distance matrix or not. The pairwise distance matrix may be too large to save for datasets with a large number of cells.
- **sids** (*homogenous list of int or string*) – sample ids. Should not contain duplicated elements.
- **fids** (*homogenous list of int or string*) – feature ids. Should not contain duplicated elements.
- **nprocs** (*int*) – the number of processes for computing pairwise distance matrix

_x
data matrix (n_samples, n_features)

Type ndarray

_d
distance matrix (n_samples, n_samples)

Type ndarray

_metric
distance metric

Type string

_sids
sample ids.

Type ndarray

_fids
sample ids.

Type ndarray

_tsne_lut
lookup table for previous tsne calculations. Each run has an indexed entry, {(param_str, index) : tsne_res}

Type dict

_last_tsne
The last *stored* tsne results. In no tsne performed before, a run with default parameters will be performed.

Type float array

_hnsw_index_lut

Type {string_index_parameters: hnsw_index}

_last_k
The last *k* used for s_knns computation.

Type int

_last_knns
The last computed s_knns.

Type (knn_indices, knn_distances)

_knn_ng_lut
{(k, aff_scale): knn_graph}

Type dict

static correlation_pdist(*x*)
Compute pairwise correlation pdist for x (n_samples, n_features).

Adapted from Waylon Flinn's post on https://stackoverflow.com/a/20687984/4638182 .

Parameters **x** (*ndarray*) – (n_samples, n_features)

Returns **d** – Pairwise distance matrix, (n_samples, n_samples).

Return type ndarray

static cosine_pdist(*x*)
Compute pairwise cosine pdist for x (n_samples, n_features).

Adapted from Waylon Flinn's post on https://stackoverflow.com/a/20687984/4638182 .

Cosine distance is undefined if one of the vectors contain only 0s.

Parameters **x** (*ndarray*) – (n_samples, n_features)

Returns **d** – Pairwise distance matrix, (n_samples, n_samples).

Return type ndarray

d

get_tsne_kv(*key*)
Get t-SNE results from the lookup table. Return None if non-existent.

Returns **res_tuple** – (key, val) pair of tsne result.

Return type tuple

id_x(*selected_sids=None*, *selected_fids=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_s_inds** (*int array*) – Index array of selected samples. If is None, select all.
- **selected_f_inds** (*int array*) – Index array of selected features. If is None, select all.

Returns **subset**

Return type *SampleDistanceMatrix*

ind_x(*selected_s_inds=None*, *selected_f_inds=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_s_inds** (*int array*) – Index array of selected samples. If is None, select all.
- **selected_f_inds** (*int array*) – Index array of selected features. If is None, select all.

Returns **subset**

Return type *SampleDistanceMatrix*

static knn_conn_mat_to_aff_graph(*knn_conn_mat*, *aff_scale=1*)

metric

static num_correct_dist_mat(*dmat*, *upper_bound=None*)

par_tsne(*param_list*, *store_res=True*, *nprocs=1*)
Run t-SNE with multiple sets of parameters parallely.

Parameters

- **param_list** (*list of dict*) – List of parameters being passed to t-SNE.
- **nprocs** (*int*) – Number of processes.

Returns **tsne_res_list** – List of t-SNE results of corresponding parameter set.

Return type list of float arrays

Notes

Parallel running results cannot be stored during the run, because racing conditions may happen.

pca_feature_gradient_plot(*fid*, *component_ind_pair=(0, 1)*, *transform=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last PCA projection with the provided gradient as color.

Parameters

- **component_ind_pair** (*tuple of two ints*) – Indices of the components to plot.
- **fid** (*feature id scalar*) – ID of the feature to be used for gradient plot.
- **transform** (*callable*) – Map transform on feature before plotting.
- **labels** (*label array*) – Labels assigned to each point, (n_samples,).
- **selected_labels** (*label array*) – Show gradient only for selected labels. Do not show non-selected.

pca_plot(*component_ind_pair=(0, 1)*, *gradient=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the PCA projection with the provided gradient as color. Gradient is None by default.

put_tsne(*str_params*, *res*)
Put t-SNE results into the lookup table.

s_ith_nn_d(*i*)
Computes the distances of the i-th nearest neighbor of all samples.

s_ith_nn_d_dist(*i*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distances of the i-th nearest neighbor of all samples.

s_ith_nn_ind(*i*)
Computes the sample indices of the i-th nearest neighbor of all samples.

s_knn_connectivity_matrix(*k*, *metric=None*, *use_pca=False*, *use_hnsw=False*, *index_params=None*, *query_params=None*, *verbose=False*)
Computes the connectivity matrix of KNN of samples. If an entry *(i, j)* has value 0, node *i* is not in node *j*'s KNN. If an entry *(i, j)* has value != 0, node *i* is in *j*'s KNN, and their distance is the entry value. If two NNs have distance euqal to 0, 0 will be replaced by -np.inf.

Parameters

- **k** (*int*) – The number of nearest neighbors.
- **metric** (*{'cosine', 'euclidean', None}*) – If none, self._metric is used.
- **use_pca** (*bool*) – Use PCA for nearest neighbors or not.
- **use_hnsw** (*bool*) – Use Hierarchical Navigable Small World graph to compute approximate nearest neighbor.
- **index_params** (*dict*) – Parameters used by HNSW in indexing.

  efConstruction: int Default 100. Higher value improves the quality of a constructed graph and leads to higher accuracy of search. However this also leads to longer indexing times. The reasonable range of values is 100-2000.

  M: int Default 5. Higher value leads to better recall and shorter retrieval times, at the expense of longer indexing time. The reasonable range of values is 5-100.

  delaunay_type: {0, 1, 2, 3} Default 2. Pruning heuristic, which affects the trade-off between retrieval performance and indexing time. The default is usually quite good.

  post: {0, 1, 2} Default 0. The amount and type of postprocessing applied to the constructed graph. 0 means no processing. 2 means more processing.

  indexThreadQty: int Default self._nprocs. The number of threads used.
- **query_params** (*dict*) – Parameters used by HNSW in querying.

  efSearch: int Default 100. Higher value improves recall at the expense of longer retrieval time. The reasonable range of values is 100-2000.

Returns knn_conn_mat – (n_samples, n_samles) Non-zero entries are nearest neighbors (NNs). The values are distances. If two NNs have distance euqal to 0, 0 will be replaced by -np.inf.

Return type float array

s_knn_graph(*k*, *gradient=None*, *labels=None*, *different_label_markers=True*, *aff_scale=1*, *iterations=2000*, *figsize=(20, 20)*, *node_size=30*, *alpha=0.05*, *random_state=None*, *init_pos=None*, *node_with_labels=False*, *fa2_kwargs=None*, *nx_draw_kwargs=None*)

Draw KNN graph of SampleDistanceMatrix. Graph layout using forceatlas2 for its speed on large graph.

Parameters

- **k** (*int*) –
- **gradient** (*float array*) – (n_samples,) color gradient
- **labels** (*label list*) – (n_samples,) labels
- **different_label_markers** (*bool*) – whether plot different labels with different markers
- **aff_scale** (*float*) – Affinity is calculated by *(max(distance) - distance) * aff_scale*
- **iterations** (*int*) – ForceAtlas2 iterations
- **figsize** (*(float, float)*) –
- **node_size** (*float*) –
- **alpha** (*float*) –
- **random_state** (*int*) –
- **init_pos** (*float array*) – Initial position of ForceAtlas2, (n_samples, 2).

- **node_with_labels** (*bool*) –
- **fa2_kwargs** (*dict*) –
- **nx_draw_kwargs** (*dict*) –

Returns **fig** – KNN graph.

Return type matplotlib figure

s_knn_ind_lut(*k*, *metric=None*, *use_pca=False*, *use_hnsw=False*, *index_params=None*, *query_params=None*, *verbose=False*)

Computes the lookup table for sample i and its KNN indices, i.e. *{i : [1st_NN_ind, 2nd_NN_ind, ..., nth_NN_ind], ... }*

s_knns(*k*, *metric=None*, *use_pca=False*, *use_hnsw=False*, *index_params=None*, *query_params=None*, *verbose=False*)

Computes the k-nearest neighbors (KNNs) of samples.

Parameters

- **k** (*int*) – The number of nearest neighbors.
- **metric** (*{'cosine', 'euclidean', None}*) – If none, self._metric is used.
- **use_pca** (*bool*) – Use PCA for nearest neighbors or not.
- **use_hnsw** (*bool*) – Use Hierarchical Navigable Small World graph to compute approximate nearest neighbor.
- **index_params** (*dict*) – Parameters used by HNSW in indexing.

  efConstruction: int Default 100. Higher value improves the quality of a constructed graph and leads to higher accuracy of search. However this also leads to longer indexing times. The reasonable range of values is 100-2000.

  M: int Default 5. Higher value leads to better recall and shorter retrieval times, at the expense of longer indexing time. The reasonable range of values is 5-100.

  delaunay_type: {0, 1, 2, 3} Default 2. Pruning heuristic, which affects the trade-off between retrieval performance and indexing time. The default is usually quite good.

  post: {0, 1, 2} Default 0. The amount and type of postprocessing applied to the constructed graph. 0 means no processing. 2 means more processing.

  indexThreadQty: int Default self._nprocs. The number of threads used.
- **query_params** (*dict*) – Parameters used by HNSW in querying.

  efSearch: int Default 100. Higher value improves recall at the expense of longer retrieval time. The reasonable range of values is 100-2000.
- **verbose** (*bool*) –

Returns

- **knn_indices** (*list of numpy arrays*) – The i-th array is the KNN indices of the i-th sample.
- **knn_distances** (*list of numpy arrays*) – The i-th array is the KNN distances of the i-th sample.

sort_features(*fdist_metric='cosine'*, *optimal_ordering=False*)

to_classified(*labels*)

Convert to SingleLabelClassifiedSamples

Parameters **labels** (*list of labels*) – sample labels.

Returns SingleLabelClassifiedSamples

tsne(*store_res=True*, ***kwargs*)
Run t-SNE on distance matrix.

Parameters

- **store_res** (*bool*) – Store the results in lookup table or not.
- ****kwargs** – Keyword arguments passed to tsne computation.

Returns **tsne_res** – t-SNE projections, (n_samples, m dimensions).

Return type float array

tsne_feature_gradient_plot(*fid*, *transform=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last t-SNE projection with the provided gradient as color.

Parameters

- **fid** (*feature id scalar*) – ID of the feature to be used for gradient plot.
- **transform** (*callable*) – Map transform on feature before plotting.
- **labels** (*label array*) – Labels assigned to each point, (n_samples,).
- **selected_labels** (*label array*) – Show gradient only for selected labels. Do not show non-selected.

tsne_lut

tsne_plot(*gradient=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last t-SNE projection with the provided gradient as color. Gradient is None by default.

umap(*use_pca=True*, *n_neighbors=5*, *n_components=2*, *n_epochs=None*, *learning_rate=1.0*, *init='spectral'*, *min_dist=0.1*, *spread=1.0*, *set_op_mix_ratio=1.0*, *local_connectivity=1.0*, *repulsion_strength=1.0*, *negative_sample_rate=5*, *transform_queue_size=4.0*, *a=None*, *b=None*, *random_state=None*, *metric_kwds=None*, *angular_rp_forest=False*, *target_n_neighbors=-1*, *target_metric='categorical'*, *target_metric_kwds=None*, *target_weight=0.5*, *transform_seed=42*, *verbose=False*)

umap_feature_gradient_plot(*fid*, *component_ind_pair=(0, 1)*, *transform=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last UMAP projection with the provided gradient as color.

Parameters

- **component_ind_pair** (*tuple of two ints*) – Indices of the components to plot.
- **fid** (*feature id scalar*) – ID of the feature to be used for gradient plot.
- **transform** (*callable*) – Map transform on feature before plotting.

- **labels** (*label array*) – Labels assigned to each point, (n_samples,).
- **selected_labels** (*label array*) – Show gradient only for selected labels. Do not show non-selected.

umap_plot(*component_ind_pair=(0, 1)*, *gradient=None*, *labels=None*, *selected_labels=None*, *plot_different_markers=False*, *label_markers=None*, *shuffle_label_colors=False*, *xlim=None*, *ylim=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)

Plot the UMAP projection with the provided gradient as color. Gradient is None by default.

TODO: refactor plotting interface. Merge multiple plotting methods into one.

scedar.eda.sdm.**tsne**(*x*, *n_components=2*, *perplexity=30.0*, *early_exaggeration=12.0*, *learning_rate=200.0*, *n_iter=1000*, *n_iter_without_progress=300*, *min_grad_norm=1e-07*, *metric='euclidean'*, *init='random'*, *verbose=0*, *random_state=None*, *method='barnes_hut'*, *angle=0.5*)

scedar.eda.sfm

class scedar.eda.sfm.**SampleFeatureMatrix**(*x*, *sids=None*, *fids=None*)

Bases: object

SampleFeatureMatrix is a (n_samples, n_features) matrix.

In this package, we are only interested in float features as measured expression levels.

Parameters

- **x** (*{array-like, sparse matrix}*) – data matrix (n_samples, n_features)
- **sids** (*homogenous list of int or string*) – sample ids. Should not contain duplicated elements.
- **fids** (*homogenous list of int or string*) – feature ids. Should not contain duplicated elements.

_x

data matrix (n_samples, n_features)

Type {array-like, sparse matrix}

_is_sparse

whether the data matrix is sparse matrix or not

Type boolean

_sids

sample ids.

Type ndarray

_fids

sample ids.

Type ndarray

f_cv(*f_cv_filter=None*)

For each sample, compute the coefficient of variation of all features.

Returns **xf** – (filtered_n_samples,)

Return type float array

f_cv_dist(*f_cv_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the feature sum of each sample, (n_samples,).

f_gc(*f_gc_filter=None*)
For each sample, compute the Gini coefficients of all features.

Returns **xf** – (filtered_n_samples,)

Return type float array

f_gc_dist(*f_gc_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the feature Gini coefficient of each sample, (n_samples,).

f_id_dist(*f_id*, *sample_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)

f_id_regression_scatter(*xf_id*, *yf_id*, *sample_filter=None*, *xlab=None*, *ylab=None*, *title=None*, ***kwargs*)
Regression plot on two features with xf_id and yf_id.

Parameters

- **xf_id** (*int*) – Sample ID of x.
- **yf_ind** (*int*) – Sample ID of y.
- **sample_filter** (*bool array, or int array, or callable(x, y)*) – If sample_filter is bool / int array, directly select features with it. If sample_filter is callable, it will be applied on each (x, y) value tuple.
- **xlab** (*str*) –
- **ylab** (*str*) –
- **title** (*str*) –

f_id_to_ind(*selected_fids*)
Convert a list of feature IDs into feature indices.

f_id_x_vec(*f_id*, *sample_filter=None*)

f_ind_dist(*f_ind*, *sample_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)

f_ind_regression_scatter(*xf_ind*, *yf_ind*, *sample_filter=None*, *xlab=None*, *ylab=None*, *title=None*, ***kwargs*)
Regression plot on two features with xf_ind and yf_ind.

Parameters

- **xf_ind** (*int*) – Sample index of x.
- **yf_ind** (*int*) – Sample index of y.
- **sample_filter** (*bool array, or int array, or callable(x, y)*) – If sample_filter is bool / int array, directly select features with it. If sample_filter is callable, it will be applied on each (x, y) value tuple.
- **xlab** (*str*) –
- **ylab** (*str*) –
- **title** (*str*) –

f_ind_x_pair(*xf_ind*, *yf_ind*, *sample_filter=None*)

f_ind_x_vec(*f_ind*, *sample_filter=None*, *transform=None*)
Access a single vector of a sample.

f_n_above_threshold(*closed_threshold*)
For each sample, compute the number of features above a closed threshold.

f_n_above_threshold_dist(*closed_threshold*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the the number of above threshold samples of each feature, (n_features,).

f_sum(*f_sum_filter=None*)
For each sample, compute the sum of all features.

Returns **rowsum** – (filtered_n_samples,)

Return type float array

f_sum_dist(*f_sum_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the feature sum of each sample, (n_samples,).

fids

static filter_1d_inds(*f*, *x*)

id_x(*selected_sids=None*, *selected_fids=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_sids** (*id array*) – ID array of selected samples. If is None, select all.
- **selected_fids** (*id array*) – ID array of selected features. If is None, select all.

Returns **subset**

Return type *SampleFeatureMatrix*

ind_x(*selected_s_inds=None*, *selected_f_inds=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_s_inds** (*int array*) – Index array of selected samples. If is None, select all.
- **selected_f_inds** (*int array*) – Index array of selected features. If is None, select all.

Returns **subset**

Return type *SampleFeatureMatrix*

s_cv(*s_cv_filter=None*)
For each feature, compute the coefficient of variation of all samples.

Returns **xf** – (n_features,)

Return type float array

s_cv_dist(*s_cv_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the sample coefficient of variation of each feature, (n_features,).

s_gc(*s_gc_filter=None*)
For each feature, compute the Gini coefficient of all samples.

Returns **xf** – (n_features,)

Return type float array

s_gc_dist(*s_gc_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the sample Gini coefficients of each feature, (n_features,).

s_id_dist(*s_id*, *feature_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)

s_id_regression_scatter(*xs_id*, *ys_id*, *feature_filter=None*, *xlab=None*, *ylab=None*, *title=None*, ***kwargs*)
Regression plot on two samples with xs_id and ys_id.

Parameters

- **xs_ind** (*int*) – Sample ID of x.
- **ys_ind** (*int*) – Sample ID of y.
- **feature_filter** (*bool array, or int array, or callable(x, y)*) – If feature_filter is bool / int array, directly select features with it. If feature_filter is callable, it will be applied on each (x, y) value tuple.
- **xlab** (*str*) –
- **ylab** (*str*) –
- **title** (*str*) –

s_id_to_ind(*selected_sids*)
Convert a list of sample IDs into sample indices.

s_ind_dist(*s_ind*, *feature_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)

s_ind_regression_scatter(*xs_ind*, *ys_ind*, *feature_filter=None*, *xlab=None*, *ylab=None*, *title=None*, ***kwargs*)
Regression plot on two samples with xs_ind and ys_ind.

Parameters

- **xs_ind** (*int*) – Sample index of x.
- **ys_ind** (*int*) – Sample index of y.
- **feature_filter** (*bool array, or int array, or callable(x, y)*) – If feature_filter is bool / int array, directly select features with it. If feature_filter is callable, it will be applied on each (x, y) value tuple.
- **xlab** (*str*) –
- **ylab** (*str*) –
- **title** (*str*) –

s_ind_x_pair(*xs_ind*, *ys_ind*, *feature_filter=None*)

s_ind_x_vec(*s_ind*, *feature_filter=None*)
Access a single vector of a sample.

s_n_above_threshold(*closed_threshold*)
For each feature, compute the number of samples above a closed threshold.

s_n_above_threshold_dist(*closed_threshold*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the the number of above threshold samples of each feature, (n_features,).

s_sum(*s_sum_filter=None*)
For each feature, computer the sum of all samples.

Returns **xf** – (filtered_n_features,)

Return type float array

s_sum_dist(*s_sum_filter=None*, *xlab=None*, *ylab=None*, *title=None*, *figsize=(5, 5)*, *ax=None*, ***kwargs*)
Plot the distribution of the sample sum of each feature, (n_features,).

sids

x

scedar.eda.slcs

class scedar.eda.slcs.**MDLSingleLabelClassifiedSamples**(*x*, *labs*, *sids=None*, *fids=None*, *encode_type='data'*, *mdl_method=<class 'scedar.eda.mdl.ZeroIGKdeMdl'>*, *d=None*, *metric='correlation'*, *nprocs=None*)
Bases: *scedar.eda.slcs.SingleLabelClassifiedSamples*

MDLSingleLabelClassifiedSamples inherits SingleLabelClassifiedSamples to offer MDL operations.

Parameters

- **x** (*2d number array*) – data matrix
- **labs** (*list of str or int*) – labels
- **sids** (*list of str or int*) – sample ids
- **fids** (*list of str or int*) – feature ids
- **encode_type** (*"auto", "data", or "distance"*) – Type of values to encode. If "auto", encode data when n_features <= 100.
- **mdl_method** (*mdl.Mdl*) – If None, use ZeroIGKdeMdl for encoded values with >= 50% zeros, and use GKdeMdl otherwise.
- **d** (*2d number array*) – distance matrix
- **metric** (*str*) – distance metric for scipy
- **nprocs** (*int*) –

_mdl_method

Type *mdl.Mdl*

class LabMdlResult(*ulab_mdl_sum*, *ulab_s_inds*, *ulab_cnts*, *ulab_mdls*, *cluster_mdl*)
Bases: tuple

cluster_mdl
Alias for field number 4

ulab_cnts
Alias for field number 2

ulab_mdl_sum
Alias for field number 0

ulab_mdls
Alias for field number 3

ulab_s_inds
Alias for field number 1

static compute_cluster_mdl(*labs*, *cl_mdl_scale_factor=1*)
Additional MDL for encoding the cluster

- labels are encoded by multinomial distribution
- parameters are encoded by 32bit float np.log(2**32) = 22.18070977791825
- scaled by factor

TODO: formalize parameter mdl

encode(*qx*, *col_summary_func=<built-in function sum>*, *non_zero_only=False*, *nprocs=1*, *verbose=False*)
Encode input array qx with fitted code without label

Parameters

- **qx** (*2d np number array*) –
- **col_summary_func** (*callable*) – function applied on column mdls
- **non_zero_only** (*bool*) – whether to encode non-zero entries only
- **nprocs** (*int*) –
- **verbose** (*bool*) –

Returns mdl for encoding qx

Return type float

lab_mdl(*cl_mdl_scale_factor=1*, *nprocs=1*, *verbose=False*, *ret_internal=False*)
Compute mdl of each feature after separating samples by labels

Parameters

- **cl_mdl_scale_factor** (*float*) – multiplies cluster related mdl by this number
- **nprocs** (*int*) –
- **verbose** (*bool*) – Not implemented

Returns mdl of matrix after separating sampels by labels

Return type float

no_lab_mdl(*nprocs=1*, *verbose=False*)
Compute mdl of each feature without separating samples by labels

Parameters

- **nprocs** (*int*) –
- **verbose** (*bool*) – Not implemented

Returns mdl of matrix without separating samples by labels

Return type float

static per_col_encoders(*x*, *encode_type*, *mdl_method=<class 'scedar.eda.mdl.ZeroIGKdeMdl'>*, *nprocs=1*, *verbose=False*)

Compute mdl encoder for each column

Parameters

- **x** (*2d number array*) –
- **encode_type** (*"data" or "distance"*) –
- **mdl_method** (mdl.Mdl) –
- **nprocs** (*int*) –
- **verbose** (*bool*) –

Returns obj: list of column mdl encoders of x

class scedar.eda.slcs.**SingleLabelClassifiedSamples**(*x*, *labs*, *sids=None*, *fids=None*, *d=None*, *metric='cosine'*, *use_pdist=True*, *nprocs=None*)

Bases: *scedar.eda.sdm.SampleDistanceMatrix*

Data structure of single label classified samples

_x

(n_samples, n_features) data matrix.

Type 2D number array

_d

(n_samples, n_samples) distance matrix.

Type 2D number array

_labs

list of labels in the same type, int or str.

Type list of labels

_fids

list of feature IDs in the same type, int or str.

Type list of feature IDs

_sids

list of sample IDs in the same type, int or str.

Type list of sample IDs

_metric

Distance metric.

Type str

Note: If sort by labels, the samples will be reordered, so that samples from left to right are from one label to another.

---

cross_labs(*q_slc_samples*)

dmat_heatmap(*selected_labels=None*, *col_labels=None*, *transform=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(10, 10)*, ***kwargs*)

Plot distance matrix with rows colored by current labels.

feature_importance_across_labs(*selected_labs*, *test_size=0.3*, *num_boost_round=10*, *nprocs=1*, *random_state=None*, *silent=1*, *xgb_params=None*, *num_bootstrap_round=0*, *bootstrap_size=None*, *shuffle_features=False*)

Use xgboost to determine the importance of features determining the difference between samples with different labels.

Run cross validation on dataset and obtain import features.

Parameters

- **selected_labs** (*label list*) – Labels to compare using xgboost.
- **test_size** (*float in range (0, 1)*) – Ratio of samples to be used for testing
- **num_bootstrap_round** (*int*) – Do num_bootstrap_round times of simple bootstrapping if *num_bootstrap_round* > *0*.
- **bootstrap_size** (*int*) – The number of samples for each bootstrapping run.
- **shuffle_features** (*bool*) –
- **num_boost_round** (*int*) – The number of rounds for xgboost training.
- **random_state** (*int*) –
- **nprocs** (*int*) –
- **xgb_params** (*dict*) – Parameters for xgboost run. If None, default will be used. If provided, they will be directly used for xgbooster.

Returns

- **feature_importance_list** (*list of feature importance of each run*) – [(feature_id, mean of fscore across all bootstrapping rounds, standard deviation of fscore across all bootstrapping rounds, number of times used all bootstrapping rounds), . . .]
- **bst_list** (*list of xgb Booster*) – Fitted boost tree model

Notes

If multiple features are highly correlated, they may not all show up in the resulting tree. You could try to reduce redundant features first before comparing different clusters, or you could also interpret the important features further after obtaining the important features.

For details about xgboost parameters, check the following links:

[1] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[2] http://xgboost.readthedocs.io/en/latest/python/python_intro.html

[3] http://xgboost.readthedocs.io/en/latest/parameter.html

[4] https://xgboost.readthedocs.io/en/latest/how_to/param_tuning.html

[5] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

feature_importance_distintuishing_labs(*selected_labs*, *test_size=0.3*, *num_boost_round=10*, *nprocs=1*, *random_state=None*, *silent=1*, *xgb_params=None*, *num_bootstrap_round=0*, *bootstrap_size=None*, *shuffle_features=False*)

Use xgboost to compare selected labels and others.

feature_importance_each_lab(*test_size=0.3*, *num_boost_round=10*, *nprocs=1*, *random_state=None*, *silent=1*, *xgb_params=None*, *num_bootstrap_round=0*, *bootstrap_size=None*, *shuffle_features=False*)
Use xgboost to compare each label with others. Experimental.

feature_swarm_plot(*fid*, *transform=None*, *labels=None*, *selected_labels=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(10, 10)*)

filter_min_class_n(*min_class_n*)

id_x(*selected_sids=None*, *selected_fids=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_s_inds** (*int array*) – Index array of selected samples. If is None, select all.
- **selected_f_inds** (*int array*) – Index array of selected features. If is None, select all.

Returns subset

Return type *SingleLabelClassifiedSamples*

ind_x(*selected_s_inds=None*, *selected_f_inds=None*)
Subset samples by (sample IDs, feature IDs).

Parameters

- **selected_s_inds** (*int array*) – Index array of selected samples. If is None, select all.
- **selected_f_inds** (*int array*) – Index array of selected features. If is None, select all.

Returns subset

Return type *SingleLabelClassifiedSamples*

lab_sorted_sids(*ref_sid_order=None*)

lab_x(*selected_labs*)

lab_x_bool_inds(*selected_labs*)

labs

labs_to_sids(*labs*)

merge_labels(*orig_labs_to_merge*, *new_lab*)
Merge selected labels into a new label

Parameters

- **orig_labs_to_merge** (*list of unique labels*) – original labels to be merged into a new label
- **new_lab** (*label*) – new label of the merged labels

Returns None

Note: Update labels in place.

---

relabel(*labels*)
Return a new SingleLabelClassifiedSamples with new labels.

static select_labs_bool_inds(*ref_labs*, *selected_labs*)

sids_to_labs(*sids*)

sort_by_labels()
Return a copy with sorted sample indices by labels and distances.

tsne_feature_gradient_plot(*fid*, *transform=None*, *labels=None*, *selected_labels=None*, *shuffle_label_colors=False*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last t-SNE projection with the provided gradient as color.

Parameters

- **fid** (*feature id scalar*) – ID of the feature to be used for gradient plot.
- **transform** (*callable*) – Map transform on feature before plotting.

tsne_plot(*gradient=None*, *labels=None*, *selected_labels=None*, *shuffle_label_colors=False*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(20, 20)*, *add_legend=True*, *n_txt_per_cluster=3*, *alpha=1*, *s=0.5*, *random_state=None*, ***kwargs*)
Plot the last t-SNE projection with the provided gradient as color.

xmat_heatmap(*selected_labels=None*, *selected_fids=None*, *col_labels=None*, *transform=None*, *title=None*, *xlab=None*, *ylab=None*, *figsize=(10, 10)*, ***kwargs*)
Plot x as heatmap.

scedar.eda.stats

scedar.eda.stats.**bidir_ReLU**(*x*, *start*, *end*, *lb=0*, *ub=1*)

scedar.eda.stats.**gc1d**(*x*)
Compute Gini Index for 1D array.

References

[1] http://mathworld.wolfram.com/GiniCoefficient.html

[2] Damgaard, C. and Weiner, J. "Describing Inequality in Plant Size or Fecundity." Ecology 81, 1139-1142, 2000.

[3] Dixon, P. M.; Weiner, J.; Mitchell-Olds, T.; and Woodley, R. "Bootstrapping the Gini Coefficient of Inequality." Ecology 68, 1548-1551, 1987.

[4] Dixon, P. M.; Weiner, J.; Mitchell-Olds, T.; and Woodley, R. "Erratum to 'Bootstrapping the Gini Coefficient of Inequality.' " Ecology 69, 1307, 1988.

[5] https://en.wikipedia.org/wiki/Gini_coefficient

[6] https://github.com/oliviaguest/gini/blob/master/gini.py

scedar.eda.stats.**multiple_testing_correction**(*pvalues*, *correction_type='FDR'*)
Consistent with R.

correct_pvalues_for_multiple_testing([0.0, 0.01, 0.029, 0.03, 0.031, 0.05, 0.069, 0.07, 0.071, 0.09, 0.1])

scedar.knn

scedar.knn.detection

class scedar.knn.detection.**RareSampleDetection**(*sdm*)
Bases: object

K nearest neighbor detection of rare samples

Perform the rare sample detection procedure in parallel, with each combination of parameters as a process. Because this procedure runs iteratively, parallelizing each individual parameter combination run is not implemented.

Stores the results for further lookup.

Parameters **sdm** (*SampleDistanceMatrix or its subclass*) –

_sdm

Type *SampleDistanceMatrix*

_res_lut
lookup table of KNN rare sample detection results

Type dict

detect_rare_samples(*k*, *d_cutoff*, *n_iter*, *nprocs=1*, *metric=None*, *use_pca=False*, *use_hnsw=False*, *index_params=None*, *query_params=None*)
KNN rare sample detection with multiple parameter combinations

Assuming that there are at least k samples look similar in this dataset, the samples with less than k similar neighbors may be rare. The rare samples can either be really distinct from the general populaton or caused by technical errors.

This procedure iteratively detects samples according to their k-th nearest neighbors. The samples most distinct from its k-th nearest neighbors are detected first. Then, the left samples are detected by less stringent distance cutoff. The distance cutoff decreases linearly from maximum distance to d_cutoff with n_iter iterations.

Parameters

- **k** (*int list or scalar*) – K nearest neighbors to detect rare samples.
- **d_cutoff** (*float list or scalar*) – Samples with >= d_cutoff distances are distinct from each other. Minimum (>=) distance to be called as rare.
- **n_iter** (*int list or scalar*) – N progressive iNN detections on the dataset.
- **metric** (*{'cosine', 'euclidean', None}*) – If none, self._sdm._metric is used.
- **use_pca** (*bool*) – Use PCA for nearest neighbors or not.
- **use_hnsw** (*bool*) – Use Hierarchical Navigable Small World graph to compute approximate nearest neighbor.
- **index_params** (*dict*) – Parameters used by HNSW in indexing.

  efConstruction: int Default 100. Higher value improves the quality of a constructed graph and leads to higher accuracy of search. However this also leads to longer indexing times. The reasonable range of values is 100-2000.

  M: int Default 5. Higher value leads to better recall and shorter retrieval times, at the expense of longer indexing time. The reasonable range of values is 5-100.

delaunay_type: {0, 1, 2, 3} Default 2. Pruning heuristic, which affects the trade-off between retrieval performance and indexing time. The default is usually quite good.

post: {0, 1, 2} Default 0. The amount and type of postprocessing applied to the constructed graph. 0 means no processing. 2 means more processing.

indexThreadQty: int Default self._nprocs. The number of threads used.

- **query_params** (*dict*) – Parameters used by HNSW in querying.

  efSearch: int Default 100. Higher value improves recall at the expense of longer retrieval time. The reasonable range of values is 100-2000.

- **nprocs** (*int*) – N processes to run all parameter tuples.

Returns Indices of non-rare samples of each corresponding parameter tuple.

Return type res_list

Notes

If parameters are provided as lists of equal length n, the n corresponding parameter tuples will be executed parallely.

Example:

k = [10, 15, 20]

d_cutoff = [1, 2, 3]

n_iter = [10, 20, 30]

(k, d_cutoff, n_iter) tuples *(10, 1, 10), (15, 2, 20), (20, 3, 30)* will be tried parallely with nprocs.

scedar.knn.imputation

class scedar.knn.imputation.**FeatureImputation**(*sdm*)

Bases: object

Impute dropped out features using K nearest neighbors approach

If the value of a feature is below min_present_val in a sample, and all its KNNs have above min_present_val, replace the value with the summary statistic (default is median) of KNN above threshold values.

_sdm

Type *SampleDistanceMatrix*

_res_lut

lookup table of the results. *{(k, n_do, min_present_val, n_iter): (pu_sdm, pu_idc_arr, stats), . . . }*

Type dict

impute_features(*k*, *n_do*, *min_present_val*, *n_iter*, *nprocs=1*, *statistic_fun=<function median>*, *metric=None*, *use_pca=False*, *use_hnsw=False*, *index_params=None*, *query_params=None*, *verbose=False*)

Runs KNN imputation on multiple parameter sets parallely.

Each parameter set will be executed in one process.

Parameters

- **k** (*int*) – Look at k nearest neighbors to decide whether to impute or not.

- **n_do** (*int*) – Minimum (>=) number of above min_present_val neighbors among KNN to be callsed as dropout, so that imputation will be performed.
- **min_present_val** (*float*) – Minimum (>=) values of a feature to be called as present.
- **n_iter** (*int*) – The number of iterations to run.
- **statistic_fun** (*callable*) – The summary statistic used to correct gene dropouts. Default is median.

Returns

resl – list of results, *[(pu_sdm, pu_idc_arr, stats), . . .]*.

pu_sdm: SampleDistanceMatrix SampleDistanceMatrix after imputation

pu_idc_arr: array of shape (n_samples, n_features) Indicator matrix of the ith iteration an entry is being imputed.

stats: str Stats of the run.

Return type list

Notes

If parameters are provided as lists of equal length n, the n corresponding parameter tuples will be executed parallely.

Example

If *k = [10, 15]*, *n_do = [1, 2]*, *min_present_val = [5, 6]*, and *n_iter = [10, 20]*, *(k, n_do, min_present_val, n_iter)* tuples *(10, 1, 5, 10) and (15, 2, 6, 20)* will be tried parallely with nprocs.

n_do, min_present_val, n_iter

scedar.utils

scedar.utils.**dict_str_key**(*d*)

Get a hash key for a dictionary, usually used for ***kwargs*.

Examples

```
>>> dict_str_key({"a": 1, "b": 2})
"[('a', 1), ('b', 2)]"
>>> dict_str_key({"b": 2, "a": 1})
"[('a', 1), ('b', 2)]"
```

Notes

Non-string keys will be converted to strings before sorting, but the original value is preserved in the generated key.

scedar.utils.**load_gz_obj**(*path*)

Load gzipped python object with pickle.load

Parameters **path** (*str*) – The path to the gzipped python object to be loaded.

scedar.utils.**load_obj**(*path*)
Load python object with pickle.load

Parameters **path** (*str*) – The path to the python object to be loaded.

scedar.utils.**parmap**(*f*, *X*, *nprocs=1*)
parmap_fun() and parmap() are adapted from klaus se's post on stackoverflow. https://stackoverflow.com/a/16071616/4638182

parmap allows map on lambda and class static functions.

Fall back to serial map when nprocs=1.

scedar.utils.**remove_constant_features**(*sfm*)
Remove features that are constant across all samples

scedar.utils.**save_obj**(*obj*, *path*)
Save python object with pickle.dump

Parameters

- **obj** (*object*) – The python object to be saved.
- **path** (*str*) – The path to save the python object.

2.4 Changelog

2.4.1 [0.2.0] - Feb 11, 2020

Changed

- The default value of `optimal_ordering` is changed to `False` in `cluster.MIRAC`.

Added

- Support to sparse matrix in core data structures, including `eda.SampleDistanceMatrix`, `eda.SampleFeatureMatrix`, `eda.SingleLabelClassifiedSamples`, and `eda.MDLSingleLabelClassifiedSamples`.
- Option to avoid pairwise distance computation in `eda.SampleDistanceMatrix` constructor by specifying `use_pdist=False`.
- Community clustering, `cluster.Community`.
- Community detection extended MIRAC clustering, `cluster.CommunityMIRAC`.
- Option to build k nearest neighbor graph with approximate nearest neighbor (ANN) search using Hierarchical Navigable Small World (HNSW) graph, in `eda.SampleDistanceMatrix.s_knn_graph`.
- Support to Python 3.7.

2.4.2 [0.1.7] - Jul 2, 2019

Changed

- Fixed `sdm.SampleDistanceMatrix.umap`. Pass parameters to the `UMAP` function call.

2.4.3 [0.1.6] - Jan 2, 2019

Added

- Changelog to keep track of changes in each update.

Changed

- Renamed `qc` module to `knn`, in accordance with the updated manuscript. This change clarifies the meaning of 'quality control' in this package, which is different from its typical meaning. In this package, quality control means to *explore the data according to certain qualities of samples and features*, rather than filtering the raw data according to technical parameters determined by domain specific knowledge.
- Renamed `qc.SampleKNNFilter` to `knn.RareSampleDetection`, in accordance with the updated manuscript. This change clarifies the purpose of this procedure, which is to identify samples distinct from their nearest neighbors *for further inspection rather than removal*. Filtering usually refers to the removal of samples.
- Renamed `qc.FeatureKNNPickUp` to `knn.FeatureImputation`, in accordance with the updated manuscript. This change clarifies the purpose of this procedure in the field of single-cell RNA-seq data analysis, which is to reasonably change zero entries in the data matrix into non-zero entries. This procedure is usually called 'imputation' in the field of single-cell RNA-seq data analysis.
- Moved `qc.remove_constant_features` to `utils.remove_constant_features`.

Python Module Index

S

Index

Symbols

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

T

U

X

Z