binderfinder Documentation

Release 1.52

Gosselink A., Werchau N.

Feb 23, 2018

Contents

Index

CHAPTER 1

Examples

The binderfinder employs two main features: PCA analysis and data reduction of multi dimensional data into an RGB Matrix. In both cases the data needs to be pre-processed into a certain csv format, see Fileformat. For the mapping of multidimensional data into RGB see Evaluation/RGB mapping

1.1 Matrix

```
defaults = dict(filename='./data/iris_dataset/iris.data',
              annotate='none',
                 stats=True,
                  sort='none',
                legend='bg',
                  ceil=True,
             normalize='channels',
             ch_labels=['red', 'Rnd', 'Rnd'],
             )

start_binderfinder(defaults)
```

1.2 PCA

```
params = dict(annotate=False,
              normalize=True,
                covplot=True,
               portions=True,
              )

p = PCA('./data/mock_data_pca_rnd.csv', annotate=True, normalize=False, covplot=True)
p.show()
```

CHAPTER 2

Fileformat

The general fileformat is defined by a header and a data section. The header has the general fromat:

```
properties;n
parameters;m
```

where n is the number of properties and m is the number of parameters. With properties, the a-priori known properties of a sample are known. A property of a sample would translate to a gen modification, treatment e.g. The properties are the label of the data or in other terms: properties are those words, numbers and sequences you'd write onto your falcon tube or eppi, describing your sample.

Leaving us with the parameters m. This is the number of parameters you are measuring. This referres to your number of readouts e.g. channels used in flowcytometry. Optionally the header can have the format:

```
properties;n
parameters;ma;name_1;name_2;...;name_m
```

where name are the description od the measured parameter (e.g. Intensity, weight, velocity,...)

A file for Matrix Analysis would look like this:

```
# header
properties;2
parameters;2;par1;par2
# data
A;a;2.1;0.25
A;b;2.2;0.89
A;c;2.3;0.98
A;d;2.4;0.57
```

Lines starting with an '#' will be ignored. There are two properties defined in the header, thus the first two field in each row are the labels. E.g. in the first data row we would have sample Aa, in the second Ab, Ac,... The following fields are the values, measuerd for the parameter par1, par2,...

A file for PCA would look like this, showing the iris dataset:

```
properties;0
parameters;4;sepal_length;sepal_width;petal_length;petal_width;class
5.1;3.5;1.4;0.2;0
4.9;3.0;1.4;0.2;0
4.7;3.2;1.3;0.2;0
4.6;3.1;1.5;0.2;0
```

Properties needs to be ‘0’ for PCA, thus in the data segment, there are only parameters shown.

CHAPTER 3

Evaluation/RGB mapping

During Matrix creation, upt to thre functions are called. Those functions are externalized for simple access and can be found in *.\binderfinder\evaluate.py*

binderfinder.evaluate.**evaluate**(*params*, *weights*, *refs*)

Calculates RGB mapping.

Functions that gets all measured parameters as stored in the csv file including weights and refs. This function is calld for each sample. Can be replaced.

Parameters

- **params** (*ndarray*) – An array containing all parameters for each sample
- **weights** (*iterable*) – Weights as definded by Matrix(weights)
- **refs** (*iterable*) – Reference values as definded by Matrix(refs)

Returns **rgb** – Tuple of tree floats (r, g, b), where the first index is the intensitiy of red, second intensitiy of green and third intensity of blue.

Return type tuple

Notes

The calculated rgb value is mapped accordingly to the sample form the csv file, the parameters where taken from. A short example would be

```
rgb = params[0], params[1], params[2]
```

Here for each sample in the csv file, the first parameter per row is mapped to the red channel, the second is mpped to the green and the third to the ble channel

```
rgb = np.median(params), np.mean(params), 0
```

Here in the read channel the median of all parameters is colorcoded red and the mean is colorcoded green. Blue is always 0.

binderfinder.evaluate.**stats_calculation**(*datapoints*)

Calculationg of row/colum statistics

The default calculates the mean value of all RGB values in each row/colum

Parameters **datapoints** (*ndarray*) – RGB data as used in the Matrix. Thus, datapoints are the rows/columns of the Matrix. Is a ndarray with the shape (n, 3). It contains the 3 RGB vlaus (axis 1) where n (axis 0) is the number of tiles/fields in the respective row/column.

Returns **rgb** – Statistical RGB value representing the row/column datapoints was taken from.

Return type tuple

binderfinder.evaluate.**sort_reduction**(*datapoints*)

Reduces RGB Value to scalar.

Sortig relays on ordering scalar values. As the statistics return RGB values (vector with three scalars). The data must be reduced. Per default, the mean value of all channels is calculated -> (r + g + b) / 3

Parameters

- **datapoints** (*ndarray*) – RGB data as calculated for the statistics rows/columns Is a ndarray with the shape (n, 3). It contains the 3 RGB vlaus (axis 1) where n (axis 0) is the number of tiles/fields statistics row/column
- **Returns** (*iterable*) – Iterable with the length of the row/column entries. If (n, 3) datapoints are given, an iterable with (n, 1)=(n,) must be returned.

CHAPTER 4

API Reference

binderfinder.**start_binderfinder**(*defaults*)

Starts the binderfinder Matrix.

Takes the same Parameters *binderfinder.Matrix* takes. For reference of the parameters please see

class binderfinder.**Matrix**(*filename, reference=[0.0, 0.0], weights=[1.0, 1.0], annotate='none', stats=False, sort='none', legend='', ceil=False, normalize='total', debug=False, cmap='grey', figsize=[10, 5], ch_labels=['red', 'green', 'blue'], legend_font={'color': 'w', 'size': 'x-small'}*)

Main class for matrix visualisation

Evaluates a csv file containing arbitary values measured for any given combination of up to two subtypes. Evaluation is done by mapping the parameters to the RGB space.

Parameters

- **filename** (*path*) – path to csv file, containing parsable data
- **reference** (*iterable*) – reference values as iterable which are passed to the evaluate function.
- **weight** (*iterable*) – weights as iterable which are passed to the evaluate function.
- **annotate** (*{'none', 'data', 'all'}*) –

  'none': Tiles in Matrix are not labeled.

  'data': Only tiles referencing to datapoints from the inputfile are labeled.

  'all': Data and statistics tiles are labeled. Additionally the used RGB values are shown in the tiles.

- **stats** (*{True, False}*) – Show per row/col statistics. The calculations are defined in elvaluate.py/stats_calculation().
- **sort** (*{'none', 'row', 'col', 'both'}*) –

  'none': No sorting. For sorting *stats* needs to be True

  'row': Sort matrix according to row statistics.

'col': Sort matrix according to column statistics.

'both': First sort by row statistics and afterwards by column statistics

- **legend** (*{'rb', 'br', 'rg', 'gr', 'gb', 'bg'}*) – Defines the legend behaviour. Needs to be a string of two chars, the chars need to be 'r', 'g' or 'b'. The first char defines the color plotted along the x-axis, the second char defines the color along the y-axis (default='bg').
- **ceil** (*{True, False}*) – Round the values for the matrix. The scalar data is categorized in decades, changing the readout to 0-10 %, 11-20 %, 21-30 %, and so on. Reduces the dynamic range and as a leads to a loss of information but increase of comparability.
- **normalize** (*{'total', 'channels'}*) –

  'total': All channels are normalized by the overal, maximum value in the matrix.

  'channels': Each channel is normalized the maximum value in the respective channel.
- **ch_lablels** (*list of strings*) – Name displayed for the RGB channels

Notes

The csv file needs to have the following layout:

properties	;	n				
parameter	;	m				
A	;	x	;	v0_Ax	;	v1_Ax
A	;	y	;	v0_Ay	;	v1_Ay
A	;	x	;	v0_Bx	;	v1_Bx
A	;	y	;	v0_By	;	v1_By

where A and B are the maintypes, x and y are the subtypes with the respective measured valus v0 and v1 for each kombination of A, B and x, y. Of course subtype and maintype are interchangeable, as long as the data is formated as described.

run(*show=False*)

Runs the transformation and the Matrix creation.

Parameters **show** (*{True, False}*) –

True: Plot and show the matrix directly after computation.

False: Only calculate, but do not plot the matrix

save_last_run()

Saves the calculated Matrix and transfomration

Saves the matrix as png image and the transformed data at csv file. The files are written into the directory of the data csv file used for the Matrix calculation.

class binderfinder.**PCA**(*filename*, *centering='mean'*, *normalize=False*, *reduce_to=-1*, *figsize=()*, *debug=False*, *portions=True*, *annotate=False*, *covplot=False*, *last_col='data'*, *show_class=()*)

filename: path to csv file, containing parsable data

centering: centering data by 'mean', 'median' or 'none' centering at all

normalize: normalize each parameter independently prior PCA

reduce_to: number of components the data is reduced to. If -1 all components are used

annotate: annotate datapoint in scatterplot. where 0 is the first data item in the data file

portions: plot portion of variance for each component

Index