zava

Release 0.0.2

Jee Vang, Ph.D.

Dec 16, 2020

CONTENTS

`zava` rocks!

Massive and high-dimensional numerical (or continuous) data may visualized using parallel coordinates. For a technical discussion of parallel coordinates see [Weg90]. In parallel coordinates, axes are drawn parallel to one another (*as opposed to drawn orthogonal to one another*). A vector (or row) of data, $(x_1, x_2, \ldots, x_n)$, is plotted by drawing x_1 on `axis 1`, x_2 on `axis 2`, and so on through x_n on `axis n`. The plotted points are joined by a broken line. The use of parallel coordinates to visualize massive and high-dimensional data is often a first step in exploratory data analysis `EDA` where one may wish to visually identify patterns, clusters, or outliers. Towards the purpose of EDA, a generalized rotation of the coordinate axes in high-dimensional space, referred to as the `Grand Tour` [EJW02], may be used in combination with hue and saturation brushing techniques [EJW96].

CHAPTER

ONE

QUICKSTART

1.1 Installation

To install `zava` from `pypi`, use `pip`.

```
pip install zava
```

1.2 Basic Usage

1.2.1 Data

Everything in zava starts with data. Your data should be either a 2-dimensional `numpy` array (ndarray) or a `pandas` dataframe. If you are using a pandas dataframe, the axis will be labeled according to the dataframe column names; otherwise, you get generic axis names.

```
import numpy as np
import pandas as pd

# you can use this numpy array
M = np.array([
    [1, 1, 1, 1],
    [2, 2, 2, 1],
    [3, 3, 3, 3],
    [1, 2, 3, 4],
    [2, 2, 1, 1],
    [1, 1, 3, 3]
])

# or you can convert the array to a pandas dataframe
columns = ['v0', 'v1', 'v2', 'v3']
M = pd.DataFrame(M, columns=columns)
```

1.2.2 Grand Tour

You can then proceed to create a `GrandTour` instance passing in the data. Note the parameters `c` and `d` which are to control the scaling of your data. Since the variables in your data may be on different scale, this normalization is required to bring all of them into the same range for plotting with parallel coordinates.

```
from zava.core import GrandTour

c = 0.0
d = 100.0

grand_tour = GrandTour(M, c, d)
```

1.2.3 Rotations

With the `GrandTour` instance, you can invoke the `rotate()` method to get the rotated data. If your data is huge, you most likely do **NOT** want to do this operation as shown below, as it will store 360 matrices (you do not even want to do this operation, it's just here for illustration purpose on how to get the rotated data).

```
R = [grand_tour.rotate(degree) for degree in range(361)]
```

1.2.4 Visualization

Most likely, you will want to rotate your data and visualize each transformation at a time. Below is a simple example of what you can do with `matplotlib` just visualizing one rotation.

```
import matplotlib.pyplot as plt

# rotates the data by 1 degree
S = grand_tour.rotate(1)

# start setting up plot with matplotlib
fig, ax = plt.subplots(figsize=(15, 3))

# note that S is a pandas dataframe
# we can use S to make line plots that mimics parallel coordinates
params = {
    'kind': 'line',
    'ax': ax,
    'color': 'r',
    'marker': 'h',
    'markeredgewidth': 1,
    'markersize': 5,
    'linewidth': 0.8
}
_ = S.plot(**params))

# some additional plotting configurations/manipulations
_ = ax.get_legend().remove()
_ = ax.xaxis.set_major_locator(plt.MaxNLocator(S.shape[0]))
_ = ax.get_yaxis().set_ticks([])
_ = ax.set_title('Grand Tour')
```

Later, we will look at how to use zava in a Jupyter notebook.

1.2.5 Animations

Below is a full example of how to use zava to create and save the animation. You should have ffmpeg installed and in your path to get this example to work since `matplotlib` relies on ffmpeg to create the video.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import animation

from zava.core import GrandTour
from zava.plot import SinglePlotter, MultiPlotter

# 1. create or get your data
columns = ['v0', 'v1', 'v2', 'v3']

M1 = np.array([
    [1, 1, 1, 1],
    [2, 2, 2, 1],
    [3, 3, 3, 3]
])
M2 = np.array([
    [1, 2, 3, 4],
    [2, 2, 1, 1],
    [1, 1, 3, 3]
])

M1 = pd.DataFrame(M1, columns=columns)
M2 = pd.DataFrame(M2, columns=columns)

# 2. create your GrandTour instances
c = 0.0
d = 100.0

gt1 = GrandTour(M1, c, d)
gt2 = GrandTour(M2, c, d)

# 3. create your plotters for each GrandTour instance
# Note how the first dataset will have red lines
# and the second dataset will have green lines.
# The parameters passed in go into drawing the lines
# and will help create powerful parallel coordinate with
# grand tour visuals with hue and saturation effects.
sp1 = SinglePlotter(gt1, params={'color': 'r'})
sp2 = SinglePlotter(gt2, params={'color': 'g'})

# 4. setup plotting and animation

# don't forget to disable warnings and set the style
plt.rcParams.update({'figure.max_open_warning': 0})
plt.style.use('ggplot')

# Note how we use MultiPlotter to plot both datasets?
fig, ax = plt.subplots(figsize=(15, 3))
mp = MultiPlotter([sp1, sp2], ax=ax)

# matplotlib.animation will help us create animations
params = {
```

(continues on next page)

(continued from previous page)

```
    'fig': fig,
    'func': mp,
    'frames': np.linspace(0, 360, 360),
    'interval': 20,
    'init_func': mp.init
}
anim = animation.FuncAnimation(**params)

plt.close(fig)

# 5. save the animation
params = {
    'filename': 'test.mov',
    'dpi': 500,
    'progress_callback': lambda i, n: print(f'Saving frame {i} of {n}'),
    'metadata': {
        'title': 'Parallel Coordinates with Grand Tour',
        'artist': 'Clint Eastwood',
        'genre': 'Action',
        'subject': 'Exploratory data visualization',
        'copyright': '2020',
        'comment': 'One-Off Coder'
    }
}
anim.save(**params)
```

Your animation video will look like the following. If you have great tips on how to customize animations with matplotlib, please let us know!

There is a lot of known issues with *ffmpeg* and *matplotlib*. You could also try saving the visualization as an animated gif.

```
# set up your MultiPlotter as before
plt.rcParams.update({'figure.max_open_warning': 0})
plt.style.use('ggplot')

# note how we do not pass in an axis?
mp = MultiPlotter([sp1, sp2], ax=None)

# save
# you have to play around with the duration parameter to get smoothness
mp.save_gif('test.gif', duration=0.0001, start=0, stop=180)
```

1.2.6 Considerations

It might not be a good idea to plot **ALL** your data due to computation and memory limitations. You might want to sample your data instead and plot that subset. Even with the simple, made-up data in this running example, creating a whole animation was intensive (laptop fans start to crank up).

CHAPTER TWO

JUPYTER

To get these examples to work in `Jupyter`, you will need to install the following.

- ipywidgets
- ffmpeg

2.1 Widgets

Let's see how we can use `zava` to work with `ipywidgets`. First, we got to get some data.

[1]:
```
import numpy as np
import pandas as pd

M = np.array([
    [1, 1, 1, 1],
    [2, 2, 2, 1],
    [3, 3, 3, 3],
    [1, 2, 3, 4],
    [2, 2, 1, 1],
    [1, 1, 3, 3]
])
```

Now we create an instance of `GrandTour` with the data and also specifying the minimum `c` and maximum `d` values for scaling.

[2]:
```
from zava.core import GrandTour

c = 0
d = 1
grand_tour = GrandTour(M, c, d)
```

Finally, we use a function `f` annotated with `@interact` to create an interactive visualization with parallel coordinates and Grand Tour.

[3]:
```
import matplotlib.pyplot as plt
from ipywidgets import interact

@interact(degree=(0, 360 * 4, 0.5))
def f(degree=0):
    S = grand_tour.rotate(degree)

    fig, ax = plt.subplots(figsize=(15, 3))
```

(continues on next page)

(continued from previous page)

```
    params = {
        'kind': 'line',
        'ax': ax,
        'color': 'r',
        'marker': 'h',
        'markeredgewidth': 1,
        'markersize': 5,
        'linewidth': 0.8
    }

    _ = S.plot(**params)
    _ = ax.get_legend().remove()
    _ = ax.set_xticks(np.arange(len(S.index)))
    _ = ax.set_xticklabels(S.index)
    _ = ax.get_yaxis().set_ticks([])
    _ = ax.set_title('Grand Tour')
```

```
interactive(children=(FloatSlider(value=0.0, description='degree', max=1440.0, step=0.
↪5), Output()), _dom_clas...
```

2.2 Animations

Now let's see how we can create HTML5 animations in a notebook using `matplotlib.animation`. Again, start with some data.

[4]:
```
import numpy as np
import pandas as pd

M = np.array([
    [1, 1, 1, 1],
    [2, 2, 2, 1],
    [3, 3, 3, 3],
    [1, 2, 3, 4],
    [2, 2, 1, 1],
    [1, 1, 3, 3]
])
```

Create a `GrandTour` instance with the data.

[5]:
```
from zava.core import GrandTour

c = 0
d = 1
grand_tour = GrandTour(M, c, d)
```

We have to wrap the `GrandTour` instance with a `SinglePlotter`. The `SinglePlotter` plots only a single set of data with an `axis` and does not concern itself with the greater plot (e.g. the title). The `params` argument is a dictionary that you can override to change the line drawings.

[6]:
```
from zava.plot import SinglePlotter

single_plotter = SinglePlotter(grand_tour, params={'color': 'r'})
```

The `MultiPlotter` controls all the plots and takes in a list of `SinglePlotters` as well as an `axis`. You can then use an instance of this object with `animation.FuncAnimation()` as usual to produce an animation.

[7]:
```
from zava.plot import MultiPlotter
from matplotlib import animation

fig, ax = plt.subplots(figsize=(5, 3))

multi_plotter = MultiPlotter([single_plotter], ax=ax)

params = {
    'fig': fig,
    'func': multi_plotter,
    'frames': np.linspace(0, 360, 360),
    'interval': 20,
    'init_func': multi_plotter.init
}
anim = animation.FuncAnimation(**params)

plt.close(fig)
```

Finally, render the video.

[8]:
```
%%time

from IPython.display import HTML

HTML(anim.to_html5_video())
```

```
CPU times: user 21.4 s, sys: 574 ms, total: 22 s
Wall time: 22.1 s
```

[8]:
```
<IPython.core.display.HTML object>
```

2.3 Animation, colors

You might find yourself doing cluster analysis of high-dimensional data. If you recover some clusters, you can break the data apart according to the clusters and visualize them with different colors. Here's a full working example (without the clustering) of how to visualize two datasets.

[9]:
```
%%time

# 1. here are your two datasets, M1 and M2

columns = ['v0', 'v1', 'v2', 'v3']

M1 = np.array([
    [1, 1, 1, 1],
    [2, 2, 2, 1],
    [3, 3, 3, 3]
])
M2 = np.array([
    [1, 2, 3, 4],
    [2, 2, 1, 1],
    [1, 1, 3, 3]
])
```

(continues on next page)

(continued from previous page)

```
M1 = pd.DataFrame(M1, columns=columns)
M2 = pd.DataFrame(M2, columns=columns)

# 2. create your GrandTour instances

c = 0.0
d = 100.0

gt1 = GrandTour(M1, c, d)
gt2 = GrandTour(M2, c, d)

# 3. create corresponding SinglePlotters

sp1 = SinglePlotter(gt1, params={'color': 'r'})
sp2 = SinglePlotter(gt2, params={'color': 'g'})

# 4. create a MultiPlotter from the SinglePlotters
fig, ax = plt.subplots(figsize=(5, 3))
mp = MultiPlotter([sp1, sp2], ax=ax)

params = {
    'fig': fig,
    'func': mp,
    'frames': np.linspace(0, 360, 360),
    'interval': 20,
    'init_func': mp.init
}
anim = animation.FuncAnimation(**params)

plt.close(fig)

# 5. display the animation
HTML(anim.to_html5_video())
```

```
CPU times: user 25 s, sys: 608 ms, total: 25.6 s
Wall time: 25.7 s
```

[9]:

```
<IPython.core.display.HTML object>
```

CHAPTER

THREE

BIBLIOGRAPHY

CHAPTER

FOUR

ZAVA

4.1 Core

class zava.core.**GrandTour**(*matrix*, *c=0.0*, *d=100.0*)

Bases: `object`

Grand Tour object.

__init__(*matrix*, *c=0.0*, *d=100.0*)

ctor

Parameters

- **matrix** – Pandas dataframe or 2-D numpy ndarray.
- **c** – Minimum value for scaling. Default 0.0.
- **d** – Maximum value for scaling. Default 100.0.

property headers

Gets a list of headers. The variable names or column names if the matrix is a Pandas dataframe; otherwise, a list of generic names $x_0, x_1, \ldots, x_n$ if the matrix is an `ndarray`.

rotate(*degree*, *transpose=True*, *return_dataframe=True*)

Rotates the matrix. When `transpose` and `return_dataframe` are set to `True`, then a transposed Pandas dataframe is returned. You can just issue `df.plot(kind='line')` as a start to get the parallel coordinate plot.

Parameters

- **degree** – Degree.
- **transpose** – Boolean. Default is True.
- **return_dataframe** – Boolean. Default is True.

Returns Pandas dataframe or 2-D numpy ndarray.

zava.core.**__get_givens**(*n*, *deg*)

Computes the Givens rotation matrix based on the specified degree.

Parameters

- **n** – The number of rows and columns.
- **deg** – Degrees.

Returns A Givens rotation matrix (squared, n x n).

`zava.core._rescale(M, C, D)`
Rescales the specified matrix, M, according to the new minimum, C, and maximum, D. C and D should be of the dimension 1 x cols.

- TODO: avoid recomputing A and B, might not be efficient

Parameters

- **M** – Matrix.
- **C** – Vector of new target minimums.
- **D** – Vector of new target maximums.

Returns Matrix.

`zava.core._rotate(M, C, D, deg=0.0)`
Rotates the specified matrix.

Parameters

- **M** – Matrix.
- **C** – Vector of new target minimums.
- **D** – Vector of new target maximums.
- **deg** – Rotation in degrees. Default 0.0.

Returns Matrix (rotated).

4.2 Plotting

`class zava.plot.MultiPlotter(plotters, ax, **kwargs)`
Bases: `object`

Parallel coordinate and Grand Tour plotter for multiple dataset.

`__call__(degree)`
Instance method to produce plot.

Parameters **degree** – Degree.

`__get_gif_frame(degree, figsize)`
Gets a GIF frame.

Parameters

- **degree** – Degree.
- **figsize** – Tuple of figure size (matplotlib).

Returns None.

`__get_gif_frames(start=0, stop=360, figsize=(15, 3))`
Gets a list of GIF frames.

Parameters

- **start** – Start degree.
- **stop** – Stop degree.
- **figsize** – Figure size. Default is (15, 3).

Returns List of frames.

__init__(*plotters*, *ax*, ***kwargs*)
ctor.

Parameters

- **plotters** – List of SinglePlotter.
- **ax** – Plotting axis.
- **kwargs** – Additional arguments (e.g. title for plot).

init()
Initialization.

save_gif(*output*, *duration*, *start=0*, *stop=360*, *figsize=(15, 3)*, *unit='s'*)
Saves the animation as an animated GIF.

Parameters

- **output** – Output path.
- **duration** – Duration per frame.
- **start** – Start degree.
- **stop** – Stop degree.
- **figsize** – Figure size. Default is (15, 3).
- **unit** – Time units. Default is 's' for seconds.

Returns None.

class zava.plot.**SinglePlotter**(*grand_tour*, *params={}*)
Bases: object

Parallel coordinate and Grand Tour plotter for a single dataset.

__call__(*degree*, *ax*)
Instance method that performs rotation and plot.

Parameters

- **degree** – Degree.
- **ax** – Plotting axis.

__init__(*grand_tour*, *params={}*)
ctor.

Parameters

- **grand_tour** – Grand Tour instance.
- **params** – Parameters for line plots.

property grand_tour
Gets the Grand Tour instance.

Returns Grand Tour.

init()
Initialization. Does nothing for now.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SIX

ABOUT

One-Off Coder is an educational, service and product company. Please visit us online to discover how we may help you achieve life-long success in your personal coding career or with your company's business goals and objectives.

-
-
-
-
-
-

CHAPTER

SEVEN

COPYRIGHT

7.1 Documentation

7.2 Software

7.3 Art

CHAPTER EIGHT

CITATION

```
@misc{oneoffcoder_zava_2020,
title={zava, Parallel Coordinates with Grand Tour},
url={https://github.com/oneoffcoder/zava},
author={Jee Vang},
year={2020},
month={Dec}}
```

CHAPTER

NINE

AUTHOR

Jee Vang, Ph.D.

-
-

CHAPTER TEN

HELP

-
-

BIBLIOGRAPHY

[EJW02] J.L. Solka E.J. Wegman. On some mathematics for visualizing high dimensional data. *Sanhkya*, 2002.

[EJW96] Q. Luo E.J. Wegman. High dimensional clustering using parallel coordinates and the grand tour. *Computing Science and Statistics*, 1996.

[Weg90] E.J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 1990.

PYTHON MODULE INDEX

Z

INDEX

Symbols

G

H

I

M

R

S

Z