pybraincompare Documentation

Release 1.0

Vanessa Sochat

Apr 18, 2018

Contents

Semantic and computational comparison methods for brain imaging data, and visualization of outputs. Modules include:

CHAPTER 1

compare

An example scatterplot image comparison, dynamically rendered using python and d3 from two raw statstical brain maps and an atlas image, scatterplot comparison demo is available. A new addition (beta) is a canvas based scatterplot that can render 150K + points.

CHAPTER 2

ontology

This module will let you convert a triples data structure defining relationships in an ontology to a an interactive d3 visualization, ontology tree demo is available. Reverse inference tree also in development.

CHAPTER 3

network

This module will work with visualization of functional connectivity data, connectivity demo is available.

CHAPTER 4

QA for Statistical Maps

This module will generate a web report for a list of statistical maps, demo is available. Much work to be done! Please submit an issue if you have feedback.

This package is maintained by Vanessa Sochat at Stanford University, proud member of Poldracklab. Pybraincompare drives image comparison in NeuroVault and published methods have also been included. Please submit bugs and feature requests on the Github Repo.

Contents:

4.1 Installation

4.1.1 manual Installation

```
git clone https://github.com/vsoch/pybraincompare
cd pybraincompare
sudo python setup.py install
```

4.1.2 pip Installation

```
pip install pybraincompare
```

4.2 Spatial Comparison and Visualization

Spatial image comparison means using a metric to derive a score that represents the similarity of two brain maps based on voxel values. Pybraincompare provides visualization methods (eg, Similarity Search, Atlas SVG) as well as computational ones (Spatial Computation) to perform and visualize this task.

4.2.1 Atlas SVG

You can generate an orthogonal view svg image, colored by region, for a brain atlas. As an example, we will use the MNI atlas provided by the package. These are also the atlases that can be integrated into a *scatterplot comparison<http://vbmis.com/bmi/share/neurovault/scatter_atlas.html>*_, with points colored by the atlas label. See here for the full example below.

```
from pybraincompare.compare import atlas as Atlas
from pybraincompare.mr.datasets import get_mni_atlas

# Color based on atlas labels
atlases = get_mni_atlas(voxdims=["2"])
atlas = atlases["2"]

# We return both paths and entire svg file
atlas.svg["coronal"]
atlas.paths["coronal"]

# Save svg views to file
output_directory = "/home/vanessa/Desktop"
atlas.save_svg(output_directory) # default will save all views generated
atlas.save_svg(output_directory,views=["coronal"]) # specify a custom set
```

4.2.2 Scatterplot Comparison

You can generate an interactive scatterplot (with an optional atlas to color regions by), and a demo is provided, along with this example as **'script<https://github.com/vsoch/pybraincompare/blob/master/example/make_scatter.py>'**_. First import functions to get images and standard brains, generate atlas SVGs, view a static html file from your local machine, and resample images:

```
# Create a scatterplot from two brain images
from pybraincompare.mr.datasets import get_pair_images, get_mni_atlas
from pybraincompare.compare.mrutils import resample_images_ref, get_standard_brain,␣
↪get_standard_mask, do_mask
from pybraincompare.compare.maths import calculate_correlation
from pybraincompare.compare import scatterplot, atlas as Atlas
from pybraincompare.template.visual import view
from nilearn.image import resample_img
import numpy
import nibabel
```

The first example shows comparison with nifti image input

```
# SCATTERPLOT COMPARE ---- (with nifti input) ----------------------------------------
↪-----------

# Images that we want to compare - they must be in MNI space
image_names = ["image 1","image 2"]
images = get_pair_images(voxdims=["2","8"])

html_snippet,data_table = scatterplot.scatterplot_compare(images=images,
                                                          image_names=image_names,
                                                          corr_type="pearson")
view(html_snippet)
```

You may also need to use pybraincompare to resample images first, an example is provided below:

```
# RESAMPLING IMAGES ------------------------------------------------------------
↪-----------

# If you use your own standard brain (arg reference) we recommend resampling to 8mm␣
↪voxel
# Here you will get the resampled images and mask returned
reference = nibabel.load(get_standard_brain("FSL"))
images_resamp,ref_resamp = resample_images_ref(images,
                                               reference=reference,
                                               interpolation="continuous",
                                               resample_dim=[8,8,8])
```

Here is an example of an equivalent comparison, but using image vectors as input. This is an ideal solution in the case that you are saving reduced representations of your data. See pybraincompare.mr.transformation to generate such vectors.

```
# SCATTERPLOT COMPARE ---- (with vector input) ----------------------------------
↪------------

# We are also required to provide atlas labels, colors, and correlations, so we␣
↪calculate those in advance
# pybraincompare comes with mni atlas at 2 resolutions, 2mm and 8mm
# 8mm is best resolution for rendering data in browser, 2mm is ideal for svg␣
↪renderings
atlases = get_mni_atlas(voxdims=["8","2"])
atlas = atlases["8"]
atlas_rendering = atlases["2"]
```

This function will return a data frame with image vectors, colors, labels. The images must already be registered / in same space as reference

```
corrs_df = calculate_correlation(images=images_resamp,mask=ref_resamp,atlas=atlas,
↪corr_type="pearson")
```

Option 1: Canvas based, no mouseover of points, will render 100,000's of points. An example is provided. Note that we are specifying an output_directory, and so the result files will be saved there, and you can drop this into a web folder, or serve one locally to view (python -m SimpleHTTPServer 9999).

```
output_directory = "/home/vanessa/Desktop/test"
scatterplot.scatterplot_canvas(image_vector1=corrs_df.INPUT_DATA_ONE,
                           image_vector2=corrs_df.INPUT_DATA_TWO,
                           image_names=image_names,
                           atlas_vector = corrs_df.ATLAS_DATA,
                           atlas_labels=corrs_df.ATLAS_LABELS,
                           atlas_colors=corrs_df.ATLAS_COLORS,
                           output_directory=output_directory)
```

Option 2: D3 based, with mouseover of points, limited sampling of images.

```
html_snippet,data_table = scatterplot.scatterplot_compare_vector(image_vector1=corrs_
↪df.INPUT_DATA_ONE,
                                                                 image_vector2=corrs_df.
↪INPUT_DATA_TWO,
                                                                 image_names=image_names,
                                                                 atlas=atlas_rendering,
                                                                 atlas_vector = corrs_df.
↪ATLAS_DATA,
```

```
                                                                    atlas_labels=corrs_df.
↪ATLAS_LABELS,
                                                                    atlas_colors=corrs_df.
↪ATLAS_COLORS,
                                                                    corr_type="pearson")
view(html_snippet)
```

You can also use your own atlas, and here is how to generate it.

```
# CUSTOM ATLAS ------------------------------------------------------------------
↪----------

# You can specify a custom atlas, including a nifti file and xml file
atlas = Atlas.atlas(atlas_xml="MNI.xml",atlas_file="MNI.nii")
Default slice views are "coronal","axial","sagittal"
```

4.2.3 Similarity Search

Pybraincompare can generate an interactive web interface to show the result of a spatial image comparison. A demo is provided, along with this complete example. This function is intended for embedding in a python-based web framework, such as Flask or Django, but you can run it locally as in the example below by using the "view" function. First, you should import these functions to view an html snippet, and generate the html for the similarity search:

```
# This will generate an interactive web interface to find similar images
from pybraincompare.template.visual import view
from pybraincompare.compare.search import similarity_search
from glob import glob
import pandas
```

You should have some kind of static (png,jpg) image that represents each of your images. This should be a list of images, the order which corresponds to the list of images you wiill give to the similarity search:

```
# Here are pre-generated png images, order should correspond to order of your data
png_paths = glob("/home/vanessa/Packages/vagrant/neurovault/image_data/images/3/*.png
↪")[0:6]
```

The search will filter images for you based on tags, which you should provide. Tags should be a list of lists, each of which is a list of strings that are tags for the images. This list should be the same length as your list of images:

```
# Create a list of tags for each image, same order as data
tags = [["Z","brain"],["Z","brain"],["brain"],["Z","brain"],["Z","brain"],["Other"]]
```

Optionally, you can provide two URLs that will link to a comparison page from the image (compare_url) and to a page with information about the image (image_url). Since this was originally generated for neurovault, the format for both is:

```
prefix/[query_id]/[other_id] # for compare_url and
prefix/[query_id] # for image_url
```

where query_id corresponds to the unique ID of the image (detailed below). Here is what providing the URLs might look like:

```
# compare URL prefix to go from linked images, undefined will link to png image
compare_url = "http://www.neurovault.org/compare" # format will be prefix/[query_id]/
↪[other_id]
image_url = "http://www.neurovault.org/image" # format will be prefix/[other_id]
```

And of course you should provide this list of image ids, a simple list:

```
# IDS that will fill in the url paths above
image_ids = [1,2,3,4,5,6]
```

Next, set up your query image. This is the image that all others are compared to. You need to point to a particular png image path (the static image of the query) as well as one of the image IDs from image_ids above:

```
# Here is the query png image and ID
query_png = png_paths[0]
query_id = image_ids[0]
```

You can optionally add bottom text and top text, meaning text that will display above and below each image in the query interface:

```
image_names = ["image 1","image 2","image 3","image 4","image 5","image 6"]
collection_names = ["collection 1","collection 1","collection 1","collection 1",
↪"collection 1","collection 2"]
```

It's up to you to calculate your comparison scores. This page includes examples of doing so with pybraincompare.

```
# Calculate your scores however you wish
image_scores = [1,0.87,0.30,0.2,0.9,0.89]
```

Finally, we put all of the inputs above into a function that will generate a static html page to display the result. You can serve this locally, or embed into your server html (Django or Flask).

```
# Calculate similarity search, get html
html_snippet = similarity_search(image_scores=image_scores,tags=tags,png_paths=png_
↪paths,
                                 button_url=compare_url,image_url=image_url,query_
↪png=query_png,
                                 query_id=query_id,bottom_text=collection_names,
                                 top_text=image_names,image_ids=image_ids)
```

Here is how to view the snippet locally from an interactive console, if you don't have a server:

```
# Show in browser
view(html_snippet)
```

As an option, you can remove script tags. You can take a look at the templates in pybraincompare.templates.html (corresponding to the folder pybraincompare/templates/html (view on github) to see the names of tags before different scripts and files. For example, here is a tag:

```
<!--[FONT_AWESOME]--><link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
↪font-awesome/4.3.0/css/font-awesome.min.css">
```

and you can remove it by specifying:

```
remove_scripts = ["FONT_AWESOME"]
```

This is important if your server already has a particular resource, but keep in mind there may be bugs depending on differences in versions, etc. Here is a full example for removing script tags:

```
# You can also remove JQUERY or BOOTSTRAP, if you are embedding in a page that
↪already has.
remove_scripts = ["JQUERY","BOOTSTRAP"]
html_snippet = similarity_search(image_scores=image_scores,tags=tags,png_paths=png_
↪paths,
                                 button_url=compare_url,image_url=image_url,query_
↪png=query_png,
                                 query_id=query_id,bottom_text=collection_names,
                                 top_text=image_names,image_ids=image_ids,remove_
↪scripts=remove_scripts)
view(html_snippet)
```

4.2.4 Plot Histogram

These functions have not been properly developed, however here is how to plot a histogram for an input image. This example is also provided as a script:

```
#!/usr/bin/python

from pybraincompare.report.histogram import plot_histogram
from pybraincompare.mr.datasets import get_pair_images

image = get_pair_images()[0]
plot_histogram(image)
```

4.3 Semantic Image Comparison

Semantic image comparison means comparing images by way of texy data that is associated with them, for example meta data, or in the case of functional neuroimaging, using an ontology like the Cognitive Atlas to associate images with contrasts and cognitive concepts. Pybraincompare has several methods for reverse inference that will be better documented with a publication, and for now, here are examples to generate trees of ontology structures.

4.3.1 Ontology Tree

This function will generate an interactive d3 representation of a graph data structure demo and script

```
#!/usr/bin/env python2

from pybraincompare.ontology.tree import named_ontology_tree_from_tsv
from pybraincompare.template.visual import view

# This defines a "mock" ontology for the openfmri proof of concept set - names should
↪be unique
relationship_table = "data/cogatlas_annotate_triples.tsv"

# Create a data structure for d3 visualization and data analysis (no output json,
↪specify with output_json=)
data_structure = named_ontology_tree_from_tsv(relationship_table)
html_snippet = make_ontology_tree_d3(data_structure)

# View in browser!
view(html_snippet)
```

4.3.2 Reverse Inference Tree

This function will generate an interactive d3 representation of a graph data structure with mouseover tags that show a reverse inference score for a node, demo and a script are available.

```
#!/usr/bin/env python2

import pandas as pd
from pybraincompare.ontology.tree import named_ontology_tree_from_tsv, make_ontology_
↪tree_d3
from pybraincompare.template.visual import view

# This defines a "mock" ontology for the openfmri proof of concept set - names should␣
↪be unique
relationship_table = "data/cogatlas_annotate_triples.tsv"

# We want to prepare a dictionary of meta data to correspond to some nodes
reverse_inference_df = pd.read_csv("data/reverse_inference_scores.tsv",sep="\t")
reverse_inferences = dict()
for row in reverse_inference_df.iterrows():
    reverse_inferences[row[1].NODE_GROUP] = row[1].REVERSE_INFERENCE_SCORE

# Create a data structure for d3 visualization and data analysis (no output json,␣
↪specify with output_json=)
data_structure = named_ontology_tree_from_tsv(relationship_table, meta_data = reverse_
↪inferences)
html_snippet = make_reverse_inference_tree_d3(data_structure)

# View in browser!
view(html_snippet)
```

4.4 Additional Tools

4.4.1 QA Report

Pybraincompare can also generate a QA report for a set of images. This is useful to quickly run QA, and then serve the result / send to someone to easily see and understand the data. An example is provided along with this script

```
#!/usr/bin/env python2

from glob import glob
from pybraincompare.report.webreport import run_qa

# Here is a set of 144 images, these are 9 openfmri studies in Neurovault, resampled␣
↪to MNI 2mm
mrs = glob("/home/vanessa/Documents/Work/BRAINMETA/IMAGE_COMPARISON/mr/resampled/*.
↪nii.gz")

# Run quality analysis
outdir = "/home/vanessa/Desktop/test"
run_qa(mrs,software="FSL",html_dir=outdir,voxdim=[2,2,2],outlier_sds=6,investigator=
↪"Vanessa",nonzero_thresh=0.25)

# For large datasets (where memory is an issue) you can set calculate_mean_
↪histogram=False
```

4.4.2 Connectogram

Generate a connectogram d3 visualization from a square connectivity matrix demo and script

```
from pybraincompare.compare.network import connectogram
from pybraincompare.template.visual import view
import pandas

# A square matrix, tab separated file, with row and column names corresponding to␣
↪node names
connectivity_matrix = "data/parcel_matrix.tsv"

parcel_info = pandas.read_csv("data/parcels.csv")
networks = list(parcel_info["Community"])
# should be in format "L-1" for hemisphere (L or R) and group number (1..N)
groups = ["%s-%s" %(parcel_info["Hem"][x],parcel_info["ID"][x]) for x in range(0,
↪parcel_info.shape[0])]

# A threshold value for the connectivity matrix to determine neighbors, eg, a value␣
↪of .95 means we only keep top 5% of positive and negative connections, and the user␣
↪can explore this top percent
threshold = 0.99
html_snippet = connectogram(matrix_file=connectivity_matrix,groups=groups,
↪threshold=threshold,network_names=networks)

view(html_snippet)
```

4.5 pybraincompare

4.5.1 pybraincompare package

Subpackages

pybraincompare.compare package

Submodules

pybraincompare.compare.atlas module

pybraincompare.compare.maths module

pybraincompare.compare.mrutils module

pybraincompare.compare.network module

pybraincompare.compare.scatterplot module

pybraincompare.compare.search module

Module contents

pybraincompare.mr package

Submodules

pybraincompare.mr.datasets module

pybraincompare.mr.transformation module

Module contents

pybraincompare.ontology package

Submodules

pybraincompare.ontology.export module

export.py: part of pybraincompare package Functions to export images and data

pybraincompare.ontology.export.**priors_to_brain_image**(*priors_df*, *output_name*, *mask_image*)

Function to save brain images for each level of priors

pybraincompare.ontology.graph module

pybraincompare.ontology.inference module

pybraincompare.ontology.tree module

Module contents

pybraincompare.report package

Submodules

pybraincompare.report.animate module

pybraincompare.report.animate.**animate_figure**()
Animate subplots from current matplotlib figure

pybraincompare.report.colors module

colors.py: part of pybraincompare package Color stuffs

pybraincompare.report.colors.**get_colors**(*N*, *color_format='decimal'*)
Get colors that I like

pybraincompare.report.colors.**peterson_roi_labels**(*colors=True*)
Colors for Peterson ROI labels

pybraincompare.report.colors.**random_colors**(*N*)
Generate N random colors

pybraincompare.report.histogram module

pybraincompare.report.image module

image.py: part of pybraincompare package Functions for static images

pybraincompare.report.image.**make_anat_image**(*nifti_file*, *png_img_file=None*)
Make anat image

pybraincompare.report.image.**make_glassbrain_image**(*nifti_file*, *png_img_file=None*)
Make glassbrain image, optional save image to png file (not vector)

pybraincompare.report.image.**make_roi_image**(*nifti_file*, *png_img_file=None*)
Make roi (mask) image

pybraincompare.report.image.**make_stat_image**(*nifti_file*, *png_img_file=None*)
Make statmap image

pybraincompare.report.image.**plot_vline**(*cur_val*, *label*, *ax*)
from chrisfilo https://github.com/chrisfilo/mriqc

pybraincompare.report.qa module

qa.py: part of pybraincompare package Functions to check quality of statistical maps

pybraincompare.report.qa.**central_tendency**(*data*)
Central tendency: standard measures of central tendency and variance

pybraincompare.report.qa.**count_voxels**(*masked_in*, *masked_out*)
Count voxels in and outside the mask

pybraincompare.report.qa.**get_percent_nonzero**(*masked_in*)
Estimate thresholded We basically check to see if number of zero voxels exceeds some

percentage (not thresholded)

pybraincompare.report.qa.**get_voxel_range**(*nii_obj*)
Get the maximum and minimum value in the image

pybraincompare.report.qa.**header_metrics**(*image*)
Metrics: Extract metrics from the header

pybraincompare.report.qa.**is_only_positive**(*nii_obj*)
Are there only positive values?

pybraincompare.report.qa.**is_thresholded**(*nii_obj*, *brain_mask*, *threshold=0.95*)
Is a nifti image thresholded? Adopted from chrisfilo for neurovault Threshold should be the percentage of voxels we want "good" Returns True/False and ratio of good voxels

pybraincompare.report.qa.**outliers**(*masked_data*, *n_std=6*)
Outliers outliers (e.g., more than ~6 SD from the mean, maybe less

depending on the action)

pybraincompare.report.qa.**t_to_z**(*mr*, *dof*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index