neurosynth_ALE

April 24, 2019

Decoding analysis for meta-analysis map from ALE.
Dependencies: Python 3, Neurosynth. Neurosynth dependencies: NumPy/SciPy,pandas,NiBabel,ply,scikit-learn.
2019/04/23 built.
2019/04/24 added decoding of all features.
Chuanji Gao

In [1]:
```
# At anaconda prompt, type: pip install import-ipynb
import import_ipynb
import os
os.chdir(r'F:\UPDATE\RESEARCHBLOG_GAO\dataScience_2_meta_analysis_methods\Meta_Analysis
import pandas as pd
```

In [7]:
```
# Core functionality for managing and accessing data
from neurosynth import Dataset
# Analysis tools for meta-analysis, image decoding, and coactivation analysis
from neurosynth.analysis import meta, decode, network
#from neurosynth import meta, decode, network
```

In [8]:
```
dataset = Dataset.load(r'F:\UPDATE\RESEARCHBLOG_GAO\dataScience_2_meta_analysis_method
```

Inputs: ALE Image (foci_av_emotionSPMfinal_ALE.nii) is an unthresholded image contains the unthresholded ALE values, one computed at every voxel in the brain. The ALE calculations first create a 3D image for each foci group using the mask, the foci and a guassian blur with a FWHM empirically derived from the subject size. These pre-ALE experiment-level images are called Modeled Activation (MA) maps. The MA maps can be calculated by finding the union or the maximum across each focus's Gaussian. Using the maximum limits the effect of an experiment with multiple foci very near one another and is referred to as "non-additive" in the preferences. The ALE image is a union of all of the MA maps.

The decoding wil not work unless the input image has the same dimensions as the typicall MNI152 2mm image.

In [10]:
```
# Decode images: simple try
decoder = decode.Decoder(dataset, features=['emotion', 'auditory', 'visual', 'audiovis
data = decoder.decode(['foci_av_emotionSPMfinal_ALE_resliced.nii'], save='decoding_re
```

Understanding term similarity score in Decoder:
The decoder returns map-wise correlation coefficients between the input and the reverse inference

maps.
In other words, it's just pearson correlation between the two vectorized maps, computed over all voxels. I.e., corr(x_1, x_2),where x_1 and x_2 are aligned vectors of voxels values from each of the two maps.

How are voxels with missing values handled?
Attempting to decode maps with relatively few non-zero values (those conservatively corrected for multiple comparisions) will produce biased results (i.e., many coefficients very close to 0). Note that the deliberate introduction of bias is not necessarily a bad thing here, because the alternative is to produce highly variable estimates that will often provide a misleading sense of the robustness of an association. In future, we will provide a user option for handing of 0 values. In general, however, we recommend decoding unthresholded, uncorrected, whole-brain maps whenever possible.

The conclusion needs to be made with caution: "e.g., there is some evidence that our pattern of activation is more consistent with language and motor processes than other kinds of processes".

Note that you probably want to pay attention to the absolute strength of the correlations too, as that can give you an informative sense of how "typically" you results are. It can be interpreted exactly like any other correlation coefficient. Which is to say, they have a clear statistical meaning (1 is identity, -1 is inversion, 0 is independence). E.g., if the single strongest correlation between your map and any of the reverse inference meta-analyses is only, say .06, the implication is probably a) your analysis is underpowered b) you have a task that is tapping some uncommon combination of cognitive processes. However, the implications for any particular application depend entirely on the context. There is no simple answer to whether a value of say, .2 is large or small.

We can't know whether the association between the map input and the neurosynth meta-analysis map is significant or not.

In [ ]:
```
# Decode images: all features,
# if we left the features argument unspecified, the decoder would default to using the
decoder = decode.Decoder(dataset)
data = decoder.decode(['foci_av_emotionSPMfinal_ALE_resliced.nii'], save='decoding_resu
```

In [49]:
```
data = pd.read_csv('decoding_results_ALE_All_Features.txt', sep=",", skiprows=1, heade
data.columns = ["Features", "r"]
```

In [55]:
```
#How many features in total
len(data)
```

Out[55]:
```
3160
```

In [46]:
```
#Display head rows
data.head()
```

Out[46]:

	Features	r
0	001	-0.0481
1	01	-0.0531
2	05	-0.0304
3	05 corrected	0.0071
4	10	-0.0822

In [21]:
```
#Display end rows
data.tail()
```

Out[21]:

	Features	r
3155	young adults	-0.0341
3156	young healthy	-0.0617
3157	younger	-0.0161
3158	younger adults	-0.0307
3159	zone	-0.0378

In [50]:
```
# Sort by correlation values
data=data.sort_values(by=['r'],ascending=False);
```

In [52]:
```
# Add ID column
data.insert(0, 'New_ID', range(1, 1 + len(data)));
```

In [56]:
```
# Display final results - 20 first rows
data.head(20)
```

Out[56]:

	New_ID	Features	r
2890	1	temporal sulcus	0.3008
2773	2	sts	0.2828
1098	3	facial	0.2720
2812	4	sulcus sts	0.2540
2878	5	temporal	0.2520
1902	6	neutral	0.2503
170	7	amygdala	0.2375
2820	8	superior temporal	0.2350
1069	9	expressions	0.2343
948	10	emotions	0.2311
277	11	audiovisual	0.2295
938	12	emotion	0.2292
172	13	amygdala hippocampus	0.2277
940	14	emotional	0.2229
1099	15	facial expression	0.2217
1097	16	faces	0.2199
1093	17	face	0.2194
1100	18	facial expressions	0.2165
2300	19	psts	0.2155
3105	20	voice	0.2123