The Impact on Inter-Subject Discriminability of Dimensionality Reduction in fMRI

Eric Bridgeford (ebridge2@jhu.edu), Tanay Agarwal (tagarwa2@jhu.edu)

November 27, 2016

1 Algorithms

Note that these algorithms are not exhaustive, and represent enough detail to demonstrate what's going on without being pure python code.

Algorithm 1: Nuisance Correction

Input:

f $\in R^{x,y,z,t}$: the fMRI timeseries loaded into a matrix.
a $\in R^{x,y,z}$ the anatomical image data.
$a_0 \in R^{x,y,z}$ an integer to indicate the type of anatomical image: 1 for T1w, 2 for T2w, 3 for PD
m $\in R^{x,y,z}$: the binary mask associated with the brain we register to in the registration step.

Result:

n $\in R^{x,y,z,t}$: a nuisance corrected brain in a matrix form.

```
v = f(m, :).T // extract the voxel timeseries over the mask we are registered to and transpose so that time is the 0th dimension
fast f -n 3 -t a_0 // use FSL's FAST to segment the brain into white matter, grey matter, and cerebrospinal fluid probability maps.  each value will indicate a probability of a particular voxel being part of each class (WM vs GM vs CSF).
wm = load_nifti(/path/to/WM/mask) // load the white matter mask into a matrix
wmm = self.extract_mask(wm, .99, 2).T // algorithm, still to be written, that thresholds the white matter mask to get all voxels with p > .99 of being WM, and then erodes by 2 voxels for every point that is considered white matter
wm_ts = f(wmm,:).T // extract the timeseries for the white matter regions over the course of the session
csf = load_nifti(/path/to/WM/mask) // the csf mask in a matrix
csfm = self.extract_mask(csf, .95)
csfm = self.extract_mask(csf, 2)
csf_ts = f(csfm, :)
r = [] // begin an array of regressors, where r ∈ R^{t,n_r}
r(:,0) = 0:t // linear drift regressor
r(:,1) = (0:t)^2 // quadratic drift regressor
r(:,2) = mean(csf_ts, axis=0) // take the mean csf signal over all voxels at every time point
comp = self.compcor(v, wm_ts, c) // Calculate compcor regressors with wm mask, where c is either a fixed number of components or a threshold of variance explained
r(:,3) = comp(0) // the components element returned by compcor method
W = self.regress_signal(v, r) // use the OLS solution for a GLM given regressors
n = (v - W).T // subtract out regressors to remove, giving us our nuisance corrected data and transpose back to standard space
return n
```

Algorithm 2: CompCor

Input:

mask_ts $\in R^{t,n}$ the timeseries extracted over a particular mask that we want to perform component correction for.

c $\in R^{1}$: the number of components to calculate.

Result:

U(:,0:c) $\in R^{t,c}$: the components we will regress out.

```
mask_ts = self.normalize_signal(mask_ts) // detrend and normalize by std
U, s, V = np.linalg.svd(mask_ts, full_matrices=False) // singular value decomposition; use numpy
    here because our data is very high dimensional and dense
return (U(:, 0:c), s)
```

Algorithm 3: Normalize Signal

Input:

v $\in R^{t,n}$ a voxel timeseries.

Result:

v $\in R^{t,n}$ a voxel timeseries that is mean-centered and normalized by standard deviation.

```
v = v(:, std(v, axis=0) != 0) // remove voxels with no standard deviation, since normalizing will
    give them NaN values
v = v - mean(v, axis=0)
v = elementwise_divide(v, std(v, axis=0)) // normalize each voxel by the standard deviation
return v
```

Algorithm 4: Extract Mask

Input:
m $\in R^{t,n}$ a thresholded probability map for a particular class of brain matter (ie, white matter, grey matter, CSF).
t $\in R^1$ a threshold over which to consider a member of that class. Ie, if we define this as .99, that means that all voxels in our probability map with probability $>$ .99 will be considered the class the map represents.

Result:
mask $\in R^{t,n}$ the mask.

```
mask = (m > t) // elements of probability map greater than threshold
saveimg(mask)
return mask
```

Algorithm 5: Extract Mask

Input:
m $\in R^{t,n}$ a thresholded probability map for a particular class of brain matter (ie, white matter, grey matter, CSF).
t $\in R^1$ a threshold over which to consider a member of that class. Ie, if we define this as .99, that means that all voxels in our probability map with probability $>$ .99 will be considered the class the map represents.

Result:
mask $\in R^{t,n}$ the mask.

```
mask = (m > t) // elements of probability map greater than threshold
saveimg(mask)
return mask
```

Algorithm 6: Erode Mask

Input:
m $\in R^{t,n}$ a brain mask to erode.
$n_v \in Z^1$ the number of voxels to erode, where erosion means that wherever we consider a masked segment "part" of a particular class, we also remove this number of voxels in every direction around the masked segment.

Result:
e $\in R^{t,n}$ the eroded mask.

```
for i = 1 : v do
    e = zeros(m.shape)
    x,y,z = idswhere(m! = 0) // get coordinates of masked entries
    for j = 1 : length(coords) do
        if (m[x,y,z]  m[x + 1,y,z]  m[x,y + 1,z]  m[x,y,z + 1]  m[x − 1,y,z]  m[x,y − 1,z]
          m[x,y,z − 1]) then
            eroded[x,y,z] = 1
    end
    m = e
end
```

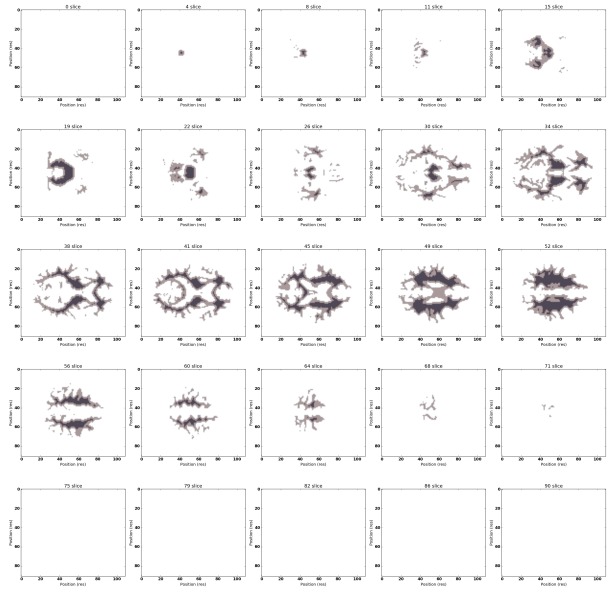
2 Quality Control

As this is part of a processing pipeline that will be used by users of our fMRI web service, a critical part of our job is to provide thorough and rigorous quality assessments of each step in our pipeline. We will add the following quality control:

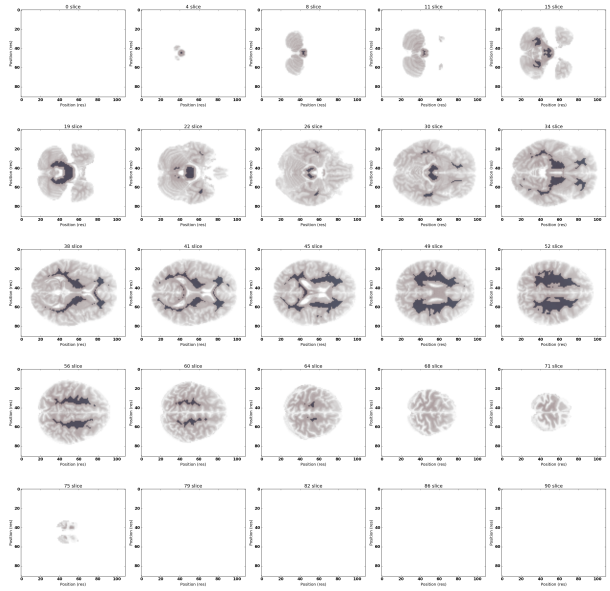
- CompCor: a plot of the expected variance of each component.
- Erode Mask: a figure showing the mask before and after erosion to show that it actually erodes the mask logically, and a figure to show the mask overlaid with the fMRI image to show that it actually

represents what it says it will represent (ie, if a CSF mask is correct, when you view it overlaid with an anatomical image, it should only have a value where the anatomical scan represents CSF).

Eroded WM Mask overlaid with WM Mask Here, we attempt to demonstrate the effectiveness of mask erosion with respect to the uneroded mask. the eroded mask should be roughly 2 voxels smaller in all directions than the uneroded.



Eroded WM Mask overlaid with Anatomical MRI Here, we show the eroded mask overlaid with the anatomical MRI.



.

3 High Level Simulations

A relatively intuitive place to start is to perform a simulation that is similar that that of [1]. A high level explanation of what is happening here is that each voxel in our "non-noise" regions will have a sinusoid

representing its simulated latent signal. Then, we will corrupt all of the voxels (non noise or noise) with an identical signal, representing our "physiological noise". We will then attempt to perform PCA on the resultant timeseries, and measure how effectively we can recover the original signal. We will measure the effectiveness of our algorithm under a variety of conditions, including raw sinusoids (ie, each voxel has a timeseries represented by an identical sinusoid, and then our noise regions have a different sinusoid, and the total signal is the sum/average of the two depending on what works best; look at R^2 as a function of amplitude of the noise region noise). Also, we will consider how zero-mean gaussian noise for our data impacts our ability to recover the signal (ie, corrupt each voxel's timeseries by some zero-mean gaussian noise of varying std, and look at R^2 as a function of std).

4 Real Data Validation

4.1 Pipeline Analysis Method

For the real data, our primary goal is identifying how nuisance impacts the ability to discriminate between the brain signals of individual subjects. Essentially, we will compute connectivity matrices for ROI timeseries using 2 different atlases (desikan and Talairach), which essentially means that we average the voxel timeseries for different labelled regions of voxels (ie, a voxel label of "1" might mean the frontal cortex, so we would average over all voxels in our labeled atlas with a value of "1" for each time point). Using these connectivity matrices, we will compute distance estimates between each pair of scans, and ultimately report the discriminability for each pipeline following the methods outlined in [2]. We will analyze the entirity of the BNU1 dataset using the following pipelines:

- no nuisance correction
- nuisance correction with CompCor using 1 component
- nuisance correction with CompCor using 5 components (expected optimal number, recommended to us by a domain expert in fMRI processing Dr. Cameron Craddock)
- nuisance correction with CompCor using 10 components
- nuisance correction wtih some tbd low threshold of explained variance
- nuisance correction with some tbd high threshold of explained variance

as time allows, we will also analyze the BNU2 dataset, the DC1 dataset, the HNU1 dataset, and the NKI dataset. Additionally, we may also include a pipeline with only frequency filtering (which removes low frequency drift in the sessions). However, this option is incredibly slow, so this will also be as time allows (and would entail writing an additional algorithm to remove drift signals with a frequency below a particular threshold).

4.2 Interpretation

We will report a discriminability value for each pipeline, for each dataset we get to, and show the discriminability as a function of pipeline choice in a scatter plot. The "best" pipeline will be reported as the pipeline with the highest discriminability.

References

[1] Y Behzadi, K Restom, J Liau, and T T Liu. A component based noise correction method (CompCor) for BOLD and perfusion based fMRI. *Neuroimage*.

[2] Shangsi Wang, Zhi Yang, Xi-Nian Zuo, Michael Milham, Cameron Craddock, Gregory Kiar, William Gray Roncal, Eric Bridgeford, Carey E Priebe, and Joshua T Vogelstein. Optimal Decisions for Discovery Science via Maximizing Discriminability: Applications in Neuroimaging. *request for preprint.*