1. Body\_map\_analysis CogSci 6월 1일
2. Markov chain analysis
3. Physio figure 8/15
4. fast\_180424\_SPM\_firstlevel\_model1
5. second level analysis
6. 핵심 Figure를 만든 코드
7. 데이터를 MNI space 상에서 보거나, 수정하거나, 저장하는 과정에 대한 코드
8. HPC 사용법
9. Body\_map\_analysis CogSci 6월 1일

Dir >> Project/FAST/scripts/behavioral/fast\_180521\_body\_map\_analysis\_v4.m

한 참가자마다 160 trial X map 2이 있다. Activation map & deactivation map이다.

160개의 Activation map과 deactivation map을 하나씩 conv\_circle로 convolution을 해준 다음 두 map을 합친다. act\_conv – deact\_conv를 하여, 각 픽셀들은 +1에서 -1 사이의 값을 갖게 된다. 그런 뒤 imgaussfilt를 통해 guassian kernel로 blur해준다. 이후 body\_white\_binary를 이용하여 몸 밖에 칠해진 데이터는 지우고 각 trial을 열로 코딩해준다. 즉 몸 안에 존재하는 픽셀의 수 63303가 160열만큼 존재한다. = [63303x160]

5가지의 survey데이터는 quartile로 4등분해주고 그 데이터를 dat{dim\_i}에 넣는다. dat은 5개의 cell로 이루어져있고, dat{dim\_i}.Y는 [252x]로 이루어져 있는데 한 참가자마다 4행을 사용, 4x63=252. 4행은 4분위를 의미한다. dat{dim\_i}.Y에는 각 4분위에 해당하는 점수들의 평균이 들어있고, dat{dim\_i}.dat에는 각 4분위에 해당하는 body map의 평균 데이터가 각 행에 들어있다. dat{dim\_i}.dat의 크기는 [63303×252 double]이며 열은 63명 x 4분위의 개수이다.

whfolds는 참가자의 subject number에 대한 인덱스로 [1;1;1;1;2;2;2;2;3…]이렇게 63x4 행이 5개의 dimension마다 들어있다. 하지만 body map이 없는 빈 trial의 경우에는 dat{dim\_i}.Y와 whfolds{dim\_i}와 dat{dim\_i}.dat에서 해당 trial을 지워준다. 이것은 survey의 dimension마다 달라질 수 있으므로 whfolds에서 train과 test를 분리한 두 변수 whfolds\_test와 whfolds\_train은 각각 5 cell로 이루어져 있다.

Train데이터와 test data를 분리하고, train data를 대상으로 cv\_pcr을 돌린다.

중요한 결과는 stats\_train에 정리. stats\_train은 survey dimension에 해당하는 5 cell로 이루어져 있고, 각 cell안에는 pcr을 돌린 component의 수의 경우의 수만큼 struct이 들어있다. 그 struct 안에는 pred\_outcome\_r 도 있다. 이것은 참가자와 상관없이 각 dimension에서 구해지는 상관계수이다.

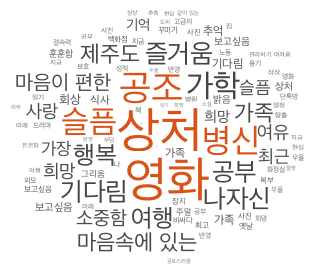
Test set에 적용하기 전에 prediction performance를 확인.  
stats\_train{dim\_i}{component#}.Y는 실제 survey data이며, 원래의 데이터는 train 42명 x 4분위 = 168x1 double이 들어있어야 하는데 NaN 데이터가 삭제되어 그보다 작은 수가 들어간다. 그 수에 맞게, stats\_train{dim\_i}{component#}.yfit이 생기는데 이 변수는 train을 통해 예측된 survey 값이다. 이 둘을 한 변수에 합치고 correlation을 구해 r이라는 변수에 넣는다. 변수 r은 survey dimension에 따른 5개의 cell이 있고 그 안에 행은 참가자수(여기에선 42), 열은 component 경우의 수(현재 상태에선 14)와 같은 크기로 상관계수들이 들어있다. mean(r{i})은 같은 열에 있는 r 값을 평균을 낸 14개(component 경우의 수)의 값으로, 각 component 경우의 수일 때 참가자마다 구한 참가자수만큼의 평균 상관계수들의 평균값들이다. max(mean(r{i}))는 여러 component 경우의 수 중에서 상관계수의 평균이 가장 높은 것을 찾아준다. max\_r(i,1)은 그 최대 상관계수의 값을, max\_ncomp(i,1)은 그때의 component 경우를 알려준다.  
r{i}이 stats\_train{dim\_i}.pred\_outcome\_r과 다른 점은 r이 참가자 for loop을 통해 한 참가자의 4쌍의 데이터들 사이에서 상관계수를 각각 구한다는 것이다.반면 stats\_...는 한 참가자당 약 4개씩 나오는 예측 값들을 모두 한 데 놓고 뿌린 뒤 실제 값과의 상관계수를 계산한 것이다. (그렇다면 뭐가 더 신뢰로움?)  
a는 mean(r{i})와 동일한 값이 들어있는데, 두 줄의 계산을 통해서 [14x5] matrix 형태이다. mean(a,2)는 같은 행에 있는 요소들의 평균을 구한 것으로, 14개의 값이 나오게 된다. a를 plot 하면, 각 component 수의 경우에, 한 참가자에서 구한 상관계수들의 평균값들이 plot된다. 각 dimension마다 5개의 선 그래프가 생기며, dimension의 값들을 합쳐서 보려면 mean(a,2)를 plot하면 된다. 이때 나오는 선은 x = 2에서 peak이 되는데, 이것은 1부터 시작되는 plot이므로 실제로는 3개의 component로 train한 경우의 참가자마다 구한 상관계수들의 평균값이 다른 component들의 조합의 경우보다 크기가 크다는 것을 말한다.

Test on testing dataset에서는 dat\_test{dim\_i}.Y를 real survey data로 사용한다. 원래는 test 21명 x 4분위 = 84x1 double이 들어있어야 하는데 NaN 데이터가 삭제되어 그보다 적은 수가 들어갈 수 있다. 이때 predicted data는 stats\_train{dim\_i}{the proper number of component}.other\_output{1} (1x63303)과 dat\_test{dim\_i}.dat(63303x81)에 들어있는 body map 데이터를 곱하여 구한 값이 된다. 이 body map들은 4분위로 나뉘어진 trials의 body map의 평균이다. the proper number of component는 j = max\_ncom(i)로 한다면 각 dimension마다 r이 제일 높은 component 경우의 수가 되는 것이고, j=3로 한다면 위에서 plot(mean(a,2))를 했을 때 peak을 이뤘던 3 component 경우의 수를 모든 dimension에 적용할 수도 있다. 무엇을 할 지 선택할 것. Real과 predicted가 합쳐진 y\_yfit의 크기는 [81x2 double]이 된다. for 문을 test set에 해당하는 참가자들 21명으로 돌려서 예측값과 실제값 사이의 상관계수를 구해본다. rr은 한 참가자의 4분위 x 값 두 개(예상과 실제 = y\_yfit의 한 행) = 4행 2열 = 8개(NaN으로 빠져서 적을 수 있음) 사이의 상관계수가 된다. pp는 그 상관계수의 p 값이 저장된다. rr과 pp를 r\_test{i}의 두 열로 저장한다.   
r\_test는 5개의 cell로 이루어져 있고 그 안에는 행이 test set의 참가자 수 만큼인 [21x2] matrix가 들어있다. mean(r\_test{i}(:,1))하면 21명의 참가자의 상관계수 평균이 나온다. max가 거기에 다시 왜 붙었는지는 모르겠지만.. max\_r\_test은 5개의 dimension & 특정 component 경우의 수에서 참가자들의 상관계수의 평균값 [5x1] matrix이 된다. max\_ncomp\_test는 각 경우의 component가 몇번째 요소인가인데, 지금 코드에서는 각 dimension에 하나의 component 경우만 넣으므로 [1;1;1;1;1]이 나온다.  
best\_test\_sub는 test set의 참가자 중에서 상관계수가 가장 높은 참가자를 각 dimension마다 찾아낸 것이다. 이때 나온 번호는 test set에서의 번호이므로 유의할 것.

visualize the model – 각dimension마다 rating의 큰 절댓값을 예측하는 바디맵을 그려본다. 그리는 데 필요한 color를 load하고, stats\_max는 각 dimension마다 r이 가장 높았던 component 경우의 수 = max\_ncomp(i)에 해당하는 trained data를 stats\_train에서 추출하여 저장한다. weights는 이 stats\_max.weight\_obj.dat으로, stats\_train에 들어있던 weight을 저장한 것이다.   
(weight\_obj.dat = other\_output{1})body\_data는 바디맵 크기만큼의 NaN으로 채웠다가, 몸의 안 부분의 정보를 가진idx를 이용하여 몸의 안쪽 부분만 weight로 채운다.   
clim은 body\_data에 저장된 weight 데이터의 절댓값이 가장 큰 값을 찾아낸 것이다. imagesc로 그림을 그릴 때 clim과 -clim을 기준으로 그린다.   
colormap으로 색을 바꿀 수 있다.

correlation between predicted and real value for each subject of test set –   
위의 test on testing set은 train과 test 할 때 모두 4분위로 나누어 진행하였다. 4분위 정보로 train을 했고, test에서 상관계수를 구할 때에 참가자마다 4분위에 해당하는 trial들의 rating의 평균을 실제 값으로 놓고, 예측 값도 참가자마다 4분위에 해당하는 값(4분위에 해당하는 mean body map 한 개와 model을 dot product한 것)이 하나씩 들어있었다. 그러나 이 섹션에서는 model에 160개의 실제 body map을 곱하여 예측 값이 구해진다. 따라서 160개의 예측과 실제 값 사이의 상관계수를 구할 수 있게 된다. 이 상관계수는 word\_r로 저장되고 행은 참가자, 열은 dimension이 된다.

Wordcloud 그리기 – wordcloud 함수는 matlab 2018a 버전 이상부터 돌아간다. 이 과정은 위에서 구한 model과 실제 body map을 이용하여, 예측한 rating의 값(5가지)에 따라서 단어들의 모음을 보여주는 것이다. 예를 들어 한 단어의 body map을 가지고 예측한 rating 값이 크다면 그 단어가 중앙에 크게 나타나고, 값이 작다면 주변부에 작은 글자로 나타난다. 이때 어떻게 예측하는가가 달라질 수 있는데 model과 body map데이터를 dot product하거나, canlab\_pattern\_similarity 함수를 이용하여 cosine similarity 방법으로 하거나, 동일한 함수를 사용하여 correlation를 구하는 방법이다. 자세한 방법은 canlab\_pattern\_similarity을 살펴보면 도움이 된다. 이런 방법으로 예측한 rating의 값이 pred이고, 이곳에 음수가 있으면 오류가 나기 때문에 normalization을 거쳐야 하는데 pred에서 최솟값을 빼주고 범위(최댓값 – 최솟값)로 나눠주는 과정을 거친 것이 pred2이다.   
wordcloud(word, pred2)에서 word는 160개의 단어이며 160x1 크기의 string이며 pred2는 같은 크기의 예측값이다.



1. Markov chain analysis

Dir >> /Project/FAST/scripts/behavioral/fast\_180330\_hmm\_on\_surveydata.m

먼저 dat에 참가자수 x 160 trials x 5 dimensions가 되도록 get\_var를 사용.  
Q에서 중요하다고 여기는 questionnaire들의 문항을 다 가져옴.

dati\_val은 한 참가자, 한 dimension에 대해서 160개의 rating을 [40 x 4]크기의 matrix로 만든 것이다. idx는 dati\_val과 동일한 크기의 빈 matrix다.   
여기에 rating들을 어떤 기준으로 나눌 것인지를 정해야 하는데, 본 코드에 따르면 valence, time, safety-threat의 경우는 -.3과 .3을 기준으로 세 가지 그룹이 형성된다. 이 기준을 neg, pos로 저장하고, 이 기준에 따라서 idx를 1, 2, 3으로 저장한다. 작은 분위가1, 가장 큰 분위가 3. 이렇게 코딩했을 때 3 그룹 중에 하나도 없는 NaN이 존재할 수 있다.

trans\_mat{참가자}{run\_i}은 참가자 수만큼의 cell로 이루어져 있고 그 cell안에는 4개 run에 해당하는 4개의cell이 있다. 그 cell에는 확률 matrix가 들어갈 [3x3] matrix가 있다. trans\_idx는 idx의 4열 중 에서 한 run에 해당하는 한 column에 대하여, n번째 idx와 n+1번째 idx가 한 row가 되는 [39x2] matrix다. trans\_idx로 trans\_mat{참가자}{run\_i}가 정해지는데, n번째 idx =1 이었다가 n+1번째 idx = 2였다면 trans\_mat의 1행 2열의 숫자가 하나 늘어나게 된다. 이를 trans\_idx의 행 개수 39만큼 반복하게 되면, 행은 n번째, 열은 n+1번째 idx 쌍의 누적도수(개수)가 된다. 확률을 계산하기 위해서 trans\_idx 3x3의 총합을 각 항에서 나눠주게 되면 trans\_prob{참가자}{run\_i}이 된다.

이 trans\_prob를 각 참가자마다, 각 run마다 거듭제곱을 하게 되면 steady\_prob\_temp가 나오게 된다. 이때 결과값은 같은 열일 때 같은 값이 나오게 되어 1행만 잘라서 steady\_prob{참가자수}{run\_i}로 저장한다.   
attract\_prob는 특정 run에 대한 trans\_prob의 3개 값의 nanmean(NaN값을 제외하고 평균을 구한 것)이 된다. NaN이 들어있는 행렬의 mean을 구하면 NaN이라고 뜨기 때문이다.

이 과정을 run마다, 참가자마다 반복한다. steady\_prob는 참가자 수만큼의 cell이 존재하고 그 안에 [4x3] matrix가 존재한다. 행은 run, 열은 세 개의 idx 그룹을 의미한다. attract\_prob도 steady\_prob와 같이 참가자 수만큼의 cell이 있고 그 안에 [4x3] matrix가 들어있다.

steady\_prob\_sum는 참가자마다의 steady\_prob를 nanmean해서 넣은 것으로 참가자수만큼의 행이 있고, 열 3개는 세 개의 idx를 의미한다.

attract\_prob\_sum은 attract\_prob를 nanmean해서 넣은 것으로 참가자수만큼의 행이 있고, 열 3개는 세 개의 idx를 의미한다.

이런 과정을 5개의 dimension에서 반복하고, 이 확률들을 개인들의 questionnaire 결과와의 상관계수를 알아본다.

markov는 5개의 dimension에 대한 5개의 cell을 가진다.

markov의 여러 행들은 questionnaire 하나씩(또는 하위 항목) 들어간 것이고 열2개는 두 가지 확률에 해당한다. 첫번째 열은 steady\_prob\_sum과 설문 결과 사이의 nancorr을 하여 구한 상관계수 값이 들어가고, 두번째 열은 attract\_prob\_sum과 설문 결과 사이의 nancorr을 하여 구한 상관계수의 값이 들어가게 된다.

그 결과 중 하나로, PANAS\_NA와 markov의 steady 확률이 attract 확률보다 상관계수가 높은 것을 볼 수 있었다.

1. Physio figure 8/15

/Users/byeoletoile/CloudStation/Project/scripts/physio/fast\_180811\_physio\_ECG\_SCR\_figure\_Wani

본 분석에서는 한 trial당 60초(자극=단어 제시 전 10초~자극 제시 후 50초)의duration을 가진 physio 데이터를 사용하여 분석한다. IBI 데이터의 경우 TAPAS를 통해 preprocess한 'FAST\_dataset\_fmri\_ECG\_IBI\_TAPAS\_LPF05Hz\_DS25Hz.mat'를 불러와 분석한다.  
  
d\_t는 get\_var를 통해 가져온 survey rating 데이터이며 크기는 [subject # x 160 trials x dimension #]이다. d\_t(:,:,dim\_i)를 통해 특정 dimension 데이터를 잘라올 수 있다. 본 분석에서는 마지막 160번째 trial이 60초보다 작은 경우에 존재하기 때문에 d\_t(160,:)=[];을 통해 마지막 trial을 제외한다.   
  
bin\_idx는 d\_t와 같은 크기인 [159 trials x 93 subjects]와 같은 matrix이다. 여기에 for loop을 통해 survey rating을 두 가지 level로 구분하여 index를 저장한다. 예를 들어 valence, time, safety의 경우는 rating이 0.3보다 큰 경우에 up level = 2로 -0.3보다 작은 경우에 low level = 1로 코딩한다. 즉 bin\_idx는 0, 1, 2로만 이루어지게 된다.   
  
all\_ibi는 한 참가자의 특정 level의 trial들을 모은 뒤 그 trial들의 physio data의 평균을 저장하는 matrix이다. 이때 크기는 [(level수x subject #) x 한 trial의 physio data의 길이]가 된다. 이 경우에는 (2 level x 88명) x 1500 으로 176 x 1500 크기의 matrix가 된다.   
all\_y는 한 참가자의 특정 level의 trial들을 모은 뒤 그 trial들의 survey rating의 평균을 저장하는 matrix이다. 따라서 all\_ibi와 크기가 같다.   
sub\_idx는 all\_ibi와 all\_y의 행들의 subject가 누구인지 알려주는 matrix이다. 예를 들어 두 변수는 현재 176 행(2 level x 88명)으로 이루어져 있는데, 참가자의 idx를 보자면 [1;1;2;2;3;3…]와 같이 될 것이다. 크기는 [176x1]이 된다.

이제는 위의 변수들에 내용을 채워넣어야 한다. 먼저 baseline의 구간을 정하여 한 참가자의 한 level의 physio 데이터에서 baseline 구간의 평균값을 구하여 나머지 구간의 data에서 값을 빼준다. baseline\_idx는 baseline의 구간을 정해주는 것이다. 이제 참가자의 for loop 과 level loop이 돌게 된다.  
  
a 는 ibi를 60초에서 나눠준 bpm 데이터가 되는데, 한 참가자의 한 level에 속하는 trial들의 데이터들을 모아준다. 이때 크기는 [1500 x trial #]가 되는데 한 trial의 data가 다른 열로 차곡차곡 들어가게 되는 것이다. 이들을 모으고 60으로 나눠준 것이 a가 된다.   
그 밑의 try-catch loop을 통해서 baseline 구간(7~10초)의 평균을 빼준다. 예를 들어 한 참가자의 한 level에 40개의 trials이 있다면, mean(a(baseline\_idx,:))' 은 [40x1] 크기의 matrix로 40개 trial마다의 평균이 들어있다. repmat(mean(a(baseline\_idx,:))', 1, 1500)' 를 통해 40개의 평균 정보들은 [1500x40]의 크기로 a와 같은 크기로 확장되고, a에서 뺄 수 있게 된다. 즉 빼고 난 뒤의 a는 trial마다 baseline의 평균값을 계산하여 뺀 값들이 된다.   
a를 통해 위의 변수들에 정보를 넣어주는 데, all\_ibi의 한 행에는 a의 평균이 들어간다. 다시 말해, 한 참가자의 한 level에 속하는 trial들의 physio data의 평균이 한 행에 들어가게 된다. 그러니 2 levels x subject # 만큼의 행이 존재한다. all\_y는 d\_t를 통해서 평균을 구한 것으로 동일한 조건의 survey rating의 평균이 들어간다고 할 수 있다.   
  
lv와 kk는 2 cell이며 한 cell에 한 level의 정보가 들어간다. 즉 모든 참가자들의 low level의 all\_ibi데이터가 lv{1}에 들어간다. 따라서 lv{1}의 크기는 [subject # x physio data 길이]가 된다. kk는 all\_ibi의 데이터 상태를 파악하기 위한 logical로 한 참가자의 데이터가 NaN이거나, 한 참가자의 1500개의 데이터의 variance가 0.0001보다 작거나 2보다 큰 경우에는 logical 1이 되도록 한다. 밑의 bad\_idx는 logical인 kk를 subject index로 바꾼 것이다. bad\_idx를 이용하여, level 1, 2를 통틀어 문제가 있는 참가자는 level 1, 2의 데이터를 모두 지운다. 즉 level 1의 데이터만 문제가 있어도 level 2의 데이터까지 지운다. 이렇게 참가자를 제외하게 되면 lv{1}, {2}의 크기는 줄어들게 된다.   
  
이제 figure을 그리는 for loop이다.   
이제 lv{i}에는 모든 참가자들의 trial 중 한 level에 속하는 trial들의 physio data가 기본적인 준비(trial 차원에서 baseline 처리와 참가자 차원에서 평균한 뒤 NaN 데이터, 이상한 variance를 보이는 참가자 제외)가 된 뒤 들어가 있다. mean(lv{i}(:,baseline\_idx),2)은 [subject # x 1] 크기의 평균 데이터이다. 즉 각 참가자의 physio data에서 baseline 구간(7~10초) 데이터의 평균을 구한 것이다. 여기에 repmat을 이용하면 [subject # x physio data 길이]로 확장되고 lv{i}에서 뺄 수 있다. 즉 baseline 처리를 위해서 trial마다의 7~10초 구간의 평균도 빼주고, 각 참가자마다 두 개의 level로 평균으로 구해진 두 데이터도 각각 7~10초 구간의 평균을 빼주는 것이다. 그렇게 lv에서 빼준 뒤 nanmean을 하면, [1 x physio data 길이=1500] 크기의 matrix가 m이 된다. se는 마찬가지의 논리를 통해서 standard deviation을 구해준 값이 되며 m과 같은 크기가 된다. 이 m과 se를 wani\_plot\_shading에 입력하여 그래프를 그려준다. 이 함수의 첫번째 입력 변수인 1:825는 그래프의 x 축 구간을 의미하며 두 번째, 세 번째 입력 변수도 크기가 이와 같아야 한다. 이 크기와 맞게 m과 se의 구간을 잘라준다.   
  
ttest(lv{1}-lv{2})는 lv{1}과 lv{2}의 차이에 대한 1-표본 t 검정을 실시한다. 두 level의 데이터가 차이가 있는지 없는지를 보고자 하는 것이다. h는 1500개의 데이터에서 0보다 유의미하게 큰 데이터 배열은(위치) 1로, 유의미하지 않은 배열은 0으로 저장된다. p는 그에 해당하는 p값이다.   
scatter 함수에 find를 이용하여 유의미한 위치를 선으로 나타내도록 한다.

lv{i}의 데이터를 imgaussfilt에 입력하고, 두번째 변수로 0.7을 선택하여 가 0.7인 Guassian smoothing kernel로 필터링한다. 필터링 한 뒤 imagesc를 가지고 map을 그린다.

1. fast\_180424\_SPM\_firstlevel\_model1

Single trial model using the canonical HRF(duration is 15s),  
no control ventricle & white matter  
results = rating 4, view 160

|  |
| --- |
| %% read canlab\_dataset    load(fullfile(basedir, 'project/FAST/data/FAST\_dataset\_fmri.mat'));  % onsets  onsets\_view = get\_var(D, 'eventonsettime', 'conditional', {'eventname', 'view'});  onsets\_emotion\_select = get\_var(D, 'eventonsettime', 'conditional', {'eventname', 'emotion\_selection'});  onsets\_concentration = get\_var(D, 'eventonsettime', 'conditional', {'eventname', 'concentration'});    % names  names\_view = get\_var(D, 'eventname', 'conditional', {'eventname', 'view'});  names\_emotion\_select = get\_var(D, 'eventname', 'conditional', {'eventname', 'emotion\_selection'});  names\_concentration = get\_var(D, 'eventname', 'conditional', {'eventname', 'concentration'});    % durations  durations\_view = get\_var(D, 'eventduration', 'conditional', {'eventname', 'view'});  durations\_emotion\_select = get\_var(D, 'eventduration', 'conditional', {'eventname', 'emotion\_selection'});  durations\_concentration = get\_var(D, 'eventduration', 'conditional', {'eventname', 'concentration'}); |

onset, names, durations 변수들은 D에 입력된 데이터를 불러오는 것이며 trial의 종류에 view, emotion\_select, concentration으로 나뉜다. 같은 trial일 경우 onset, names, durations는 각각 크기가 같고, view의 경우에는 행 = 참가자 수, 열 = 160 trials, emotion\_select는 열이 20 trials, concentration의 경우 8 trials가 된다. 1 run에 emotion\_selection은 5번, concentration은 2번 응답하기 때문이다. names는 trial 종류의 이름을 담고 있고, onsets와 durations는 각각을 담고 있다.

|  |
| --- |
| subject\_id = D.Subj\_Level.id{sub\_j};  subj\_dir = fullfile(img\_dir, subject\_id);  subj\_outputdir = fullfile(modeldir, subject\_id);  cd(subj\_dir);    TR = 0.46;  hpfilterlen = 180;  images\_by\_run = filenames(fullfile(subj\_dir, 'func/sw\*task-fastfmri\*.nii'));  conditions\_per\_run = [41 41 41 41];    view\_run\_idx = repmat(1:4, 40,1); view\_run\_idx = view\_run\_idx(:); % 160 x 1  emotion\_select\_run\_idx = repmat(1:4, 5,1); emotion\_select\_run\_idx = emotion\_select\_run\_idx(:); % 20 x 1  concentration\_select\_run\_idx = repmat(1:4, 2,1); concentration\_select\_run\_idx = concentration\_select\_run\_idx(:); % 8 x 1 |

기본 설정들. conditions\_per\_run은 spm에 적용할 condition의 개수를 의미하는 것으로, 한 run에 41개의 condition이 있다는 것을 보여준다. images\_by\_run은 smoothing까지 끝난 preprocessed image이다.   
view\_run\_idx과 emotion\_select\_run\_idx, concentration\_select\_run\_idx은 view, emotion, concentration에 해당하는 run 번호를 나타내는 것이다. 예를 들어 view\_run\_idx는 160x1의 크기인데 순서대로 1이 40개, 2가 40개, 3이 40개, 4가 40개로 이루어져 있다. 마찬가지로 emotion\_select\_run\_idx 는 20x1, concentration\_select\_run\_idx은 8x1의 크기이다.

|  |
| --- |
| onsets = [];  durations = [];  names = [];    for run\_i = 1:4    onsets\_temp = num2cell(onsets\_view(sub\_j, view\_run\_idx==run\_i)');  onsets = [onsets;onsets\_temp];    duration\_temp = num2cell(durations\_view(sub\_j, view\_run\_idx==run\_i)');  durations = [durations;duration\_temp];    names\_temp = names\_view(sub\_j, view\_run\_idx==run\_i)';  for trial\_i = 1:numel(names\_temp)  names\_temp{trial\_i} = sprintf('%s\_%02d', names\_temp{trial\_i}, trial\_i);  end  names = [names;names\_temp];    onsets\_temp{1} = [onsets\_emotion\_select(sub\_j, emotion\_select\_run\_idx==run\_i)'; onsets\_concentration(sub\_j, concentration\_select\_run\_idx==run\_i)'];  onsets = [onsets;onsets\_temp];    duration\_temp{1} = [durations\_emotion\_select(sub\_j, emotion\_select\_run\_idx==run\_i)'; durations\_concentration(sub\_j, concentration\_select\_run\_idx==run\_i)'];  durations = [durations;duration\_temp];    names\_temp = names\_emotion\_select(sub\_j, emotion\_select\_run\_idx==run\_i)';  names = [names;{'rating'}];  end |

onsets, durations, names에 한 참가자의 정보만 넣을 것이다.   
for문에서 한 run씩 돌 때마다 onsets에는 41x1 cell이 늘어나 결국 164x1 cell이 된다. Run 1에 해당하는 41개의 cell 중에는 40개가 view에 해당하는 trial의 onset 정보가, 남은 1개의 cell에는 emotion\_selection과 concentration에 해당하는 trial의 onsets정보가 함께 들어있는 7x1 cell이 들어있다. duration도 마찬가지이며, names의 경우에는 ‘view\_01’ ~ ‘view\_40’ cell 40개와 ‘rating’ cell 1개가 있다.

|  |
| --- |
| %% prepare and save nuisance  load(fullfile(subj\_dir, 'PREPROC.mat'));  nuisance\_dir = fullfile(subj\_dir, 'nuisance\_mat');  if ~exist(nuisance\_dir, 'dir'), mkdir(nuisance\_dir); end    % without remove vw  if numel(filenames(fullfile(nuisance\_dir, 'nuisance\_run\*.mat')))==9  % skip  else  warning('No nuisance files. Please check')  input('');  for run\_i = 1:numel(PREPROC.nuisance.mvmt\_covariates)  R = [[PREPROC.nuisance.mvmt\_covariates{run\_i}, PREPROC.nuisance.mvmt\_covariates{run\_i}.^2, [zeros(1,6); diff(PREPROC.nuisance.mvmt\_covariates{run\_i})],  [zeros(1,6); diff(PREPROC.nuisance.mvmt\_covariates{run\_i})].^2],  PREPROC.nuisance.spike\_covariates{run\_i}];  R = [R zscore((1:size(R,1))')]; % linear drift  savename = fullfile(nuisance\_dir, sprintf('nuisance\_run%02d.mat', run\_i));  fprintf('\nsaving... %s', savename);  save(savename, 'R');  end  end    multi\_nuisance\_matfilenames{1} = fullfile(nuisance\_dir, sprintf('nuisance\_run%02d.mat', 3));  multi\_nuisance\_matfilenames{2} = fullfile(nuisance\_dir, sprintf('nuisance\_run%02d.mat', 5));  multi\_nuisance\_matfilenames{3} = fullfile(nuisance\_dir, sprintf('nuisance\_run%02d.mat', 7));  multi\_nuisance\_matfilenames{4} = fullfile(nuisance\_dir, sprintf('nuisance\_run%02d.mat', 9)); |

preprocessed에 들어있는 참가자 폴더에 들어가면 PREPROC 파일이 하나씩 있고, PREPROC.nuisance.mvmt\_covariates에는 nuisance 변수 중 movement와 관련된 상수들이 들어있다. run개수만큼 9개의 cell이 들어있고, 각 cell마다 그 run을 촬영한 TR 개수에서 disdaq으로 뺀 TR개수가 행으로 존재하는 double들이 들어있다.

|  |
| --- |
| 1 ~ 7번 열  [790×6 double] [510×6 double] [1970×6 double] [510×6 double] [1970×6 double] [510×6 double] [1970×6 double]  8 ~ 9번 열  [510×6 double] [1970×6 double] |

R은 각 run에 대한 PREPROC.nuisance를 사용하여, 새롭게 저장할 nuisance\_run.. 파일의 내용이 된다. R을 해체해보자.   
먼저 첫번째 열 6개에는 PREPROC.nuisance.mvmt\_covariates{run\_i}가 들어가고, 두번째 열 6개에는 이들의 제곱이 들어가고, 3번째 열 6개에는 첫번째 열 6개의 diff가 들어가고, 4번째 열 6개에는 3번째 열의 제곱이 들어간다. 여기까지가 24개의 열이며 이 이후의 열은 PREPROC.nuisance.spike\_covariates{run\_i}이 된다. 이 값은 run마다 다른데, 행은 TR개수이며, 열은 spike로 잡힌 TR의 개수로 run마다 다르다. 보통 TR 개수의 비례하여 증가한다. 이 값이 무엇인지 알기 위해서는 ‘humanfmri\_b3\_spike\_id.m’을 참고한다. 여기에 따르면 canlab tool의 preprocess라는 함수를 사용하여 outlier(spike)를 찾아낸다. 내가 실행한 코드에서는 mahalanobis distance를 기준으로 outlier time point를 찾고 root-mean-square successive difference를 통해서 한 번 더 outlier time point를 찾는다(더 자세한사항은 preprocess 함수 참조). 이때 그 시간에 대한 정보가 PREPROC.nuisance.spike\_covariates{run\_i}에 들어가게 된다. 여기에 1부터 R의 행수(TR개수)까지의 등차수열을 평균 0에 표준편차가 1인 정규분포로 바꾼 수열을 R의 마지막 열로 더해준다. 이것은 시간의 linear한 효과이며 linear drift라고도 한다. 이 각 열들은 하나의 regressor을 의미한다.  
  
여기에 model 7에서는 ventricle과 white matter를 통제해주는 nuisance variable을 추가하였다.

|  |
| --- |
| % ventricle & WM  [value, components] = extract\_gray\_white\_csf(fmri\_data(images\_by\_run{(run\_i-1)/2}));  R = [R components{2} components{3}];  R = double(R); |

이 R은 run 수만큼, 9개가 나오게 되며 nuisance\_run#.mat으로 저장된다.   
model에서는 task run(3, 5, 7, 9)의 R 4개를 multi\_nuisance\_matfilenames의 cell 안에 넣어둔다.

이 이후의 부분은 canlab\_spm\_fmri\_model\_job으로 matlabbatch를 만들고, 이를 ‘spm\_model\_spec\_estimate\_job.mat’을 저장하고 spm\_jobman으로 실행한다.   
그 다음은 design matrix를 확인할 수 있도록 out을 만들고 vifs.mat으로 저장한다.  
여기까지 오면 beta~.nii 파일이 매우 많이 만들어지는데, 이 중 일부를 삭제하고 필요한 파일만 새 이름으로 link를 만든다. SPM.xX.name에 ‘R’이 들어가거나 ‘constant’가 들어갈 경우의 beta파일을 삭제하고 view 160, rating 4개에 해당하는 beta이미지는 새로운 링크 파일을 만든다.   
  
model 2의 경우는 model 1과 설정은 동일한데, HRF 대신에 fir 설정으로 모델을 만든 것이다.

fast\_180430\_basic\_contrast는 model1의 second level analysis이다. view와 rating에 해당하는 fmri data를 불러와서 두 그룹 간의 group-level t-test를 해볼 수 있다. orthviews를 하면 3D에 활성화가 그려진다. brain\_activations\_wani는 3차원 brain을 그려준다.

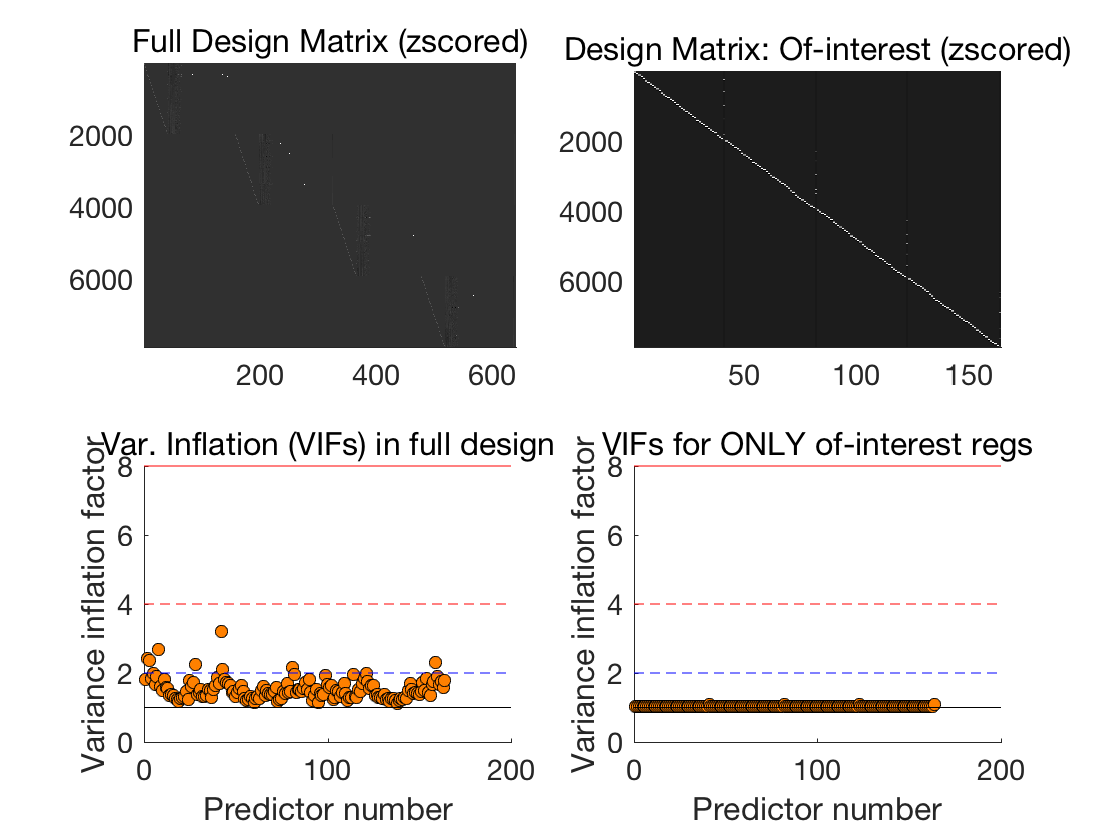
nuisance file을 보는 법.   
>> load(‘nuisance\_run01.mat’)

>> imagesc(R)

이때 뜨는 그림은 nuisance 효과를 design matrix로 만든 것이며 추후에 regression에 사용하게 된다.

SPM으로 first level을 돌리고 난 뒤 기본적인 확인법.  
참가자 폴더마다 하나씩 있는 SPM.mat을 load.  
imagesc(SPM.xX.X) 는 design matrix를,   
imagesc(SPM.xX.nKX) 는 filtering 한 후의 design matrix를 보여준다.   
이것은 전체 run에 대한 design matrix를 모두 포함하는 그림으로, 아래 그림의 Full Design Matrix와 같은 그림이 나오게 된다.

이 이미지는 out = scn\_spm\_design\_check(subj\_outputdir,’events\_only’);을 통해서 생성되며, VIFs for ONLY of-interest regions가 낮은 것이 중요하다.  
높다는 것은 design matrix의 서로 다른 열끼리 상관이 높다는 것을 의미하고, 이것은 nuisance variable에 문제가 있을 수 있다.



1. Second level analysis

**second\_level/model1\_survey\_single\_tr\_predict  
code >> fastprojectdir/scripts/imaging/fast\_180430\_survey\_single\_tr\_predict\_v1.m**

model 1을 가지고 survey data를 예측해본다. 현재 코드에서는 cv\_lassopcr을 사용하였고, model에 씌울 mask를 ‘EMOActivation\_FWE\_all.img’와 ‘mni\_brain\_mask.nii’ 두 가지를 사용하였다.

또한 각 run을 fold를 잡아 학습하였다. 그리고 참가자마다 실제 rating과 model에 의한 예측값사이의 correlation은 stats.pred\_outcome\_r에 저장된다. 여기서 다시 ‘EMO\_highvif\_5’라는 변수에 각 참가자마다, 각 dimension마다의 상관계수가 저장된다. 이 변수는 ‘EMO\_highvif\_mat’으로 저장되었고, 결과를 확인해보면 r 값의 참가자 간 평균이 모든 dimension에서 0을 넘지 못한다.

mni mask를 사용한 결과도 마찬가지이다.   
[개선방법] run으로 fold를 잡지 않고 10 fold 정도 해볼 것. 그러나 160 trials에 10 folds가 적당한 것인지는 확인이 필요하다.

**second\_level/model1\_basic\_contrast  
code >> fastprojectdir/scripts/imaging/fast\_180430\_basic\_contrasts.m**

model 1에서 만든 한 참가자의 여러 trial들의 view beta(160개)와 rating beta(4개)들을 평균하여 한 사람당 하나의 view 와 하나의 rating 이미지를 만든다.

이후 모든 참가자들의 view와 rating 평균 이미지에 대해 각각 t-test를 진행한다. ‘tdat’에 그 정보가 저장된다. 그리고 ‘map\_savedir’에 t-value image를 nii 파일로 저장한다.

파일이름은 ‘group\_view\_1st\_visit\_bfr.nii’와 ‘group\_rating\_1st\_visit\_bfr.nii’이다.

이 파일은 orthviews로 볼 수 있는데, view trial에서 참가자들간의 일관된 brain activity가 붉은 색으로 표현된다.

fast\_180430\_model1\_survey\_predict\_v1.m은 EMOActivation\_FWE\_all.img 를 마스크로 사용하여, 한 사람마다 brain으로 rating을 예측하는 within-subject pcr을 돌린 것. **leave-one-run-out**도 했으나 잘 나오지 않았고, 10 folds도 잘 나오지 않음.

fast\_180524\_model1\_prediction\_v1.m은 mni mask를 이용하여 참가자 내에서 한 dimension rating을 기준으로 5(4)가지 level로 trial들을 나눈 뒤, 같은 level에 해당하는 trial들의 평균 값과 평균 brain 데이터를 모든 참가자에 대해서 stack한다. train과 test set을 나눈 뒤에 먼저 training data에서 **leave-one-subject-out**으로 prediction을 했을 때 결과가 잘 나오지 않았다. 그래서 buckner mask도 시도했음.

fast\_180917\_model7\_prediction\_v1.m은 참가자 내에서 run을 4개의 fold로 만들고 하나의 fold만 제외 후 학습한 뒤 제외한 fold를 예측해보면(=**leave-one-run-out**) 잘 나오지 않고, 8, 10 folds를 한 경우에만 0.3~0.4 정도 나온다.

fast\_180917\_model7\_prediction\_loso.m은 5가지 dimension에 더하여 절댓값 dimension까지 더하여 확인해보았다. res에는 각 참가자의 10 folds **within-subject prediction**의 pred\_outcome\_r과 weight\_obj(weight map)가 들어있다. 이때의 결과는 not bad.

그런 뒤, 이것을 가지고 leave one subject out(**LOSO**)으로 prediction을 하였는데, res를 복사한 model\_dat에서 한 참가자를 지운 뒤 나머지 참가자들의 weight map을 평균한다. (Woo 2017 Nat.Com와 같은 방법으로, 모든 참가자들의 데이터를 다 dat에 넣고 leave-one-subject-out하는 것과는 조금 다르다. 이 방법은 나머지를 mean하는 것. 위의 방법은 noisy할 수 있고 Wani의 경험 상 잘 된 적이 없다고 한다.)  
 그런 뒤 지운 참가자의 dat.dat과 이 평균 값을 weight처럼 dot product를 하여 survey 예측값 pexp를 구한다. 그런 뒤 실제 rating 값인 dat.Y와 pexp 사이의 상관 값을 res.valence\_loso\_por에 저장한다. 이 값은 참가자마다 하나가 나온다. 이 값들을 참가자 간에 평균을 내었을 때 결과를 보면 0.1을 넘는 것이 없다.

fast\_180917\_model7\_prediction\_loso\_neurosynth\_mask.m은 먼저 참가자 내에서, 참가자마다 neurosynth mask를 이용하여 10 folds **within-subject prediction**을 하였다. 이 pred\_outcome\_r을 차원마다 평균을 내었을 때 0.3보다 큰 r값이 나온다. 다음 단계로 위 코드와 동일한 방식으로 **leave-one-subject-out**을 해보았을 때의 r은 거의 0으로 잘 나오지 않는다.

fast\_180917\_model7\_prediction\_loso\_level.m 은 survey results를 4개의 level로 나눈다. 나눈 뒤 trial의 level index에 따라 평균을 낸다. 한 사람의 survey rating를 level에 따라 mean 하고, brain data도 mean을 하여 여러 명의 데이터를 차곡차곡 쌓는다(stack). whfolds에 참가자 번호를 index로 남겨놓고 **leave-one-subject-out**으로 training and prediction한다. 한 참가자를 뺐을 때 각각 구한 상관값으로 총 참가자 명수만큼 값이 나오게 되는데 이들을 평균했을 때에는 vividness가 0.1로 나오는 것이 최대로, 안 나온다. 또 within-subject correlation으로, 한 참가자 내에서의 correlation만 구하여 이들을 평균한 값은 time이 0.3, vivid가 0.1로 나오고 다른 차원에서는 나오지 않는다. 여기서 radius가 5인 searchlight을 적용하여 ‘sl\_res’에 그 결과를 저장한다. = 'fast\_180918\_quartile\_predict\_model7\_combined.mat'

fast\_180920\_model7\_prediction\_searchlight\_masking\_level.m은 위의 'fast\_180918\_quartile\_predict\_model7\_combined.mat' 을 불러와 searchlight 기반으로 잘 나온 영역들을 골라 mask를 만들게 된다. 두 가지 방법으로 mask를 만들었는데, searchlight에서 prediction r이 높은 상위 (1-prc)%의 voxel들 중에서, k개 이상 모여있는 voxel cluster들을 먼저 고른다. 그런 뒤 이들을 더 크게 만들기 위해서 r만큼의 반지름으로 영역을 확대하여 mask를 만든다. 그런데 이 경우에는 원래 searchlight으로 찾은 상위 voxels들 중에서 r이 더 높은 voxel과 낮은 voxel을 구분하지 않고 모두 mask에 포함되게 된다. r이 더 높은 voxel들을 더 중요하게 걸러내기 위해 voxel cluster에 대해서 r 크기의 smoothing을 한다. 그런 뒤에 그 값이 상위 95% 이상인 voxel들만 mask에 포함시키는 것이다. 두 경우 모두 mask에 포함된 영역은 1, 아닌 영역은 0으로 binary 데이터가 되게 된다. 이렇게 mask를 저장한 뒤에 LOSO로 참가자 간 예측을 한다. 이 예측한 값과 실제 값을 참가자마다 구하여 평균을 낸 within-subject prediction의 결과를 보았을 때 세 가지 model이 살아남게 된다.

fast\_180927\_model7\_prediction\_searchlight\_masking\_level\_testset.m은 ‘~ model7\_prediction\_loso\_level.m’와 같이 test set에도 적용시켜본 코드이다. test set에 대해 진행하고 그것을 'fast\_180927\_quartile\_predict\_model7\_testdata.mat'에 저장한다. 그런 뒤 test set의 참가자들의 brain data를 mask와 dot product하여rating 예측 값을 구한다. 이를 참가자마다 실제 값과의 correlation을 구하면, 잘 안 나온다. 이때의 예측 값이 level별로 나눠서 한 것이었으므로, trial-by-trial로 within-subject prediction을 다시 한번 시도해본다. 결과가 나오지 않음.

fast\_180927\_model7\_prediction\_numcomponent\_level.m 은 fast\_180917\_model7\_prediction\_loso\_level.m에서 만든 'fast\_180918\_quartile\_predict\_model7\_data.mat'을 이용하여 분석한다. 이 데이터는 gray matter mask로 masking했고, 여러 참가자의 quartile level마다 평균한 이미지 dat과 참가자 index인 whfolds가 들어있다. 즉 한 참가자마다 4개의 평균 이미지가 dat에 들어있다. 이를 가지고 **leave-one-subject-out** pcr로 training를 하는데, 이때 component 수를 조절하는 옵션 ['numcomponents', num\_comp(comp\_n)]을 더하여 각 경우마다 for 문을 돌린다. 여기서 component의 수는 5, 10, 20, 30, 40, 50, 60, 70, 90, 120, 150개로 정하였다. 그리고 그 통계값 stats\_train을 ‘fast\_180927\_quartile\_predict\_model7\_numcom.mat’에 저장한다.

먼저 leave-one-subject-out 의 pre\_outcome\_r을 보면 전체적으로 모든 component의 경우의 수에서 결과가 좋지 않으며, vividness에서만 가장 좋은 경우에 0.2가 넘는다.  
within-subject prediction의 결과를 보면 time에서만 .3이 나오며 가장 많은 component 개수인 150개를 모두 사용한 경우에 이런 결과가 나왔다. 즉, component 수를 조절해서 잘 되지는 않았다.

fast\_180928\_model7\_prediction\_weighted\_loso.m 은 fast\_180917\_model7\_prediction\_loso.m에서처럼 gray matter mask를 사용하고 level 없이, 10 folds로 within-subject training 한 데이터를 이용한다. 'fast\_180917\_indiv\_pred\_all\_dims.mat'를 불러오면 된다. 불러온 뒤 LOSO를 시도하는 과정에서 추가적인 과정을 시행한다. 먼저 ‘model\_dat’에 res.valence.dat을 불러온 뒤, 해당 한 참가자 데이터를 leave out하고, 나머지 참가자들의 데이터를 standard deviation-based normalization을 한다. 즉, std로 전체를 나누어준 값이 ‘model\_dat’가 된다. 그런 뒤, precision-weighted mean 처리를 하는데, 해당dimension에 해당하는 pred\_outcome\_r 값을 ‘prec’(41x1)으로 불러와서 해당 참가자 데이터를 지우고, 나머지 데이터 중에서도 r값이 0이하인 값은 0으로 만든다. 그런 뒤 이 변수 prec을 prec의 총합으로 나눈 ‘w’를 만든다. 그런 다음, ‘model\_dat’과 ‘w’를 곱한 값을 ‘model\_dat’로 다시 저장한 뒤에, 참가자의 brain data와 dot product를 한다. 이것이 예측값이며 이를 실제와 상관하여 ‘res.valence\_loso\_por\_wei’라고 저장한다. 이들을 평균한 결과, 잘 나오지 않는다.

fast\_180928\_model7\_parcel\_based\_prediction\_v1.m 은 참가자마다 구한 parcellation에 대한 상관계수를 구한 뒤 general한 mask를 만들고 그 mask를 이용하여 다시 training을 하는 방법이다. 먼저 참가자마다 278개의 parcellation마다 상관계수를 하나씩 구한다. 그런 뒤에 parcellation에 해당하는 상관계수 값들을 참가자 간의 평균을 낸다. 그랬을 때 t-test를 하여 t-value가 상위 30프로 이내인 region들을 골라 dimension마다 하나의 mask를 만든다. 이 mask를 적용한 brain data로 다시 cv-pcr training을 참가자마다 진행한다. 참가자 수만큼 저장된 pred\_outcome\_r을 평균내면 참가자 내에서의 예측값과 실제의 상관계수를 알 수 있다. 이것은 앞의 결과와 일관되게 0.3 이상이 나온다. 하지만 이 mask를 적용한 brain data들을 가지고 leave-one-subject-out을 시행하여 평균 상관계수를 보면 유의미하지 않다.

1. 데이터를 MNI space 상에서 보거나, 수정하거나, 저장하는 과정에 대한 코드

/project/FAST/scripts/imaging/fast\_180920\_model7\_prediction\_searchlight\_masking\_level.m

/project/FAST/scripts/imaging/fast\_180928\_model7\_parcel\_based\_prediction\_v1.m

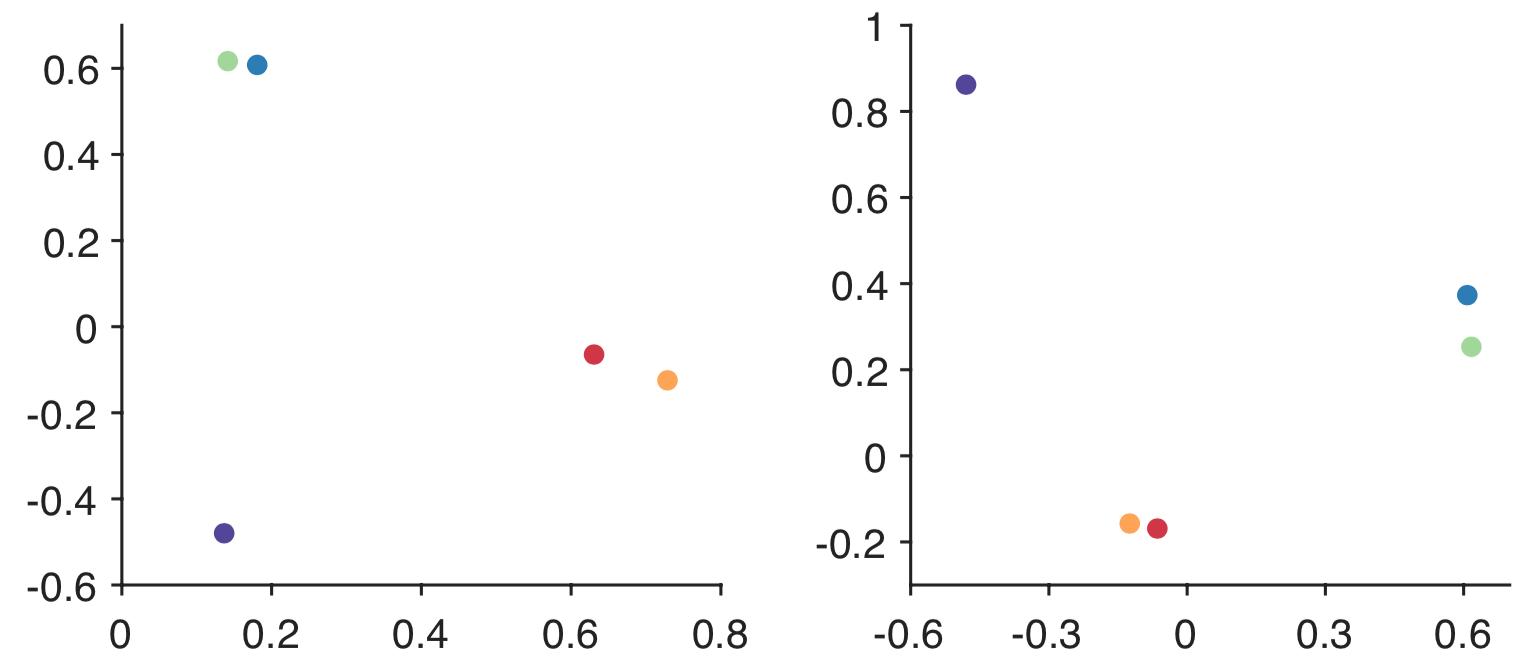
/project/FAST/scripts/imaging/fast\_181006\_model7\_view\_and\_PCA\_activation.m

/project/FAST/scripts/imaging/fast\_181015\_model10\_displayresults.m  
model 10에서의 이미지를 평균한 것, t value가 특정 값 이상, p value 가 특정 값 이상인 영역을 골라서 저장한다.   
Output = ‘model10\_ttest\_mean.nii’, ‘model10\_ttest\_tval.nii’, ‘model10\_ttest\_pval.nii’

/project/FAST/scripts/imaging/fast\_181017\_model10\_region\_select\_fir\_display.m  
위 코드의 output을 불러와 region을 나누고, 일정 크기 이상의 영역을 골라내고, 눈으로 보고 확인하고, 원하는 영역만 다시 골라내어 저장하는 것까지 나와있다.

Neurosynth에서 nii를 다운받아 열면 다른 mask이더라도 dat의 크기는 동일하기 때문에 여러 mask를 합치고 싶은 경우에는 or을 사용하고, fullpath만 변경하여 저장하면 된다. (더 자세한 것은 기억이 안나니 교수님…)

1. 핵심 Figure를 만든 코드

① fast\_181024\_model7\_PCA\_encoding\_model.m  
behavioral ratings (61x160x5)을 (9760x5) matrix로 만든 뒤, pca 함수에 넣어서 pca\_coeff를 뽑는다. pca\_coeff는 열마다 principal component에 해당하는 coefficient가 들어있고, 첫번째 열이 가장 주요한 component가 된다. pca\_coeff는 그 중에서도 주요한 component 3개를 잡는다.  
  
이번에는 model 7의 이미지를 불러온 뒤에, dat.X에 위에서 구한 coefficient와 실제 dimension rating의 값을 곱해준다. 예를 들어 dim\_score \* pca\_coeff를 하면 (160x5)\*(5x3)이 되어서 결과값은 (160x3)이 된다. 이것을 dat.X에 넣고 dat을 regress함수에 넣어서 regression을 진행한다. temp.b는 regress 함수의 결과 중에서도 beta에 해당된다. 이 크기는 (211339x4)가 되는데, 추정컨대 3개의 pc에 해당하는 beta값과 intercept 값이 추가되어 4개의 열이 된다. 이 beta값을 pc의 순서에 따라 다시 정렬하고 여러 참가자에게서 얻은 데이터를 쌓아 놓은 것이 dat\_reg{reg\_i}이다. dat\_reg는 4개의 cell로 이루어져 있고, dat\_reg의 첫번째 cell이 첫번째 pc에 해당하는 beta값이다. 61명의 참가자 데이터를 모았다면 나중에 dat\_reg{1}.dat은 (211339x61)의 값이 들어가게 된다.  
  
이후에 여러 가지 figure들이 그려지는데, SfN 등에서 이용했던 basic contrast map은 ‘view\_average\_model7\_bfr05.nii’이다. 또한 PC에 따른 brain map은 ‘view\_dim1\_PC1\_valence\_model7\_unc001\_01\_05pruned.nii’이다. 이는 beta값이었던 dat\_reg을 t-test로 거른 다음 pruning을 통해 map을 만든 것이다.  
  
‘PC\_dimensions\_n61’은 오른쪽과 같은 그림이다.   
이는 위에서 pca\_coeff에 들어있는 5x3 데이터에서 3개 중 2개의 열을 골라서 scatter plot으로 나타낸 것이다. 즉 이 좌표들은 5가지 dimension rating을 pca의 3가지 축으로 나타낸 coefficient이다.   
  
similarity with large-scale networks는 위에서 전체 참가자의 contrast 이미지와 buckner가 나눈 large-scale network와의 correlation을 구해서 plot으로 나타낸 것이다.   
  
  
다음에 찾을 코드, decoding 시도했던 코드찾기.

②③④⑤⑥

1. HPC 사용

필요한 code, function들이 들어있는 폴더들을 매트랩을 열 때마다 addpath 하거나 rmpath 해야한다. 따라서 코드 안에 넣어둔다. Imaging 파일은 addpath에 넣어두지 않아도 된다. 아래의 코드로 Accumbens에서 업데이트한 github 파일을 터미널을 통해 HPC의 저장공간인 sas1으로 바로 copy할 수 있다.  
>> rsync -avuP /Volumes/accumbens/github/\* cocoanlab@115.145.185.181:/sas1/cocoanlab/Resources/github/

HPC에서 Matlab을 이용할 때 두 가지 방법이 있다. 하나는 XQuartz프로그램과 matlab& 명령어를 이용해 GUI 창을 열어서 사용하는 방법과 터미널을 통해 background로 matlab을 실행하는 경우가 있다. GUI를 이용할 때 오래 걸리는 code를 시행시켰을 때 GUI 전체가 오랜 시간 동안 정지되는 에러가 발생하여 전체 실행을 완료하는데 시간이 예상보다 훨씬 초과할 때가 있다. 따라서 background에서 돌리는 것의 장점이 있는데 이렇게 돌릴 때에는 실행시킬 코드가 function의 형태로 완성되어 있어야 하고, 중간에 에러가 났을 때 처리하기가 어려우므로, try-catch를 써서 에러가 났을 때 현재 변수들을 다 저장하고 닫히도록 한다. 그리고 command 창에서 나오는 출력값들을 txt 파일로 저장하는 명령어를 추가하도록 하자.   
  
예시 코드  
>> matlab\_orig -nodesktop -nodisplay -nosplash -r "cd /sas1/cocoanlab/FAST/scripts; addpath(genpath(pwd)); tictoctest(1); exit" > ~/tictoctest\_no1.txt 2>&1 < /dev/null &

2>&1 2 = standard error, 1 = standard output. Error를 log에 적으라는 의미가 된다.   
dev/null/ Alert and start Matlab

& background 작업을 의미

jobs를 보면 작업 상태를 볼 수 있음

cat tictoctest\_na1.txt. txt 읽기

중단하고 싶을 때는 jobs에서 숫자를 확인한 다음 e.g) kill %3   
또는 kill PIDnumber를 쳐도 꺼진다. Nohub실행을 이렇게 끌 수 있음

kill -9 -1  이러면 내가 킨 명령들이 다 꺼진다.

fg %3  하면 그 jobs로 들어갈 수 있다. 어떻게 나오는지는 모름.  
  
이것을 여러 참가자에 대해 반복해야 할 때는 매트랩으로 command에 쓸 내용을 클립보드에 복사한 뒤에 특정 노드에 들어가 있는 터미널에 복붙을 하면 된다.   
clipboard('copy', byeol\_command) 은 클립보드에 한 줄로 이루어진 Byeol\_command를 복사해준다.