E3_compute_precision_sensitivity

October 31, 2024

[1]:
```python
#Python version 3.11.7
#Jupyter Notebook version 7.0.8
import numpy as np # version 1.26.4
from scipy import stats # version 1.11.4
import pandas as pd # version 2.1.4
import matplotlib.pyplot as plt # version 3.8.0

plt.rcParams["font.family"] = "Arial"
plt.rcParams["font.size"] = 12
```

The following levels are considered: - scramble boundaries: 1 bar (2 seconds) - scramble boundaries: 2 bars (4 seconds) - every 3 bars (6 seconds) - half-phrase: 4 bars (8 seconds) - every 5 bars (10 seconds) - phrase: 8 bars (16 seconds) - half-section: 16 bars (32 seconds)

Final alignment plot (figure 4): each panel in the final plot will show all of the above levels. Each condition is a different color. Musicians and non-musicians will be in separate subplots.

1 Load the data and ground truth

Load the timestamps.

[2]:
```python
timestamps = pd.read_csv('../data/timestamps_filtered_long.csv')
print(timestamps)
```

```
      exp_subject_id Musician  stimulus_set scramble  stim_num   value
0             377777       No             1       2B         1   6.924
1             377777       No             1       2B         1  27.737
2             377777       No             1       2B         1  37.522
3             377777       No             1       2B         1  45.708
4             377777       No             1       2B         1  61.284
…                  …       …             …        …         …       …
3547          611455       No             3       2B        12  52.989
3548          611455       No             3       2B        12  63.226
3549          611455       No             3   Intact         1  32.335
3550          611455       No             3   Intact         1  58.304
3551          611455       No             3   Intact         2  32.814

[3552 rows x 6 columns]
```

[4]:
```python
# load ground truths
gts = pd.read_csv('../data/combined/ground_truths.csv')
# remove last column (NaNs - IDK why it's there)
gts = gts.drop("Unnamed: 5", axis=1)
print(gts)
```

```
      stimulus_set scramble  stim_num  level  boundary_time
0                1   Intact         1     16             34
1                1   Intact         1     16             66
2                1   Intact         1      8             18
3                1   Intact         1      8             34
4                1   Intact         1      8             50
…              …      …          …    …              …
3171             4       1B        16      1             52
3172             4       1B        16      1             54
3173             4       1B        16      1             56
3174             4       1B        16      1             58
3175             4       1B        16      1             60

[3176 rows x 5 columns]
```

2 Functions

[5]:
```python
def compute_ps_chance(data, gt, window_before=0.25, window_after=1.0,␣
↪samples=1000):
    '''
    Computes precision, sensitivity, and alignment (F) for single subject,␣
↪single condition - used within `ps_wrapper`

    Default window before is 0.25 seconds, default window after is 1.0 seconds.␣
↪Number of samples used to make null distribution is 1000.
    '''
    levels = pd.unique(gt['level'])
    trials = pd.unique(data['stim_num'])
    output = np.zeros([3, len(levels)]) # first dim is precision, sensitivity,␣
↪F; second dim is each level

    for level in range(len(levels)):
        # what are the ground truth boundary times for this level?
        these_gt_vals_both = gt[gt['level'] == levels[level]]

        # set up list to hold both trials
        precision = []
        sensitivity = []
        avg_chance_precision = []
        avg_chance_sensitivity = []
```

```
    for tr in trials:
        # grab the responses for this trial
        these_responses = data[data['stim_num'] == tr]['value'].to_numpy()
        total_responses = np.shape(these_responses)[0]
        these_gt_vals = these_gt_vals_both[these_gt_vals_both['stim_num']␣
↪== tr]['boundary_time'].to_numpy()

        # compute the number of "in window responses"
        # for each GT boundary, is there a response in the window around␣
↪that?
        in_window_response_by_bound = np.zeros(these_gt_vals.shape[0])
        for w in range(len(these_gt_vals)):
            # define the "in-window" range
            range_before = these_gt_vals[w] - window_before
            range_after = these_gt_vals[w] + window_after

            # for each response, check if the response is in the range
            for r in these_responses:
                if r > range_before and r <= range_after:
                    # if it is, set the corresponding in-window count to 1
                    in_window_response_by_bound[w] = 1 # this prevents␣
↪double-counting
                # otherwise do nothing

        in_window_responses = np.sum(in_window_response_by_bound)

        # compute precision and sensitivity
        precision.append(in_window_responses / total_responses)
        sensitivity.append(in_window_responses / np.shape(these_gt_vals)[0])

        # compute chance using a bootstrap approach
        # lists to hold results from many samples
        chance_precision = []
        chance_sensitivity = []

        for sample in range(samples):
            # generate random responses
            responses_random = np.random.rand(total_responses) * 68 # to␣
↪account for length of trial

            # compute the number of "in window responses"
            # for each GT boundary, is there a response in the window␣
↪around that?
            in_window_response_by_bound = np.zeros(these_gt_vals.shape[0])
            for w in range(len(these_gt_vals)):
                # define the "in-window" range
                range_before = these_gt_vals[w] - window_before
```

```
                    range_after = these_gt_vals[w] + window_after

                    # for each response, check if the response is in the range
                    for r in responses_random:
                        if r > range_before and r <= range_after:
                            # if it is, set the corresponding in-window count␣
↪to 1
                            in_window_response_by_bound[w] = 1 # this prevents␣
↪double-counting
                        # otherwise do nothing

                in_window_responses = np.sum(in_window_response_by_bound)

                chance_precision.append(in_window_responses / total_responses)
                chance_sensitivity.append(in_window_responses / np.
↪shape(these_gt_vals)[0])

            avg_chance_precision.append(np.mean(chance_precision))
            avg_chance_sensitivity.append(np.mean(chance_sensitivity))

        # take the mean and adjust for chance
        precision_mean_adj = np.mean(precision) - np.mean(avg_chance_precision)
        sensitivity_mean_adj = np.mean(sensitivity) - np.
↪mean(avg_chance_sensitivity)

        # average precision and sensitivity across trials and save in the␣
↪output array
        output[0,level] = precision_mean_adj
        output[1,level] = sensitivity_mean_adj
        # compute and save F
        if precision_mean_adj == 0.0 and sensitivity_mean_adj == 0.0:␣
↪output[2,level] = 0.0
        else: output[2,level] = (2 * precision_mean_adj * sensitivity_mean_adj)␣
↪/ (precision_mean_adj + sensitivity_mean_adj)

    return output
```

[6]:
```
def ps_wrapper(data, gt, group, stimulus_set, window_before=0.25,␣
↪window_after=1.0, samples=1000):

    # all the data gets passed, so first have to filter by group and stimulus␣
↪set
    this_data = data[data['Musician'] == group]
    this_data = this_data[this_data['stimulus_set'] == stimulus_set]

    # pull out subject ids
```

```
    sub_ids = pd.unique(this_data['exp_subject_id'])
    # the conditions array should be defined earlier in the notebook, but copy␣
↪it here for sanity
    conditions = ['Intact', '8B', '2B', '1B']
    # pull out the levels (compute_ps also does this)
    levels = pd.unique(gt['level'])

    # initialize the output array
    # 3 (P,S,F) x number of subjects x number of conditions x number of levels
    output = np.zeros([3, np.shape(sub_ids)[0], len(conditions), len(levels)])

    # each subject individually
    for s in range(sub_ids.shape[0]):
        this_sub_data = this_data[this_data['exp_subject_id'] == sub_ids[s]]

        # further, filter by condition
        for c in range(len(conditions)):
            this_cond_data = this_sub_data[this_sub_data['scramble'] ==␣
↪conditions[c]]
            if this_cond_data.empty:
                #print("Subject %s is missing data." %sub_ids[s])
                continue
            this_gt = gt[gt['scramble'] == conditions[c]]

            output[:,s,c,:] = compute_ps_chance(this_cond_data, this_gt,
                                                window_before=window_before,␣
↪window_after=window_after, samples=samples)


    return output
```

3 Compute precision, sensitivity, and overall alignment

ps_wrapper takes one group (musician/non-musician) and one stimulus set at a time.

[7]:
```
psf_M_1 = ps_wrapper(timestamps, gts, group='Yes', stimulus_set=1)
psf_M_3 = ps_wrapper(timestamps, gts, group='Yes', stimulus_set=3)
psf_M_4 = ps_wrapper(timestamps, gts, group='Yes', stimulus_set=4)
psf_NM_1 = ps_wrapper(timestamps, gts, group='No', stimulus_set=1)
psf_NM_3 = ps_wrapper(timestamps, gts, group='No', stimulus_set=3)
psf_NM_4 = ps_wrapper(timestamps, gts, group='No', stimulus_set=4)
# this cell takes a bit
```

Combine all stimulus sets.

[9]:
```
psf_M_all = np.concatenate((psf_M_1, psf_M_3, psf_M_4), axis = 1)
psf_NM_all = np.concatenate((psf_NM_1, psf_NM_3, psf_NM_4), axis = 1)
```

[10]:
```python
print(np.shape(psf_M_all))
print(np.shape(psf_NM_all))
```

```
(3, 45, 4, 7)
(3, 45, 4, 7)
```

Data structure is P/S/F x number of subjects x condition x levels.

3.1 Save alignment values

Wrangle F values into a long form with labels so we can read it in R. If anyone has any suggestions for how to do this more efficiently, please let me know :)

[14]:
```python
levels = ['16', '8', '5', '4', '3', '2', '1']
```

[16]:
```python
f = psf_M_all[2,:,:,:]
```

Separate each condition and save as a separate dataframe

[17]:
```python
f_I = pd.DataFrame(f[:,0,:], columns = levels)
f_I.insert(0, 'scramble', 'Intact')
f_8B = pd.DataFrame(f[:,1,:], columns = levels)
f_8B.insert(0, 'scramble', '8B')
f_2B = pd.DataFrame(f[:,2,:], columns = levels)
f_2B.insert(0, 'scramble', '2B')
f_1B = pd.DataFrame(f[:,3,:], columns = levels)
f_1B.insert(0, 'scramble', '1B')
```

[18]:
```python
# concatenate
f_M = pd.concat([f_I, f_8B, f_2B, f_1B])
# reset index so we have a subject column
f_M = f_M.reset_index()
f_M = f_M.rename(columns = {"index": "sub"})
# add a group column
f_M.insert(0, 'Musician', 'Yes')
```

[19]:
```python
print(f_M)
```

```
    Musician  sub scramble        16         8         5         4         3  \
0        Yes    0   Intact  0.092180  0.029338 -0.052741 -0.013109  0.008378
1        Yes    1   Intact  0.076953 -0.065617 -0.039501 -0.129176 -0.071079
2        Yes    2   Intact -0.056000 -0.062857 -0.063950 -0.069115 -0.068206
3        Yes    3   Intact  0.447246  0.222488 -0.014428  0.085393  0.024988
4        Yes    4   Intact -0.045513 -0.049706 -0.049871 -0.051598 -0.051099
..       …   …      …        …        …        …        …        …
175      Yes   40       1B  0.000000  0.000000  0.000000  0.000000  0.000000
176      Yes   41       1B -0.062358 -0.085917  0.160500 -0.091668 -0.015594
177      Yes   42       1B  0.198519  0.208151 -0.016299  0.233151 -0.086834
178      Yes   43       1B  0.074500 -0.011125  0.033000 -0.012333  0.063667
179      Yes   44       1B  0.114200  0.310857  0.302111  0.117364  0.048929
```

```
            2         1
0    0.007587  0.007182
1   -0.111014 -0.079989
2   -0.070340 -0.051410
3    0.009150  0.019770
4   -0.014525 -0.024580
..        ...       ...
175  0.000000  0.000000
176  0.063928  0.042361
177  0.049953 -0.008668
178 -0.069650  0.024676
179  0.191000  0.083694

[180 rows x 10 columns]
```

Repeat for non-musicians

[20]:
```python
f = psf_NM_all[2,:,:,:]

f_I = pd.DataFrame(f[:,0,:], columns = levels)
f_I.insert(0, 'scramble', 'Intact')
f_8B = pd.DataFrame(f[:,1,:], columns = levels)
f_8B.insert(0, 'scramble', '8B')
f_2B = pd.DataFrame(f[:,2,:], columns = levels)
f_2B.insert(0, 'scramble', '2B')
f_1B = pd.DataFrame(f[:,3,:], columns = levels)
f_1B.insert(0, 'scramble', '1B')

# concatenate
f_NM = pd.concat([f_I, f_8B, f_2B, f_1B])
# reset index so we have a subject column
f_NM = f_NM.reset_index()
f_NM = f_NM.rename(columns = {"index": "sub"})
# add a group column
f_NM.insert(0, 'Musician', 'No')
```

[21]:
```python
print(f_NM)
```

```
    Musician  sub scramble        16         8         5         4         3  \
0         No    0   Intact -0.031429 -0.030462 -0.036842 -0.034240 -0.033636
1         No    1   Intact -0.046273 -0.050119 -0.050346 -0.051160 -0.053852
2         No    2   Intact -0.094570  0.058013 -0.057804  0.016128  0.072560
3         No    3   Intact  0.184605  0.067875 -0.079070 -0.004863 -0.085352
4         No    4   Intact -0.065915 -0.142856  0.018058 -0.067925  0.108733
..       ...  ...      ...       ...       ...       ...       ...       ...
175       No   40       1B  0.073097  0.120352 -0.148001  0.108612  0.040119
176       No   41       1B -0.029418  0.053685  0.027600 -0.007265  0.250436
177       No   42       1B -0.030400 -0.030000  0.119538 -0.032000  0.053304
```

```
178       No   43       1B -0.052000 -0.056800 -0.059714 -0.062020  0.021518
179       No   44       1B  0.075264 -0.165842  0.025425 -0.112314 -0.034047

            2         1
0   -0.034776 -0.035253
1   -0.054053 -0.033572
2    0.047997  0.008747
3   -0.045718 -0.061631
4    0.070121  0.114615
..        …         …
175  0.124664  0.043521
176  0.189848  0.118477
177 -0.036364  0.023015
178  0.055352 -0.008931
179 -0.066093 -0.076170

[180 rows x 10 columns]
```

Concatenate across both groups and save

[23]:
```
f_all = pd.concat([f_M, f_NM])
f_all.to_csv('alignment.csv', index = False)
```

Only issue is that both musicians and non-musicans are both labelled 0-44. This is addressed in `E3_alignment.Rmd`

4 Plot alignment values

[24]:
```
conditions = ['Intact', '8B', '2B', '1B']
cond_colors = ['red', 'orange',  'green', 'blue']
cond_jitter = [-.225, -.075, .075, .225]
levels = np.asarray([1,2,3,4,5,8,16])
levels = np.flip(levels)
```

[25]:
```
fig, ax = plt.subplots(1, 2, sharey = True, figsize = (18,6))
#plt.tight_layout()

for c in range(len(conditions)):
    ax[0].plot(levels + cond_jitter[c], np.mean(psf_M_all[2,:,c,:], axis=0),␣
 ↪color = cond_colors[c], alpha = 1, label = conditions[c])
    ax[0].scatter(levels + cond_jitter[c], np.mean(psf_M_all[2,:,c,:], axis=0),␣
 ↪color = cond_colors[c], alpha = 1)
    ax[0].errorbar(levels + cond_jitter[c], np.mean(psf_M_all[2,:,c,:],␣
 ↪axis=0), yerr = stats.sem(psf_M_all[2,:,c,:], axis=0),
                       color = cond_colors[c], capsize = 3, alpha = 0.4)
```

```
    ax[1].plot(levels + cond_jitter[c], np.nanmean(psf_NM_all[2,:,c,:],␣
↪axis=0), color = cond_colors[c], alpha = 1,
                label = conditions[c])
    ax[1].scatter(levels + cond_jitter[c], np.nanmean(psf_NM_all[2,:,c,:],␣
↪axis=0), color = cond_colors[c], alpha = 1)
    ax[1].errorbar(levels + cond_jitter[c], np.nanmean(psf_NM_all[2,:,c,:],␣
↪axis=0),
                   yerr = stats.sem(psf_NM_all[2,:,c,:], axis=0, nan_policy =␣
↪'omit'),
                   color = cond_colors[c], capsize = 3, alpha = 0.4)

ax[0].set_ylabel('Overall Alignment', fontsize = 22)
ax[0].set_title('Musicians', fontsize = 20)
ax[1].set_title('Non-musicians', fontsize = 20)

for col in range(2):
    ax[col].set_xlim(0, 17)
    ax[col].hlines(0,17,0, color = 'black', alpha = 0.2)
    ax[col].set_xticks(levels)
    ax[col].set_xticklabels(levels, fontsize = 16)
    ax[col].tick_params(axis='y', which='major', labelsize=14)
    ax[col].set_xlabel('Level (Bars)', fontsize = 18)
    ax[col].legend(fontsize=16)

#plt.savefig('alignment.png', dpi=500)
```