E3_compute_precision_sensitivity

October 2, 2025

[54]:
```
#Python version 3.11.8
#Jupyter Notebook version 7.0.8
import numpy as np # version 1.26.4
from scipy import stats # version 1.14.1
import pandas as pd # version 2.2.3
import matplotlib.pyplot as plt # version 3.10.0

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

import random
random.seed(10)
```

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.

[7]:
```
timestamps = pd.read_csv('../data/E3/timestamps_filtered_long.csv')
print(timestamps)
```

```
      exp_subject_id Musician  stimulus_set scramble  stim_num   value
0             342236       No             3       1B        14  13.650
1             342236       No             3       1B        14  19.409
2             342236       No             3       1B        14  30.038
3             342236       No             3       1B        14  37.846
4             342236       No             3       1B        14  47.451
…                …        …             …        …         …       …
3696          393230      Yes             1   Intact         3  35.817
3697          393230      Yes             1       2B         1  25.218
3698          393230      Yes             1       2B         1  31.133
3699          393230      Yes             1       2B         1  41.952
3700          393230      Yes             1       2B         1  57.157
```

```
[3701 rows x 6 columns]
```

[8]:

```python
# load ground truths
gts = pd.read_csv('../data/stimulus_info_E3/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

[10]:

```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
```

[11]:
```
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)

    print('done with group: %s, stimulus set: %d' %(group, stimulus_set))

    return output
```

3 Compute precision, sensitivity, and overall alignment

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

[14]:
```
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
```

```
done with group: Yes, stimulus set: 1
done with group: Yes, stimulus set: 3
done with group: Yes, stimulus set: 4
done with group: No, stimulus set: 1
done with group: No, stimulus set: 3
done with group: No, stimulus set: 4
```

Combine all stimulus sets.

[16]:
```
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)
```

[17]:
```
print(np.shape(psf_M_all))
print(np.shape(psf_NM_all))
```

```
(3, 49, 4, 7)
(3, 46, 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.

[21]:
```
levels = ['16', '8', '5', '4', '3', '2', '1']
```

[22]:
```
f = psf_M_all[2,:,:,:]
```

Separate each condition and save as a separate dataframe

[24]:
```
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')
```

[25]:
```
# 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')
```

[26]:
```
print(f_M)
```

```
    Musician  sub scramble        16         8         5         4         3  \
0        Yes    0   Intact  0.097392  0.024424 -0.048692 -0.016205  0.011249
```

```
1         Yes    1   Intact -0.223805 -0.072769 -0.033259 -0.126604 -0.074780
2         Yes    2   Intact -0.054750 -0.067071 -0.061400 -0.069538 -0.069059
3         Yes    3   Intact  0.446562  0.225265 -0.016145  0.086177  0.022813
4         Yes    4   Intact -0.049277 -0.049094 -0.053803 -0.049845 -0.053905
..        …   …       …        …         …         …         …
191       Yes   44       1B  0.000000  0.000000  0.000000  0.000000  0.000000
192       Yes   45       1B -0.061877 -0.074087  0.164583 -0.089344 -0.023060
193       Yes   46       1B  0.186635  0.210594 -0.024251  0.230003 -0.085436
194       Yes   47       1B  0.116600  0.314286  0.302667  0.126727  0.046786
195       Yes   48       1B  0.072500 -0.013875  0.044200 -0.011750  0.060467

            2         1
0    0.006503  0.006241
1   -0.112704 -0.082417
2   -0.069860 -0.052150
3    0.009655  0.017970
4   -0.013774 -0.024379
..        …         …
191  0.000000  0.000000
192  0.061877  0.042618
193  0.050299 -0.007389
194  0.192316  0.084000
195 -0.065450  0.022595

[196 rows x 10 columns]
```

Repeat for non-musicians

[28]:
```
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')
```

[29]:
```
print(f_NM)
```

```
    Musician  sub scramble        16         8         5         4         3  \
0         No    0   Intact -0.032000 -0.036615 -0.035263 -0.035200 -0.034727
1         No    1   Intact -0.045228 -0.051503 -0.050793 -0.052283 -0.053124
2         No    2   Intact -0.093635  0.052007 -0.053470  0.022265  0.070561
3         No    3   Intact  0.187252  0.070191 -0.078889 -0.007331 -0.087380
4         No    4   Intact -0.068624 -0.138911  0.017329 -0.069850  0.112449
..       …   …      …       …        …       …        …       …
179       No   41       1B  0.045131  0.199869  0.044812  0.237609  0.194410
180       No   42       1B -0.050000 -0.058400 -0.064286 -0.066772  0.019281
181       No   43       1B -0.060508 -0.073355  0.200744  0.020808  0.080379
182       No   44       1B  0.069969 -0.035077  0.024607 -0.112559 -0.030084
183       No   45       1B -0.083680  0.035532  0.207179  0.046981 -0.015543

            2         1
0   -0.034776 -0.037192
1   -0.053219 -0.033204
2    0.046217  0.006833
3   -0.043343 -0.058272
4    0.070832  0.114896
..        …       …
179  0.359801  0.392748
180  0.052384 -0.008317
181  0.015350  0.013711
182 -0.065983 -0.076799
183  0.050623  0.017134

[184 rows x 10 columns]
```

Concatenate across both groups and save

[31]:
```python
f_all = pd.concat([f_M, f_NM])
f_all.to_csv('../data/E3/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

[34]:
```python
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)
```

[58]:
```python
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('../figures/Fig4_alignment.png', dpi=500)
```