

Ввод [1]:

```
1 from pathlib import Path
2 from matplotlib import pylab as plt
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import json
7
8 %matplotlib inline
9 plt.style.use('bmh')
10 plt.rcParams.update({'font.size': 14})
11
```

Загрузка

Описание формата csv файла:

<http://openvibe.inria.fr/csv-file-format-description/> (<http://openvibe.inria.fr/csv-file-format-description/>)

Коды событий:

<http://openvibe.inria.fr/stimulation-codes/> (<http://openvibe.inria.fr/stimulation-codes/>)

Ввод [2]:

```
1 openvibe_config = json.load(open("./config/openVibe.json", "r"))
```

Ввод [3]:

```
1 raw_data = pd.read_csv("./Записи/OpenVibe signals/motor-imagery-csp-1-a
```

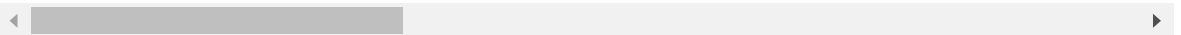
Ввод [4]:

```
1 raw_data.head(6)
```

Out[4]:

	Time:250Hz	Epoch	Channel 1	Channel 2	Channel 3	Channel 4	Cha
0	0.000	0	171386.796875	165426.734375	154487.453125	162283.515625	181144.1
1	0.004	0	171392.593750	165444.875000	154490.781250	162296.968750	181143.5
2	0.008	0	171387.046875	165428.828125	154489.281250	162286.640625	181143.6
3	0.012	0	171377.921875	165400.750000	154485.890625	162266.593750	181143.3
4	0.016	0	171376.703125	165396.484375	154484.875000	162261.906250	181142.1
5	0.020	0	171385.546875	165420.531250	154489.937500	162277.234375	181143.1

6 rows × 21 columns



Фильтруем события, для нас интересны стрелки (лево/право)

Ввод [5]:

```
1 def filter_events(df: pd.DataFrame, config: dict[str, any]) -> pd.DataF
2     """Получение табличной нотации для экспериментов из файлов OpenVibe
3
4     Args:
5         df (pd.DataFrame): сырой файл .ov сконвертированный в .csv и за
6         config (dict[str, any]): словарь с расшифровкой кодов событий 0
7
8     Returns:
9         pd.DataFrame: фрейм (таблица) в нашей собственной нотации
10    """
11
12    events = []
13    __temp_event = {}
14
15    new_frame = df.copy()
16
17    __ov_events = df[~df['Event Id'].isna()][['Event Id']]
18    for idx in range(__ov_events.shape[0]):
19        __row_events = __ov_events.iloc[idx, 0].split(":")
20
21        if config['main_actions']['left'] in __row_events or config['ma
22            __temp_event = {"start": __ov_events.index[idx]}
23            if config['main_actions']['left'] in __row_events:
24                __temp_event['action'] = 'left'
25            if config['main_actions']['right'] in __row_events:
26                __temp_event['action'] = 'right'
27
28        # В записи stop повторяется дважды (как и любые другие действия
29        if config['main_actions']['stop'] in __row_events and len(__tem
30            __temp_event['stop'] = __ov_events.index[idx + 1]
31            events.append(__temp_event.copy())
32            __temp_event = {}
33
34    new_frame = new_frame.drop(columns=['Epoch', 'Event Id', 'Event Dat
35    new_frame['action'] = None
36
37    for event in events:
38        new_frame.loc[event['start']: event['stop'], 'action'] = event[
39
40    new_frame['action'] = new_frame['action'].fillna('relax')
41    return new_frame
```

Ввод [6]:

```
1 def get_ranges(labels: np.ndarray) -> list[list[int, int, int]]:
2     """Преобразование сырых лейблов временного ряда в отрезки для визуа.
3
4     Args:
5         labels (np.ndarray): лейблы
6
7     Returns:
8         list[list[int, int, int]]: список отрезков для визуализации, ка
9         тройка "начало", "конец", "тип действия"
10    """
11
12    actions = []
13    __temp_action = [0]
14    __prev_action = labels[0]
15    idx = 1
16
17    while True:
18        if idx + 1 > len(labels):
19            __temp_action.extend([idx, __prev_action])
20            actions.append(__temp_action)
21            break
22
23        if labels[idx] != __prev_action:
24            __temp_action.extend([idx - 1, __prev_action])
25            actions.append(__temp_action)
26            __temp_action = [idx]
27            __prev_action = labels[idx]
28
29        idx += 1
30
31    return actions
```

Ввод [7]:

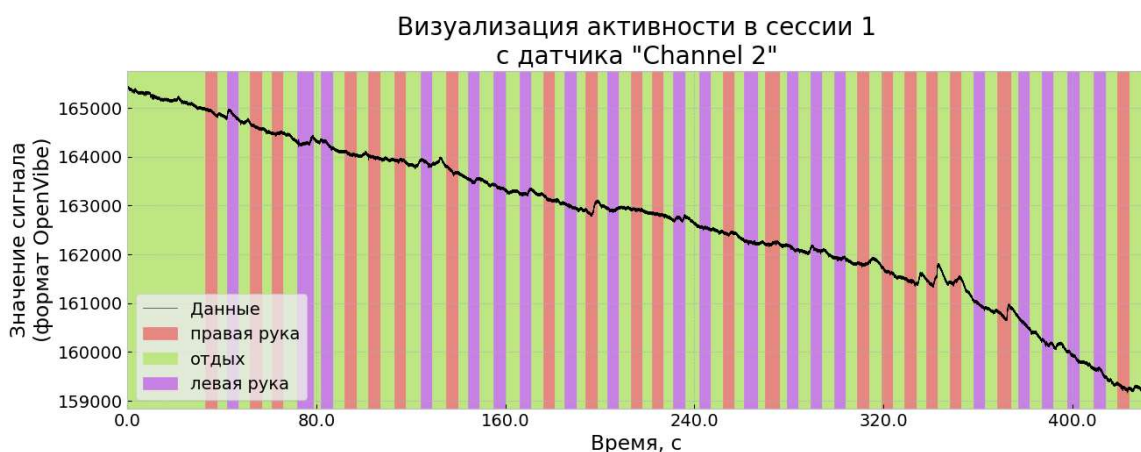
```
1 data = filter_events(raw_data, openvibe_config)
```

Ввод [8]:

```

1 plot_data = data.copy()
2 actions = get_ranges(plot_data['action'])
3 action_colors = {"relax": "#C0E986", "left": "#CA86E9", "right": "#E98B"}
4 action_names = {"relax": "отдых", "left": "левая рука", "right": "права"}
5 index = 2
6
7 fig, ax = plt.subplots(figsize=(15, 5))
8 __main = ax.plot(plot_data.iloc[:, index], color="black", linewidth=.4,
9 ax.set_xticks(ax.get_xticks())
10 ax.set_xticklabels([x * 0.004 for x in ax.get_xticks()])
11 ax.set_xlim(0, 107_776)
12
13 __patches = []
14 for act in actions:
15     __patches.append(plt.axvspan(act[0], act[1], color=action_colors[ac
16
17 ax.set_title(f"Визуализация активности в сессии 1\nc датчика \"{data.co
18 ax.set_xlabel("Время, с")
19 ax.set_ylabel("Значение сигнала\n(формат OpenVibe)")
20 ax.legend(loc="best", handles=[__main[0]] + __patches[7:10])
21 plt.show()
22
23 del __main, __patches

```

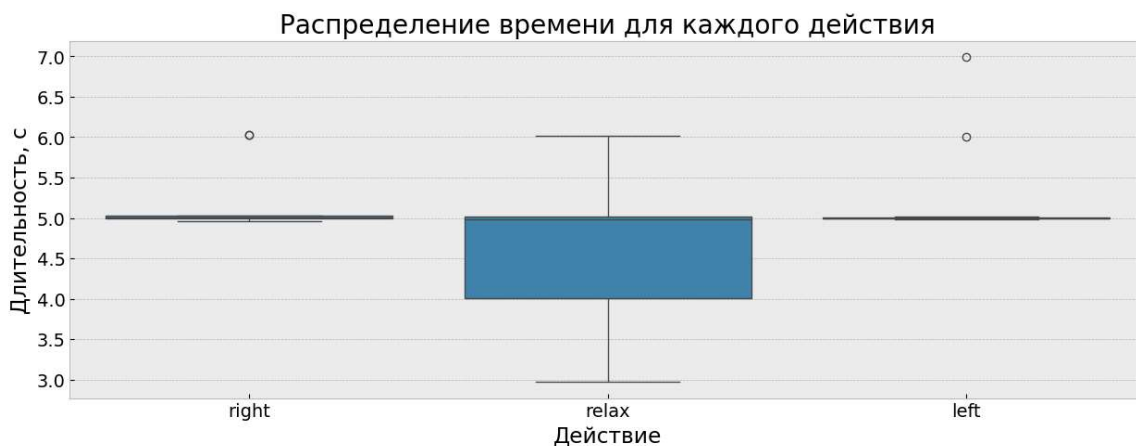


Ввод [9]:

```

1 fig, ax = plt.subplots(1, figsize=(15, 5))
2 __temp = pd.DataFrame({'Длительность, с': [(x[1] - x[0]) / 250 for x in
3 sns.boxplot(data=__temp, x='Действие', y='Длительность, с', ax=ax)
4 ax.set_title('Распределение времени для каждого действия')
5 del __temp
6 plt.show()

```



Что делаем с выбросами? Это так и задуманно? (вопрос 1 из сообщения).

▼ Формирование бинарного датасета 👤

Параметры датасета

- Полосовой фильтр ✓
 - частота 8 — 30Гц
 - порядок 5
- Эпохи ✓
 - 0,5с после предъявления стимула
 - 4с - длительность эпохи
 - Насколько смещаемся?
- CSP-фильтр по эпохам ✓
 - количество признаков 1024

▼ Полосовой фильтр

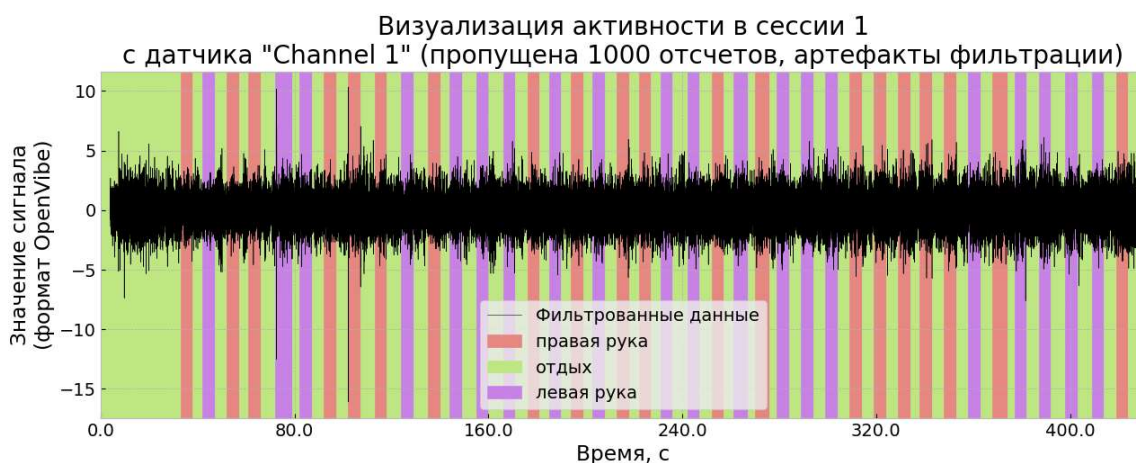
Ввод [10]: 1 `from scipy.signal import butter, lfilter`

Ввод [11]:

```
1 def __create_constants(low: float, high: float, sample_rate: float, ord
2     """Вычисление констант полосового фильтра (Баттерворта)
3
4     Args:
5         low (float): нижняя частота среза
6         high (float): верхняя частота среза
7         sample_rate (float): частота дискретизации
8         order (int): порядок фильтра
9
10    Returns:
11        tuple[float, float]: константы фильтра, необходимые для вычисле
12    """
13    return butter(order, [low, high], fs=sample_rate, btype='band')
14
15 def filter_signal(df: pd.DataFrame, cols: list[str], constants: tuple[f
16     """Фильтрация датасета (сигналов с датчиков)
17
18     Args:
19         df (pd.DataFrame): датафрейм с данными из OpenVibe
20         cols (list[str]): список колонок, данные в которых нужно отфиль
21         constants (tuple[float, float]): константы для вычисления фильт
22
23     Returns:
24         pd.DataFrame: входной фрейм с данными, с примененным фильтром
25     """
26     new_df = df.copy()
27     for col in cols:
28         new_df[col] = lfilter(*constants, df[col].values)
29
30     return new_df
31
```

Ввод [12]:

```
1 index = 1
2
3 constants = __create_constants(8.0, 30.0, 250.0, 5)
4 plot_data = filter_signal(plot_data, [data.columns[index]], constants)
5 actions = get_ranges(plot_data['action'])
6 action_colors = {"relax": "#C0E986", "left": "#CA86E9", "right": "#E98B"}
7 action_names = {"relax": "отдых", "left": "левая рука", "right": "права"}
8
9 fig, ax = plt.subplots(figsize=(15, 5))
10 __filtered = ax.plot(plot_data.iloc[1000:, index], color="black", linewidth=2)
11 ax.set_xticks(ax.get_xticks())
12 ax.set_xticklabels([x * 0.004 for x in ax.get_xticks()])
13 ax.set_xlim(0, 107_776)
14
15 __patches = []
16 for act in actions:
17     __patches.append(plt.axvspan(act[0], act[1], color=action_colors[act]))
18
19 ax.set_title(f"Визуализация активности в сессии 1\nc датчика \"{data.columns[index]}\"")
20 ax.set_xlabel("Время, с")
21 ax.set_ylabel("Значение сигнала\n(формат OpenVibe)")
22 ax.legend(loc="best", handles=__filtered[0] + __patches[7:10])
23 plt.show()
24
25 del __patches, __filtered
```



Ввод [13]:

```
1 constants = __create_constants(low=8.0, high=30.0, sample_rate=250.0, o
2 filtered_data = filter_signal(df=data, cols=[x for x in data.columns if
```

▼ Скользящее окно

Пояснение к вопросу 3 из сообщения

Разбиение:

|-----| Стимул (5с)

|-----| Эпоха 1 (0, 5с паузы + 4с данных)

|-----| Эпоха 2 (0, 5с паузы + 0, 5с смещение окна + 4с данных)

Ввод [14]:

```
1 def split_epochs_binary(  
2     df: pd.DataFrame,  
3     epoch_duration: int,  
4     init_pause: int,  
5     final_pause: int,  
6     step: int,  
7     map_labels: dict[str, int],  
8     cols: list[str]  
9 ) -> tuple[np.ndarray, np.ndarray]:  
10     """Формирование бинарного датасета (из отрезков, когда есть действи  
11  
12     Args:  
13         df (pd.DataFrame): фрейм с данными  
14         epoch_duration (int): длительность эпохи (в отсчетах сигнала, t  
15         init_pause (int): начальная пауза - сколько отсчетов выбросить  
16         final_pause (int): конечная пауза - сколько отсчетов выкинуть в  
17         step (int): шаг смещения внутри времени прдъявления стимула (в  
18         map_labels (dict[str, int]): словарь соотносящий действие и его  
19         cols (list[str]): имена колонок, которые будут добавлены в дата  
20  
21     Returns:  
22         tuple[np.ndarray, np.ndarray]: кортеж с 2 массивами:  
23  
24         * обучающие данные, размерность [количество образцов X длит  
25  
26         * метки классов (в соответствии с map_labels), размерность [  
27  
28     Raise:  
29         ValueError: словарь map_labels содержит информацию не о 2х клас  
30     """  
31  
32     if len(map_labels) != 2:  
33         raise ValueError(f"Словарь map_labels должен содержать 2 класса  
34  
35     actions = get_ranges(df['action'])  
36     x = []  
37     y = []  
38  
39     for action in actions:  
40         if action[2] in map_labels.keys():  
41             __low_bound = action[0] + init_pause  
42             __upper_bound = action[1] - final_pause - epoch_duration +  
43             for inner_idx in range(__low_bound, __upper_bound, step):  
44                 x.append(df.loc[inner_idx: inner_idx + epoch_duration -  
45                 y.append(map_labels[action[2]])  
46     return np.array(x), np.array(y)
```

Ввод [15]:

```
1 x, y = split_epochs_binary(  
2     df=filtered_data,  
3     epoch_duration=4 * 250,  
4     init_pause=int(0.5 * 250),  
5     final_pause=0,  
6     step=int(0.5 * 250),  
7     map_labels={"left": 0, "right": 1},  
8     cols=[x for x in data.columns if "Channel" in x]  
9 )
```


Ввод [16]:

```
1 x.shape, y.shape
```

Out[16]: ((71, 1000, 16), (71,))



CSP фильтр

Ввод [17]:

```
1 import scipy
```

Ввод [18]:

```
1 class CSPFilter:
2     """
3     Реализация CSP фильтра в стиле sklearn.
4     Основано на реализации github.co/Hiroaki-K4/total_perspective_vorte
5     """
6
7     def __init__(self, n_components: int) -> None:
8         self.n_components = n_components
9         self.classes = None
10        self.filters = None
11        self.patterns = None
12        self.mean = None
13        self.std = None
14
15    def __get_covariate(self, x: np.ndarray, y: np.ndarray, cls: any) -
16        """Получение ковариаций
17
18        Args:
19            x (np.ndarray): отсчеты сигнала
20            y (np.ndarray): метки класса
21            cls (any): текущий класс
22
23        Returns:
24            np.ndarray: матрица ковариаций
25        """
26
27        x_class = x[y == cls]
28        _, _, n_channels = x_class.shape
29        # x_class = np.transpose(x_class, [1, 0, 2])
30        x_class = x_class.reshape(n_channels, -1)
31        cov = np.dot(x_class, x_class.T)
32        return cov
33
34    def fit(self, x: np.ndarray, y: np.ndarray) -> None:
35        """Обучение фильтра
36
37        Args:
38            x (np.ndarray): отсчеты сигнала (числа)
39            y (np.ndarray): метки класса (любые данные)
40        """
41        self.classes = np.unique(y)
42        n_classes = len(self.classes)
43        if n_classes != 2:
44            raise ValueError(f"Количество классов для CSP должно быть =
45
46        cov_neg = self.__get_covariate(x, y, self.classes[0])
47        cov_pos = self.__get_covariate(x, y, self.classes[1])
48
49        eig_vals, eig_vecs = scipy.linalg.eigh(cov_neg, cov_pos)
50        for i in range(len(eig_vecs)):
51            eig_vecs[i] = eig_vecs[i] / np.linalg.norm(eig_vecs[i])
52
53        sorted_vals = np.argsort(eig_vals)
54        idxs = np.empty_like(sorted_vals)
55        idxs[0::2] = sorted_vals[len(sorted_vals) // 2 :][::-1]
56        idxs[1::2] = sorted_vals[: len(sorted_vals) // 2]
57
58        eig_vecs = eig_vecs[:, idxs]
59        self.filters = eig_vecs.T
60        self.patterns = np.linalg.inv(eig_vecs)
61        pick_filters = self.filters[:, self.n_components]
```

```

62
63     x = np.asarray([np.dot(pick_filters, epoch.T) for epoch in x])
64     x = (x ** 2).mean(axis=1)
65     self.mean = x.mean(axis=0)
66     self.std = x.std(axis=0)
67
68     def transform(self, x: np.ndarray) -> np.ndarray:
69         """Применение фильтра
70
71         Args:
72             x (np.ndarray): отсчеты сигнала (числа)
73
74         Returns:
75             np.ndarray: вторичные параметры (фичи)
76         """
77         pick_filters = self.filters[: self.n_components]
78         x = np.asarray([np.dot(pick_filters, epoch.T) for epoch in x])
79         x = (x ** 2).mean(axis=1)
80         x -= self.mean
81         x /= self.std
82         return x
83
84     def fit_transform(self, x: np.ndarray, y: np.ndarray) -> np.ndarray:
85         """Обучение с последующим применением ко входным данным
86
87         Args:
88             x (np.ndarray): отсчеты сигнала (числа)
89             y (np.ndarray): метки класса (любые данные)
90
91         Returns:
92             np.ndarray: вторичные параметры (фичи)
93         """
94         self.fit(x, y)
95         return self.transform(x)

```

Ввод [19]:

```

1 csp_filter = CSPFilter(4)
2 csp_filter.fit(x, y)
3 csp_filter.transform(x).shape

```

Out[19]: (71, 1000)

Ввод []:

```
1
```