

In [1]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import statsmodels.api as sm
5 import seaborn as sns
6 from tqdm import tqdm_notebook
7 import warnings
8 from statsmodels.tsa.stattools import kpss
9 from statsmodels.stats.multitest import multipletests
10 from statsmodels.tsa.holtwinters import ExponentialSmoothing
11 from sklearn.preprocessing import StandardScaler
12 from sklearn.preprocessing import KBinsDiscretizer
13 from sklearn.model_selection import StratifiedKFold, KFold
14 warnings.filterwarnings('ignore')
15 sns.set_style('whitegrid')
```

## 0. Загрузка данных

In [2]:

```
1 train = pd.read_csv('data/train.csv')
2 test = pd.read_csv('data/test.csv')
3
4 print(train.shape)
5 print(test.shape)
6 print(train.head())
```

```
(5000000, 3)
(2000000, 2)
   time  signal  open_channels
0  0.0001 -2.7600             0
1  0.0002 -2.8557             0
2  0.0003 -2.4074             0
3  0.0004 -3.1404             0
4  0.0005 -3.1525             0
```

## 1. Снижение количества потребляемой памяти

Т.к. датасет большой, то для скорости работы и избежания ошибок out of memory, нужно максимально "ужать" типы данных в датасете, чтобы он занимал меньше места. Будем смотреть на максимальные по модулю значения и подгонять под них тип данных.

Идея для реализации взята из этого ноутбука: <https://www.kaggle.com/teejmahal20/ion-550-features-lightgbm> (<https://www.kaggle.com/teejmahal20/ion-550-features-lightgbm>).

```

In [3]: 1 def reduce_mem_usage(df, verbose=True):
2         numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
3         start_mem = df.memory_usage().sum() / 1024 ** 2
4         for col in df.columns:
5             col_type = df[col].dtypes
6             if col_type in numerics:
7                 c_min = df[col].min()
8                 c_max = df[col].max()
9                 if str(col_type)[:3] == 'int':
10                    if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
11                        df[col] = df[col].astype(np.int8)
12                    elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
13                        df[col] = df[col].astype(np.int16)
14                    elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
15                        df[col] = df[col].astype(np.int32)
16                    elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
17                        df[col] = df[col].astype(np.int64)
18                else:
19                    if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
20                        df[col] = df[col].astype(np.float16)
21                    elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
22                        df[col] = df[col].astype(np.float32)
23                    else:
24                        df[col] = df[col].astype(np.float64)
25            end_mem = df.memory_usage().sum() / 1024 ** 2
26            if verbose:
27                print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(
28                    start_mem - end_mem) / start_mem)
29            return df

```

```

In [4]: 1 train = reduce_mem_usage(train)

```

Mem. usage decreased to 23.84 Mb (79.2% reduction)

Как видим, даже для датасета без доп. фичей удаётся снизить память почти на 80%. В дальнейшем будем еще часто обращаться к этой функции.

## 2. Скользящие статистики

Полезными для обучения моделей могут стать скользящие статистики:

$$y_t = f(y_{t-1}, \dots, y_{t-\text{window-size}})$$

В качестве  $f$  можно брать среднее, медиану, дисперсию и т.д.

Ширину окна можно подбирать разной, поэтому в метод добавления признаков в датафрейм будет передаваться список из разных вариантов для ширины. Также т.к. данные представляют собой независимые батчи по 500000 измерений, статистики будем считать отдельно по батчам (в каждом батче первые window-size значений будем заполнять либо нулями, либо просто значением сигнала).

Некоторые идеи о том, какие могут быть фичи, взяты из этого discussion:

<https://www.kaggle.com/c/liverpool-ion-switching/discussion/134648>

(<https://www.kaggle.com/c/liverpool-ion-switching/discussion/134648>)

```
In [5]: 1 def add_rolling_features(df, window_sizes, multibatch=True):
2         num_objects = df.shape[0]
3         batch_size = 500*(10**3)
4         num_batches = num_objects // batch_size
5
6         df['batch'] = df.index // batch_size
7
8         for window in tqdm_notebook(window_sizes):
9             df["rolling_mean_" + str(window)] = df['signal'].rolling(window=window)
10            df["rolling_std_" + str(window)] = df['signal'].rolling(window=window)
11            df["rolling_var_" + str(window)] = df['signal'].rolling(window=window)
12            df["rolling_min_" + str(window)] = df['signal'].rolling(window=window)
13            df["rolling_max_" + str(window)] = df['signal'].rolling(window=window)
14            df["rolling_median_" + str(window)] = df['signal'].rolling(window=window)
15
16            df["rolling_min_max_ratio_" + str(window)] = df["rolling_min_" + str(window)] / df["rolling_max_" + str(window)]
17            df["rolling_min_max_diff_" + str(window)] = df["rolling_max_" + str(window)] - df["rolling_min_" + str(window)]
18
19            a = (df['signal'] - df['rolling_min_' + str(window)]) \
20                / (df['rolling_max_' + str(window)] - df['rolling_min_' + str(window)])
21            df["norm_" + str(window)] = a * (np.floor(df['rolling_max_' + str(window)]) - np.ceil(df['rolling_min_' + str(window)]))
22
23            df = df.replace([np.inf, -np.inf], np.nan)
24            df.fillna(0, inplace=True)
25
26         return df
```

```
In [6]: 1 window_sizes = [5, 100, 500, 5000]
2         train = add_rolling_features(train, window_sizes)
```

```
HBox(children=(IntProgress(value=0, max=4), HTML(value='')))
```

In [7]:

```
1 train.head()
```

Out[7]:

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_mi
0	0.0001	-2.759766	0	0	0.000000	0.000000	0.000000	0.000
1	0.0002	-2.855469	0	0	0.000000	0.000000	0.000000	0.000
2	0.0003	-2.408203	0	0	0.000000	0.000000	0.000000	0.000
3	0.0004	-3.140625	0	0	0.000000	0.000000	0.000000	0.000
4	0.0005	-3.152344	0	0	-2.863281	0.307551	0.094587	-3.152

5 rows × 40 columns



In [8]:

```
1 train = reduce_mem_usage(train)
```

Mem. usage decreased to 371.93 Mb (74.7% reduction)

### 3. Центрирование и стандартизация

Многим моделям нужны масштабированные признаки для работы, поэтому функция масштабирования лишней не будет. Также это будет полезно для сокращения используемой памяти.

In [9]:

```
1 def scaling(df):  
2     scaler = StandardScaler()  
3     return scaler.fit_transform(df)
```

### 4. Экспоненциальное сглаживание

При проведении EDA выяснилось, что экспоненциальное сглаживание хорошо убирает дисперсию, при этом сохраняет общий тренд и значимые колебания. Поэтому добавим сглаживание в признаки.

```
In [10]: 1 def exp_array_smoothing(y, alpha):
2         res = np.zeros(len(y))
3         res[0] = y[0]
4
5         for i in range(1, len(y)):
6             res[i] = res[i-1] + alpha*(y[i] - res[i-1])
7
8         return res
9
10 def exponential_smoothing(df, alphas):
11     for alpha in alphas:
12         df['exp_' + str(alpha)] = exp_array_smoothing(np.array(df['signal']))
13                                     alpha)
14
15     return df
```

```
In [11]: 1 train = reduce_mem_usage(exponential_smoothing(train, alphas=[0.5, 0.1]))
```

Mem. usage decreased to 391.01 Mb (12.8% reduction)

## 5. Сдвиги

Если верить указанному выше обсуждению на kaggle, то важными получаются признаки сдвига сигнала. Добавим их.

```
In [12]: 1 def signal_shifts(df, shifts):
2         for shift in shifts:
3             df['shift_' + str(shift)] = df.signal.shift(shift)
4
5         df = df.replace([np.inf, -np.inf], np.nan)
6         df.fillna(0, inplace=True)
7
8         return df
```

```
In [13]: 1 train = reduce_mem_usage(signal_shifts(train, shifts=[1,2, -1, -2]))
```

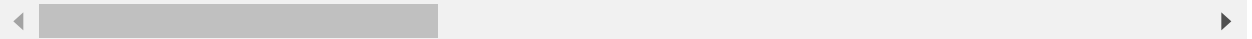
Mem. usage decreased to 429.15 Mb (13.5% reduction)

In [14]: 1 train.head()

Out[14]:

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_mi
0	0.0001	-2.759766	0	0	0.000000	0.000000	0.000000	0.000
1	0.0002	-2.855469	0	0	0.000000	0.000000	0.000000	0.000
2	0.0003	-2.408203	0	0	0.000000	0.000000	0.000000	0.000
3	0.0004	-3.140625	0	0	0.000000	0.000000	0.000000	0.000
4	0.0005	-3.152344	0	0	-2.863281	0.307617	0.094604	-3.152

5 rows × 46 columns



## 6. Агрегация статистик по батчам

Будем разбивать данные на батчи (размер батчей регулируется параметром `batch_sizes`) и внутри каждого такого батча считать разные статистики.

In [15]:

```
1 def batch_stats(df, batch_sizes):
2     for batch_size in batch_sizes:
3         df['tmp_index'] = df.index // batch_size
4         d = {}
5         d[f'mean_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'].m
6         d[f'median_batch{batch_size}'] = df.groupby(['tmp_index'])['signal']
7         d[f'max_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'].ma
8         d[f'min_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'].mi
9         d[f'std_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'].st
10        d[f'mean_abs_chg_batch{batch_size}'] = df.groupby(['tmp_index'])['si
11        d[f'abs_max_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'
12        d[f'abs_min_batch{batch_size}'] = df.groupby(['tmp_index'])['signal'
13        d[f'max-min_batch{batch_size}'] = d[f'max_batch{batch_size}'] - \
14            d[f'min_batch{batch_size}']
15        d[f'max/min_batch{batch_size}'] = d[f'max_batch{batch_size}'] / d[f'
16        d[f'abs_avg_batch{batch_size}'] = (d[f'abs_min_batch{batch_size}'] +
17        for v in d:
18            df[v] = df['tmp_index'].map(d[v].to_dict())
19
20        df = df.drop(columns=['tmp_index'])
21
22    return df
```



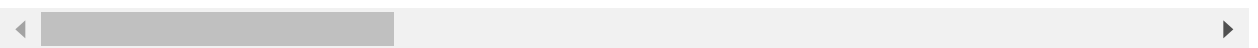
In [16]: 1 train = batch\_stats(train, [25000, 2500])

In [17]: 1 train.head()

Out[17]:

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_mi
0	0.0001	-2.759766	0	0	0.000000	0.000000	0.000000	0.000
1	0.0002	-2.855469	0	0	0.000000	0.000000	0.000000	0.000
2	0.0003	-2.408203	0	0	0.000000	0.000000	0.000000	0.000
3	0.0004	-3.140625	0	0	0.000000	0.000000	0.000000	0.000
4	0.0005	-3.152344	0	0	-2.863281	0.307617	0.094604	-3.152

5 rows × 68 columns



## 7. Вычитание сигнала из статистик

In [18]:

```
1 def add_minus_signal(df):
2     for feat in [feat_ for feat_ in df.columns if feat_ not in ['time', 'sig
3         df[feat + '_msignal'] = df[feat] - df['signal']
4
5     return df
```

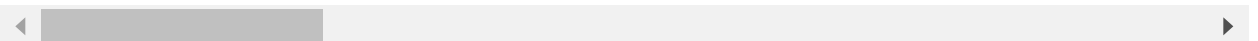
In [19]: 1 train = add\_minus\_signal(train)

In [20]: 1 train.head()

Out[20]:

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_mi
0	0.0001	-2.759766	0	0	0.000000	0.000000	0.000000	0.000
1	0.0002	-2.855469	0	0	0.000000	0.000000	0.000000	0.000
2	0.0003	-2.408203	0	0	0.000000	0.000000	0.000000	0.000
3	0.0004	-3.140625	0	0	0.000000	0.000000	0.000000	0.000
4	0.0005	-3.152344	0	0	-2.863281	0.307617	0.094604	-3.152

5 rows × 132 columns



In [21]: 1 train = reduce\_mem\_usage(train)

Mem. usage decreased to 1306.53 Mb (47.9% reduction)

## 8. Убираем n объектов из конца и начала каждого батча.

Т.к. у нас есть rolling признаки, и признаки, агрегированные по батчам, то имеет смысл выкидывать из батчей объекты, для которых эти признаки считаются некорректно.

```
In [22]: 1 def delete_objects_after_rolling(df, n):
2         num_batches = df.shape[0] // 500000
3         indices_to_delete = []
4         for i in range(num_batches):
5             indices_to_delete += list(range(i*500000, i*500000+n))
6
7         df = df.drop(index=indices_to_delete)
8
9         return df
```

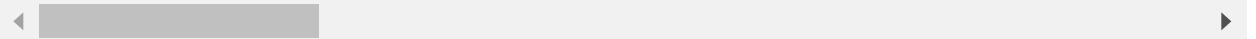
```
In [23]: 1 train = delete_objects_after_rolling(train, 10)
```

```
In [24]: 1 train.head()
```

```
Out[24]:
```

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_r
10	0.0011	-3.113281	0	0	-2.765625	0.202759	0.041107	-3.11
11	0.0012	-2.623047	0	0	-2.751953	0.211670	0.044830	-3.11
12	0.0013	-2.732422	0	0	-2.779297	0.194336	0.037750	-3.11
13	0.0014	-2.902344	0	0	-2.826172	0.189087	0.035736	-3.11
14	0.0015	-2.773438	0	0	-2.828125	0.187744	0.035248	-3.11

5 rows × 132 columns



## 9. Квантили сигнала

```
In [2]: 1 def add_quantiles(train, test, n_bins_arr):
2         for n_bins in n_bins_arr:
3             binner = KBinsDiscretizer(n_bins, encode='ordinal')
4             binner.fit(train.signal.values.reshape(-1, 1))
5             train[f'quant_{n_bins}'] = binner.transform(train.signal.values.reshape(-1, 1))
6             test[f'quant_{n_bins}'] = binner.transform(test.signal.values.reshape(-1, 1))
```

```
In [26]: 1 add_quantiles(train, test, [3, 7])
```

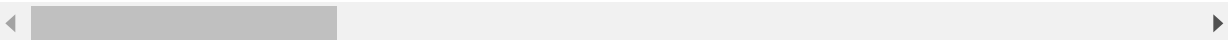


In [27]: 1 train.head()

Out[27]:

	time	signal	open_channels	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_r
10	0.0011	-3.113281	0	0	-2.765625	0.202759	0.041107	-3.11
11	0.0012	-2.623047	0	0	-2.751953	0.211670	0.044830	-3.11
12	0.0013	-2.732422	0	0	-2.779297	0.194336	0.037750	-3.11
13	0.0014	-2.902344	0	0	-2.826172	0.189087	0.035736	-3.11
14	0.0015	-2.773438	0	0	-2.828125	0.187744	0.035248	-3.11

5 rows × 134 columns



In [28]: 1 test.head()

Out[28]:

	time	signal	quant_3	quant_7
0	500.0001	-2.6498	0	0
1	500.0002	-2.8494	0	0
2	500.0003	-2.8600	0	0
3	500.0004	-2.4350	0	1
4	500.0005	-2.6155	0	0

## 10. Target encoding

За счет квантилей у нас появились категориальные признаки, к которым можно применить target encoding, причем брать можно не только среднее таргета, но и другие статистики.

In [30]:

```
1 def add_target_encoding(train, test, n_bins_arr):
2     # обычный target encoding для теста
3     for n_bins in tqdm_notebook(n_bins_arr):
4         train_quant_channel = train[[f'quant_{n_bins}', 'open_channels']]
5         train_encoding_mean = train_quant_channel.groupby(f'quant_{n_bins}').
6         train_encoding_std = train_quant_channel.groupby(f'quant_{n_bins}').
7         train_encoding_var = train_quant_channel.groupby(f'quant_{n_bins}').
8
9         d = {}
10        for q, v in zip(train_encoding_mean.index.values,
11                        train_encoding_mean['open_channels'].values):
12            if q not in d:
13                d[q] = v
14        test_values = []
15        for q in test[f'quant_{n_bins}'].values:
16            test_values.append(d[q])
17        test[f'quant_{n_bins}_mean'] = test_values
18
19        d = {}
20        for q, v in zip(train_encoding_std.index.values,
21                        train_encoding_std['open_channels'].values):
22            if q not in d:
23                d[q] = v
24        test_values = []
25        for q in test[f'quant_{n_bins}'].values:
26            test_values.append(d[q])
27        test[f'quant_{n_bins}_std'] = test_values
28
29        d = {}
30        for q, v in zip(train_encoding_var.index.values,
31                        train_encoding_var['open_channels'].values):
32            if q not in d:
33                d[q] = v
34        test_values = []
35        for q in test[f'quant_{n_bins}'].values:
36            test_values.append(d[q])
37        test[f'quant_{n_bins}_var'] = test_values
38
39    for n_bins in n_bins_arr:
40        train[f'quant_{n_bins}_mean'] = np.zeros(train.shape[0])
41        train[f'quant_{n_bins}_var'] = np.zeros(train.shape[0])
42        train[f'quant_{n_bins}_std'] = np.zeros(train.shape[0])
43
44    # cv loop для train
45    n_fold = 5
46    folds = KFold(n_splits=n_fold, shuffle=True, random_state=17)
47    for training_index, validation_index in folds.split(train):
48        print(training_index)
49        print(validation_index)
50        print()
51        x_train = train.iloc[training_index]
52        x_validation = train.iloc[validation_index]
53        for n_bins in n_bins_arr:
54            column = f'quant_{n_bins}'
55            print(x_train)
56            means = x_validation[column].map(x_train.groupby(column).open_ch
```

```

57         stds = x_validation[column].map(x_train.groupby(column).open_cha
58         vars_ = x_validation[column].map(x_train.groupby(column).open_ch
59
60         x_validation[f'quant_{n_bins}_mean'] = means
61         x_validation[f'quant_{n_bins}_std'] = stds
62         x_validation[f'quant_{n_bins}_var'] = vars_
63         print(x_validation)
64         print()
65         train.iloc[validation_index] = x_validation

```

```

In [41]: 1 %%time
        2 add_target_encoding(train, test, [3, 7])

```

HBox(children=(IntProgress(value=0, max=2), HTML(value='')))

Wall time: 42.9 s

```

In [44]: 1 test.head()

```

```

Out[44]:
      time  signal  quant_3  quant_7  quant_3_mean  quant_3_std  quant_3_var  quant_7_mean
0  500.0001 -2.6498        0        0      0.354876      0.531318      0.282298      0.001691
1  500.0002 -2.8494        0        0      0.354876      0.531318      0.282298      0.001691
2  500.0003 -2.8600        0        0      0.354876      0.531318      0.282298      0.001691
3  500.0004 -2.4350        0        1      0.354876      0.531318      0.282298      0.505374
4  500.0005 -2.6155        0        0      0.354876      0.531318      0.282298      0.001691

```

```

In [45]: 1 train.head()

```

```

Out[45]:
      time  signal  open_channels  batch  rolling_mean_5  rolling_std_5  rolling_var_5  rolling_r
10 -1.0000 -3.113281              0      0      -2.765625      0.202759      0.041107      -3.1
11 -2.0000 -2.623047              0      0      -2.751953      0.211670      0.044830      -3.1
12  0.0013 -2.732422              0      0      -2.779297      0.194336      0.037750      -3.1
13  0.0014 -2.902344              0      0      -2.826172      0.189087      0.035736      -3.1
14  0.0015 -2.773438              0      0      -2.828125      0.187744      0.035248      -3.1

```

5 rows × 140 columns

```

In [36]: 1 data = {'signal': np.arange(1, 15), 'open_channels': np.arange(11, 25)}
        2 train = pd.DataFrame.from_dict(data)
        3 test = pd.DataFrame.from_dict(data)

```

```

In [38]: 1 add_quantiles(train, test, [3])

```

