0. Скрипты для загрузки данных, обучения, кросс-валидации, загрузки предсказаний в файл

In [1]:
```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import f1_score
from datetime import datetime
from xgboost.sklearn import XGBRegressor
import gc
```

Загрузка данных

In [2]:
```
from feature_engineering import reduce_mem_usage, add_rolling_features
from feature_engineering import exponential_smoothing, signal_shifts
from feature_engineering import batch_stats2, add_minus_signal
from feature_engineering import delete_objects_after_rolling
from feature_engineering import add_quantiles, add_target_encoding
```

In [3]:
```
def prepare_df(df, window_sizes, alphas, shifts, batch_sizes):
    df = reduce_mem_usage(df)
    df = add_rolling_features(df, window_sizes)
    df = reduce_mem_usage(df)
    df = exponential_smoothing(df, alphas)
    df = reduce_mem_usage(df)
    df = signal_shifts(df, shifts)
    df = reduce_mem_usage(df)
    df = batch_stats2(df, batch_sizes)
    df = reduce_mem_usage(df)
    #df = add_minus_signal(df)
    #df = reduce_mem_usage(df)

    if 'open_channels' in df.columns:
        y = df['open_channels']
        df = df.drop(columns=['time'])
        return df, y
    else:
        df = df.drop(columns=['time'])
        return df
```

In [4]:

```
train = pd.read_csv('data/train_clean.csv')
test = pd.read_csv('data/test_clean.csv')

window_sizes = [5, 100, 1000, 5000]
alphas = [0.5, 0.2, 0.05]
shifts = [1,-1,2,-2]
batch_sizes = [50000, 25000, 2500]


X_train, y_train = prepare_df(train, window_sizes, alphas, shifts, batch_siz
X_test = prepare_df(test, window_sizes, alphas, shifts, batch_sizes)

y_train = np.array(y_train)

add_quantiles(X_train, X_test, [3, 7, 15])
add_target_encoding(X_train, X_test, [3, 7, 15])

X_train = reduce_mem_usage(X_train)
X_test = reduce_mem_usage(X_test)

X_train = X_train.drop(columns=['open_channels'])
```

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

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Mem. usage decreased to 381.47 Mb (74.0% reduction)
Mem. usage decreased to 410.08 Mb (17.3% reduction)
Mem. usage decreased to 448.23 Mb (13.0% reduction)
Mem. usage decreased to 762.94 Mb (55.3% reduction)
Mem. usage decreased to  7.63 Mb (75.0% reduction)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Mem. usage decreased to 154.50 Mb (73.0% reduction)
Mem. usage decreased to 165.94 Mb (17.1% reduction)
Mem. usage decreased to 181.20 Mb (6.9% reduction)
Mem. usage decreased to 307.08 Mb (55.2% reduction)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

```
Mem. usage decreased to 853.54 Mb (26.0% reduction)
Mem. usage decreased to 343.32 Mb (25.9% reduction)
```

In [5]:
```
1 X_test.head()
```

Out[5]:

	signal	batch	rolling_mean_5	rolling_std_5	rolling_var_5	rolling_min_5	rolling_max_5	rolli
0	-2.650391	0	0.000000	0.000000	0.000000	0.000000	0.000000	
1	-2.849609	0	0.000000	0.000000	0.000000	0.000000	0.000000	
2	-2.859375	0	0.000000	0.000000	0.000000	0.000000	0.000000	
3	-2.435547	0	0.000000	0.000000	0.000000	0.000000	0.000000	
4	-2.615234	0	-2.681641	0.177246	0.031433	-2.859375	-2.435547	

5 rows × 90 columns

Кастомная метрика для валидации xgb

In [6]:
```
1 def MacroF1Metric(preds, dtrain):
2     labels = dtrain.get_label()
3     preds = np.round(np.clip(preds, 0, 10)).astype(int)
4     score = f1_score(labels, preds, average = 'macro')
5     return 'MacroF1Metric', score
```

Обучение модели с разбиением на трейн и валидацию

In [7]:
```
1  def fit_model(X_train, y_train, params, num_iterations):
2      print('splitting...')
3      X_train_, X_valid, y_train_, y_valid = train_test_split(X_train, y_train
4                                                              test_size=0.3,
5                                                              random_state=17)
6
7      print('xgb matrix...')
8      train_set = xgb.DMatrix(X_train_, y_train_)
9      #val_set = xgb.DMatrix(X_valid, y_valid)
10
11     del X_train_, y_train_
12     #del X_valid, y_valid
13
14     print('training...')
15     model = xgb.train(params, train_set,
16                       num_boost_round=num_iterations)
17                       #evals=[(val_set, 'val')],
18                       #verbose_eval=1)
19
20     prediction = model.predict(xgb.DMatrix(X_valid))
21     prediction = np.round(np.clip(prediction, 0, 10)).astype(int)
22     score = f1_score(y_valid, prediction, average = 'macro')
23
24     print(f'score = {score}')
25
26     return model
```

Дообучение после каждой print_every итераций, сохраняем модель в файл, валидируемся, выводим скор.

In [13]:

```
def fit_model_with_save(X_train, y_train, params, num_iterations, model_file
    print('splitting...')
    X_train_, X_valid, y_train_, y_valid = train_test_split(X_train, y_train
                                                            test_size=0.3,
                                                            random_state=17)

    print('xgb matrix...')
    train_set = xgb.DMatrix(X_train_, y_train_)
    val = xgb.DMatrix(X_valid)

    del X_train_, y_train_, X_valid

    print(datetime.now().strftime('%Y-%m-%d %H:%M:%S'), 'training...')
    print_every = 15

    # первое обучение
    model = xgb.train(params, train_set,
                      num_boost_round=print_every)
    model.save_model(model_file)
    prediction = model.predict(val)
    prediction = np.round(np.clip(prediction, 0, 10)).astype(int)
    score = f1_score(y_valid, prediction, average = 'macro')
    print(datetime.now().strftime('%Y-%m-%d %H:%M:%S'), '; ',
          f'score = {score}', ';', f'iter = {print_every}')

    # обучаемся дальше
    for i in range((num_iterations - print_every)//print_every):
        model = xgb.train(params, train_set,
                          num_boost_round=print_every,
                          xgb_model=model_file)
        model.save_model(model_file)
        prediction = model.predict(val)
        prediction = np.round(np.clip(prediction, 0, 10)).astype(int)
        score = f1_score(y_valid, prediction, average = 'macro')
        print(datetime.now().strftime('%Y-%m-%d %H:%M:%S'), ';',
              f'score = {score}', ';', f'iter = {(i+2)*print_every}')

    return model
```

Кросс валидация с возвращением scores, oof предсказаний и предсказаний на тесте на каждом шаге

In [9]:

```python
def xgb_cv_loop(X_train, y_train, X_test, params, num_iterations):
    n_fold = 5
    folds = KFold(n_splits=n_fold, shuffle=True, random_state=17)

    oof = np.zeros(X_train.shape[0])
    prediction = np.zeros(X_test.shape[0])
    scores = []

    for training_index, validation_index in tqdm_notebook(folds.split(X_trai
        # разбиение на трэйн и валидацию
        X_train_ = X_train.iloc[training_index].values
        y_train_ = y_train[training_index]
        X_valid = X_train.iloc[validation_index].values
        y_valid = y_train[validation_index]

        # обучение модели
        train_set = xgb.DMatrix(X_train_, y_train_)
        del X_train_, y_train_
        model = xgb.train(params, train_set, num_iterations)

        # скор на валидации
        X_val_ = xgb.DMatrix(X_valid)
        del X_valid
        preds = model.predict(X_val_)
        oof[validation_index] = preds.reshape(-1,)
        preds = np.round(np.clip(preds, 0, 10)).astype(int)
        score = f1_score(y_valid, preds, average = 'macro')
        scores.append(score)

        # предсказание на тесте
        X_test_ = xgb.DMatrix(X_test)
        preds = model.predict(X_test_)
        prediction += preds

        print(f'score: {score}')

    prediction /= n_fold
    prediction = np.round(np.clip(prediction, 0, 10)).astype(int)

    return scores, oof, prediction
```

Обучение на трэйне, предсказания на тесте и загрузка предсказаний в файл

In [10]:
```
def xgb_final_fit(X_train, y_train, X_test, params, num_iterations,
                  filename):
    train_set = xgb.DMatrix(X_train, y_train)

    model = xgb.train(params, train_set,
                      num_boost_round=num_iterations)

    X_test_ = xgb.DMatrix(X_test)
    preds = model.predict(X_test_)
    preds = np.round(np.clip(preds, 0, 10)).astype(int)

    sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str}
    sample_df['open_channels'] = preds
    sample_df.to_csv(filename, index=False, float_format='%.4f')
```

1. Baseline

Интуитивно подобранные параметры. Просто посмотрим, на какие результаты можно рассчитывать

In [16]:
```
params = {'colsample_bytree': 0.375,
          'learning_rate': 0.1,
          'max_depth': 15,
          'subsample': 0.6,
          'objective':'reg:squarederror',
          'random_state': 17,
          'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}
```

In [13]:
```
num_iterations = 150
model_file = 'baseline'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-22 20:58:48 training...
2020-04-22 21:05:00 ;  score = 0.2514853229113489
2020-04-22 21:11:05 ;  score = 0.9066280718838962
2020-04-22 21:18:04 ;  score = 0.9356554089896456
2020-04-22 21:23:54 ;  score = 0.9364960805419013
2020-04-22 21:29:53 ;  score = 0.9366241019441741
2020-04-22 21:35:53 ;  score = 0.9366606590319936
2020-04-22 21:42:00 ;  score = 0.9365105940507171
2020-04-22 21:48:08 ;  score = 0.9363963739454916
2020-04-22 21:54:13 ;  score = 0.9365528560911008
2020-04-22 22:00:25 ;  score = 0.9365558127127536
```

In [18]:
```
xgb_final_fit(X_train, y_train, X_test, params, 180,
              'xgb1.csv')
```

```
[00:58:39] WARNING: src/learner.cc:686: Tree method is automatically selected t
o be 'approx' for faster speed. To use old behavior (exact greedy algorithm on
single machine), set tree_method to 'exact'.
```

Результат: 0.74 на public lb

2. Подбор гиперпараметров

Кросс-валидацию делать не будем, потому что это занимает ну ооочень много времени

1) Выставляем lr = 0.1 и подбираем оптимальное кол-во базовых моделей

In [11]:
```
params = {'colsample_bytree': 0.8,
          'subsample': 0.8,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 5,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}
```

In [12]:
```
num_iterations = 240
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-23 14:15:02 training...
2020-04-23 14:18:49 ;  score = 0.2554834467327609
2020-04-23 14:22:15 ;  score = 0.8963421300196369
2020-04-23 14:25:33 ;  score = 0.9321451392206783
2020-04-23 14:29:07 ;  score = 0.9338145896679587
2020-04-23 14:32:23 ;  score = 0.934131460133078
2020-04-23 14:35:38 ;  score = 0.9346158157227461
2020-04-23 14:38:49 ;  score = 0.9348994447464274
2020-04-23 14:42:17 ;  score = 0.935092895363558
2020-04-23 14:45:40 ;  score = 0.9352733770589446
2020-04-23 14:48:59 ;  score = 0.9353673372153349
2020-04-23 14:52:16 ;  score = 0.9354925468736269
2020-04-23 14:55:33 ;  score = 0.9355846320619841
2020-04-23 14:58:53 ;  score = 0.9356856854618399
2020-04-23 15:02:09 ;  score = 0.9357058751648332
2020-04-23 15:05:21 ;  score = 0.935755781712166
2020-04-23 15:08:51 ;  score = 0.9357435761580665
```

После 210 итераций скор перестал расти, но пока оставим с запасом 240

2) фиксируем оптимальное кол-во базовых моделей и подбираем max_depth и min_child_weight

In [14]:
```
gc.collect()
```

Out[14]:
```
289
```

In [15]:

```
params = {'colsample_bytree': 0.8,
          'subsample': 0.8,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 8,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 240
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-23 15:20:04 training...
2020-04-23 15:25:23 ;  score = 0.25006823742430356
2020-04-23 15:30:26 ;  score = 0.912823272810611
2020-04-23 15:35:33 ;  score = 0.9368060153883081
2020-04-23 15:41:11 ;  score = 0.9374241853801102
2020-04-23 15:46:24 ;  score = 0.9374653167767765
2020-04-23 15:51:40 ;  score = 0.9375196402162413
2020-04-23 15:56:51 ;  score = 0.937549552978194
2020-04-23 16:02:51 ;  score = 0.9375587758492795
2020-04-23 16:08:21 ;  score = 0.9375751008459111
2020-04-23 16:14:23 ;  score = 0.9376111089557089
2020-04-23 16:19:57 ;  score = 0.9376400798505671
2020-04-23 16:25:44 ;  score = 0.9376723901809817
2020-04-23 16:31:15 ;  score = 0.9376656160130431
2020-04-23 16:36:44 ;  score = 0.9376754383874787
2020-04-23 16:42:24 ;  score = 0.9376577086764635
2020-04-23 16:47:44 ;  score = 0.9376625218435387
```

In [11]:

```
gc.collect()

params = {'colsample_bytree': 0.8,
          'subsample': 0.8,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 240
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-23 17:20:59 training...
2020-04-23 17:27:58 ;  score = 0.2503194481649376
2020-04-23 17:35:02 ;  score = 0.9138654867752734
2020-04-23 17:42:26 ;  score = 0.9368336956019466
2020-04-23 17:49:23 ;  score = 0.9374082400604258
2020-04-23 17:56:15 ;  score = 0.9374970812672777
2020-04-23 18:03:17 ;  score = 0.9375623214346437
2020-04-23 18:11:00 ;  score = 0.9376161451657743
2020-04-23 18:18:07 ;  score = 0.9375875117333036
2020-04-23 18:25:03 ;  score = 0.937709455140595
2020-04-23 18:32:24 ;  score = 0.9377068554739886
2020-04-23 18:40:21 ;  score = 0.9376544036216959
2020-04-23 18:49:05 ;  score = 0.9377614891878415
2020-04-23 18:57:37 ;  score = 0.9377882780678884
2020-04-23 19:06:52 ;  score = 0.9377616578276331
2020-04-23 19:15:58 ;  score = 0.9377937239329167
2020-04-23 19:24:42 ;  score = 0.9376539733991446
```

In [12]:

```
gc.collect()

params = {'colsample_bytree': 0.8,
          'subsample': 0.8,
          'min_child_weight': 3,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 150
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-23 19:37:22 training...
2020-04-23 19:46:14 ;  score = 0.2501612177044458
2020-04-23 19:53:51 ;  score = 0.9137808478451652
2020-04-23 20:01:58 ;  score = 0.9370674609513882
2020-04-23 20:10:09 ;  score = 0.937498926185457
2020-04-23 20:18:06 ;  score = 0.9375460114532341
2020-04-23 20:25:55 ;  score = 0.937623676348955
2020-04-23 20:33:53 ;  score = 0.9376627121558919
2020-04-23 20:41:24 ;  score = 0.9376842206643093
2020-04-23 20:48:21 ;  score = 0.9376003658310953
2020-04-23 20:55:16 ;  score = 0.9376562714149107
```

3) подбираем gamma

In [14]:

```
gc.collect()

params = {'colsample_bytree': 0.8,
          'subsample': 0.8,
          'min_child_weight': 1,
          'gamma': 0.2,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 150
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-23 20:57:16 training...
2020-04-23 21:04:06 ;  score = 0.2506296987539088 ; iter = 15
2020-04-23 21:10:48 ; score = 0.9139050347991468 ; iter = 30
2020-04-23 21:17:27 ; score = 0.9370139031734332 ; iter = 45
2020-04-23 21:24:29 ; score = 0.9374695295735728 ; iter = 60
2020-04-23 21:31:45 ; score = 0.9375796895269083 ; iter = 75
2020-04-23 21:39:00 ; score = 0.9376779571769867 ; iter = 90
2020-04-23 21:46:19 ; score = 0.9376490541078816 ; iter = 105
2020-04-23 21:53:46 ; score = 0.9376201510755025 ; iter = 120
2020-04-23 22:01:20 ; score = 0.9376784071296256 ; iter = 135
2020-04-23 22:08:46 ; score = 0.9376747032184686 ; iter = 150
```

4) подбираем subsample

In [15]:

```
gc.collect()

params = {'colsample_bytree': 0.6,
          'subsample': 0.6,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 150
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-24 00:07:20 training...
2020-04-24 00:13:41 ;  score = 0.25081748530214587 ; iter = 15
2020-04-24 00:19:38 ; score = 0.9126555502112467 ; iter = 30
2020-04-24 00:25:45 ; score = 0.9367373146138956 ; iter = 45
2020-04-24 00:31:37 ; score = 0.9373653116171698 ; iter = 60
2020-04-24 00:37:23 ; score = 0.9375435578604457 ; iter = 75
2020-04-24 00:42:55 ; score = 0.937601320170155 ; iter = 90
2020-04-24 00:48:43 ; score = 0.9376592288963951 ; iter = 105
2020-04-24 00:54:26 ; score = 0.9376345979531446 ; iter = 120
2020-04-24 01:00:16 ; score = 0.9376077333855745 ; iter = 135
2020-04-24 01:05:49 ; score = 0.9376453293658905 ; iter = 150
```

In [16]:

```
gc.collect()

params = {'colsample_bytree': 1.,
          'subsample': 1.,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 150
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-24 01:13:25 training...
2020-04-24 01:22:30 ;  score = 0.251063064380594 ; iter = 15
2020-04-24 01:31:43 ; score = 0.9159318612995222 ; iter = 30
2020-04-24 01:40:56 ; score = 0.9370485494045749 ; iter = 45
2020-04-24 01:50:29 ; score = 0.9377247401053093 ; iter = 60
2020-04-24 01:58:55 ; score = 0.9377206048084582 ; iter = 75
2020-04-24 02:08:30 ; score = 0.937720093571756 ; iter = 90
2020-04-24 10:32:37 ; score = 0.93776339305632 ; iter = 105
2020-04-24 10:42:21 ; score = 0.9378310238524001 ; iter = 120
2020-04-24 10:51:42 ; score = 0.9378398147773944 ; iter = 135
2020-04-24 11:01:32 ; score = 0.9378258351411914 ; iter = 150
```

5) подбираем регуляризацию

In [17]:

```
gc.collect()

params = {'colsample_bytree': 1.,
          'subsample': 1.,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.1,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          'reg_alpha': 0.05,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 150
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-24 11:49:19 training...
2020-04-24 11:59:58 ;  score = 0.25146122830402234 ; iter = 15
2020-04-24 12:10:07 ; score = 0.9163313544606223 ; iter = 30
2020-04-24 12:20:56 ; score = 0.9370002102709702 ; iter = 45
2020-04-24 12:31:31 ; score = 0.9375705060204939 ; iter = 60
2020-04-24 12:42:07 ; score = 0.9376683673301709 ; iter = 75
2020-04-24 12:51:46 ; score = 0.9376570803333721 ; iter = 90
2020-04-24 14:16:31 ; score = 0.9376869417562399 ; iter = 105
2020-04-24 14:26:30 ; score = 0.9376398873257444 ; iter = 120
2020-04-24 14:35:58 ; score = 0.9377006404643846 ; iter = 135
2020-04-24 14:45:30 ; score = 0.9377315938085657 ; iter = 150
```

6) уменьшаем lr

In [ ]:

```
gc.collect()

params = {'colsample_bytree': 1.,
          'subsample': 1.,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.05,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 210
model_file = 'xgb2'
model = fit_model_with_save(X_train, y_train, params, num_iterations,
                            model_file)
```

```
splitting...
xgb matrix...
2020-04-24 14:47:23 training...
2020-04-24 14:56:34 ;  score = 0.16272613132626346 ; iter = 15
2020-04-24 15:05:05 ; score = 0.24746736608246309 ; iter = 30
2020-04-24 15:13:13 ; score = 0.5617137717322215 ; iter = 45
2020-04-24 15:22:04 ; score = 0.9079213359533269 ; iter = 60
2020-04-24 15:31:50 ; score = 0.9338825748073436 ; iter = 75
```

3. Финальные предсказания

In [ ]:

```
gc.collect()

params = {'colsample_bytree': 1.,
          'subsample': 1.,
          'min_child_weight': 1,
          'gamma': 0,
          'learning_rate': 0.05,
          'max_depth': 10,
          'objective':'reg:squarederror',
          'random_state': 17,
          'scale_pos_weight': 1,
          #'sample_fraction': 0.3,
          #'disable_default_eval_metric': 1,
          #'feval': MacroF1Metric,
          #'silent': False,
          'verbosity': 1}

num_iterations = 210
model_file = 'xgb3'
xgb_final_fit(X_train, y_train, X_test, params, num_iterations, model_file)
```

Результат: 0.84 на public lb