В этом ноутбуке происходят попытки выбить максимальный скор из случайного леса. Discussion на kaggle показывают, что лес здесь неплохо справляется

In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tqdm import tqdm_notebook
from sklearn.ensemble import RandomForestClassifier
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import f1_score
import gc
```

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

Загрузка данных

In [2]:
```python
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]:
```python
def prepare_df(df, shifts):
    df = reduce_mem_usage(df)
    df = signal_shifts(df, shifts)
    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')

shifts = list(np.arange(-20, 0)) + list(np.arange(1, 21))

train["category"] = 0
test["category"] = 0

# train segments with more then 9 open channels classes
train.loc[2_000_000:2_500_000-1, 'category'] = 1
train.loc[4_500_000:5_000_000-1, 'category'] = 1

# test segments with more then 9 open channels classes (potentially)
test.loc[500_000:600_000-1, "category"] = 1
test.loc[700_000:800_000-1, "category"] = 1

X_train, y_train = prepare_df(train, shifts)
X_test = prepare_df(test, shifts)

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 28.61 Mb (81.2% reduction)
Mem. usage decreased to 410.08 Mb (14.0% reduction)
Mem. usage decreased to  9.54 Mb (79.2% reduction)
Mem. usage decreased to 162.12 Mb (7.6% reduction)
Mem. usage decreased to 400.54 Mb (0.0% reduction)
Mem. usage decreased to 158.31 Mb (0.0% reduction)
```

Обучение леса

In [5]:

```
def fit_model(X_train, y_train, params):
    gc.collect()

    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('fit...')
    model = RandomForestClassifier(**params)
    model.fit(X_train_, y_train_)

    print('predict...')
    prediction = model.predict(X_valid)
    score = f1_score(y_valid, prediction, average = 'macro')

    print(f'score = {score}')

    return model


def fit_model_with_save(X_train, y_train, X_test, params, modelname):
    gc.collect()

    print('fit...')
    model = RandomForestClassifier(**params)
    model.fit(X_train, y_train)

    print('predict...')
    prediction = model.predict(X_test)
    np.save(modelname + '_test_preds.npy', prediction)

    print('saving predictions...')
    sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str}
    sample_df['open_channels'] = prediction
    sample_df.to_csv(modelname + '.csv', index=False, float_format='%.4f')

    print('probapility...')
    probs = model.predict_proba(X_test)
    np.save(modelname + '_test_probs.npy', probs)

    return model
```

Кросс-валидация

In [6]:

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

    oof = np.zeros(X_train.shape[0])
    oof_probs = np.zeros((X_train.shape[0], 11))

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

    for training_index, validation_index in tqdm_notebook(folds.split(X_trai
        gc.collect()

        # разбиение на трэйн и валидацию
        X_train_ = X_train.iloc[training_index]
        y_train_ = y_train[training_index]
        X_valid = X_train.iloc[validation_index]
        y_valid = y_train[validation_index]

        # обучение модели
        model = RandomForestClassifier(**params)
        model.fit(X_train_, y_train_)

        # скор на валидации
        preds = model.predict(X_valid)
        oof[validation_index] = preds
        score = f1_score(y_valid, preds, average = 'macro')
        scores.append(score)

        # вероятности на валидации
        probs = model.predict_proba(X_valid)
        oof_probs[validation_index] = probs

        # предсказание на месте
        preds = model.predict(X_test)
        prediction += preds

        print(f'score: {score}')

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

    np.save(modelname + '_cv_test_preds.npy', prediction)
    np.save(modelname + '_oof_preds.npy', oof)
    np.save(modelname + '_oof_probs.npy', oof_probs)

    sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str}
    sample_df['open_channels'] = prediction
    sample_df.to_csv(modelname + '_cv.csv', index=False, float_format='%.4f'

    return scores, oof, prediction
```

1. Бейзлайн

Параметры для RF и для признаков возьмем из этого ноутбука: [https://www.kaggle.com/sggpls/shifted-rfc-pipeline (https://www.kaggle.com/sggpls/shifted-rfc-pipeline)](https://www.kaggle.com/sggpls/shifted-rfc-pipeline)

In [7]:

```
%%time

params = {
    'n_estimators': 200,
    'max_depth': 19,
    'max_features': 10,
    'random_state': 17,
    'n_jobs': -1,
    'verbose': 2
}

scores, oof, prediction = cv_loop(X_train, y_train, X_test, params, 'rf1')
```

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.

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
```

In [8]:

```
print(np.mean(scores))
```

```
0.9377745874531744
```

In [7]:
```
%%time
params = {
    'n_estimators': 200,
    'max_depth': 19,
    'max_features': 10,
    'random_state': 17,
    'n_jobs': -1,
    'verbose': 2
}
forest = fit_model_with_save(X_train, y_train, X_test,
                             params, 'rf1')
```

```
fit...

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.

building tree 1 of 200building tree 2 of 200building tree 3 of 200building tr
ee 4 of 200building tree 5 of 200building tree 6 of 200building tree 7 of 200

building tree 8 of 200





building tree 9 of 200building tree 10 of 200

building tree 11 of 200
building tree 12 of 200
building tree 13 of 200
building tree 14 of 200
```

Результат: 0.939 на public lb.

2. Добавим таргет енкодинг, подкрутим параметры

In [11]:

```
%%time

params = {
    'n_estimators': 500,
    'max_depth': 25,
    'max_features': 15,
    'random_state': 17,
    'n_jobs': -1,
    'verbose': 2
}

forest = fit_model(X_train, y_train, params)
```

```
splitting...
fit...

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.

building tree 1 of 500building tree 2 of 500building tree 3 of 500building tr
ee 4 of 500building tree 5 of 500building tree 6 of 500building tree 7 of 500
building tree 8 of 500







building tree 9 of 500
building tree 10 of 500
building tree 11 of 500
building tree 12 of 500
```

In [12]:

```
for feature, imp in zip(X_train.columns, forest.feature_importances_):
    print(feature, imp)
```

```
signal 0.2810031591309477
category 0.10557577662222818
shift_-20 0.004632864496387041
shift_-19 0.003406895845358958
shift_-18 0.0027048016027770012
shift_-17 0.0020497408946804035
shift_-16 0.001785484209258236
shift_-15 0.0016448296427000225
shift_-14 0.0014370097477635424
shift_-13 0.0013014553553040565
shift_-12 0.0013215275808369784
shift_-11 0.0012317645641277744
shift_-10 0.0011798043295531532
shift_-9 0.001240049090643348 5
shift_-8 0.0012042642453490972
shift_-7 0.0013310274181537538
shift_-6 0.0015851816879912362
shift_-5 0.001878256879192865
shift_-4 0.0024917497499253078
shift_-3 0.004939598532460965
shift_-2 0.01272673147324323
shift_-1 0.04282155384271084
shift_1 0.030177157010225886
shift_2 0.007917132601380212
shift_3 0.004828813324772249
shift_4 0.0022441743662006013
shift_5 0.002056906409358257 6
shift_6 0.001755599363317114 7
shift_7 0.001981204290553191 6
shift_8 0.001295086443712815
shift_9 0.0012752507007509001
shift_10 0.0012460415743923575
shift_11 0.001308760799017611
shift_12 0.001293529238323259
shift_13 0.0013616279430836787
shift_14 0.0014013848078526256
shift_15 0.0015343370034235388
shift_16 0.0017743871197682972
shift_17 0.0018204118122319211
shift_18 0.002639553591744839
shift_19 0.0033765028438492535
shift_20 0.004378495921514019
quant_3 0.0016559186504529312
quant_7 0.050809769163690184
quant_15 0.10270333776499588
quant_3_mean 0.00079676505374657 73
quant_3_std 0.002782910556776887
quant_3_var 0.0015300683173072784
quant_7_mean 0.05826341553032381
quant_7_std 0.04388602205179024
quant_7_var 0.04973625940031991
quant_15_mean 0.10956949554724996
```

```
quant_15_std 0.01152907906164337
quant_15_var 0.011577074794636813
```