В этом ноутбуке рассматриваются разные методы стекинга моделей, полученных в других ноутбуках

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from tqdm import tqdm_notebook
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBRegressor, XGBClassifier
from sklearn.model_selection import cross_val_score, KFold
```

In [3]:

```
def pred_proc(pred):
    pred = np.round(np.clip(pred, 0, 10))
    return pred.astype(int)

def cv_loop(oof_df, y, model):
    n_fold = 5
    folds = KFold(n_splits=n_fold, shuffle=True, random_state=17)

    scores = []

    for training_index, validation_index in tqdm_notebook(folds.split(oof_df
        X_train = oof_df.iloc[training_index]
        y_train = y[training_index]
        X_valid = oof_df.iloc[validation_index]
        y_valid = y[validation_index]

        model.fit(X_train, y_train)

        # скор на валидации
        preds = model.predict(X_valid)
        preds = np.round(np.clip(preds, 0, 10)).astype(int)
        score = f1_score(y_valid, preds, average = 'macro')
        scores.append(score)
        print(score)

    return scores
```

1. Стекинг LGBM и Catboost

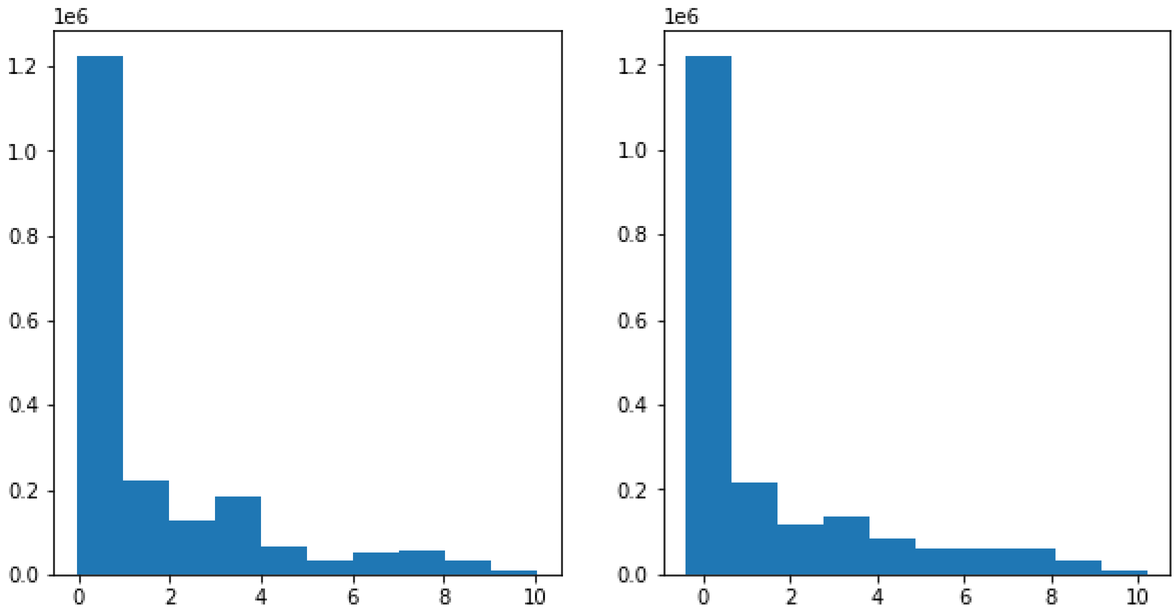
1.1. Усреднение ответов

In [4]:

```
lgb_preds = np.load('for_stacking/lgb13_test_preds.npy')
cb_preds = np.load('for_stacking/preds_best_catboost.npy')
```

In [4]:

```
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.hist(lgb_preds)
plt.subplot(122)
plt.hist(cb_preds)
plt.show()
```



In [5]:

```
preds = 0.5*lgb_preds + 0.5*cb_preds
preds = pred_proc(preds)
sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str})
sample_df['open_channels'] = preds
sample_df.to_csv("lgb13_bestcb.csv", index=False, float_format='%.4f')
```

1.2. Метамодель - линейная регрессия

In [3]:

```
lgb_oof = np.load('for_stacking/lgb13_oof_preds.npy')
cb_oof = np.load('for_stacking/oof_best_catboost.npy')
```

In [4]:

```
oof_df = pd.DataFrame({'lgb_oof': lgb_oof, 'cb_oof': cb_oof})
oof_df.head()
```

Out[4]:

	lgb_oof	cb_oof
0	0.132022	0.030460
1	0.097993	-0.028301
2	0.081194	-0.085979
3	0.138549	0.036849
4	0.060607	0.035619

In [9]:

```python
train = pd.read_csv('data/train_clean.csv')
test = pd.read_csv('data/test_clean.csv')
y = train.open_channels.values
train = train.drop(columns=['open_channels'])
```

In [39]:

```python
X_train, X_val, y_train, y_val = train_test_split(oof_df, y,
                                                  random_state=17,
                                                  test_size=0.3)
```

In [40]:

```python
%%time
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
Wall time: 601 ms
```

Out[40]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [41]:

```python
preds = lr.predict(X_val)
preds = np.round(np.clip(preds, 0, 10)).astype(int)
score = f1_score(y_val, preds, average = 'macro')
print(f'score on val: {score}')
```

```
score on val: 0.937807230681399
```

In [42]:

```python
scores = cv_loop(oof_df, y, LinearRegression())
print(scores)
print(np.mean(scores))
```

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.

```
[0.9378667233513109, 0.938132013690785, 0.9376538834963697, 0.9393951023074935,
0.9386472946846514]
0.938339003506122
```

In [25]:

```python
test_df = pd.DataFrame({'lgb_oof': lgb_preds, 'cb_oof': cb_preds})
```

In [28]:

```python
lr.fit(oof_df, y)
print(lr.coef_)
preds = lr.predict(test_df)
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('lgb_cb_linreg.csv', index=False, float_format='%.4f')
```

```
[0.83597733 0.16406567]
```

Результат: 0.940 на public lb

1.3. Метамодель - XGBRegressor

In [32]:

```
%%time
xg = XGBRegressor()
xg.fit(X_train, y_train)
preds = xg.predict(X_val)
preds = np.round(np.clip(preds, 0, 10)).astype(int)
score = f1_score(y_val, preds, average = 'macro')
print(f'score on val: {score}')
```

```
[11:22:54] WARNING: src/objective/regression_obj.cu:152: reg:linear is now depr
ecated in favor of reg:squarederror.
score on val: 0.9377933690895529
Wall time: 1min 51s
```

In [33]:

```
xg.fit(oof_df, y)
preds = xg.predict(test_df)
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('lgb_cb_xgb.csv', index=False, float_format='%.4f')
```

```
[11:25:25] WARNING: src/objective/regression_obj.cu:152: reg:linear is now depr
ecated in favor of reg:squarederror.
[11:25:25] 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: 0.824 на public lb

2. Добавим oof-предсказания RandomForest

2.1. Метамодель - линейная регрессия

In [6]:

```
rf_oof = np.load('for_stacking/rf1_oof_preds.npy')
lgb_oof = np.load('for_stacking/lgb13_oof_preds.npy')
cb_oof = np.load('for_stacking/oof_best_catboost.npy')
```

In [7]:

```python
oof_df = pd.DataFrame({'lgb_oof': lgb_oof, 'cb_oof': cb_oof,
                       'rf_oof': rf_oof})
oof_df.head()
```

Out[7]:

	lgb_oof	cb_oof	rf_oof
0	0.132022	0.030460	0.0
1	0.097993	-0.028301	0.0
2	0.081194	-0.085979	0.0
3	0.138549	0.036849	0.0
4	0.060607	0.035619	0.0

In [7]:

```python
train = pd.read_csv('data/train_clean.csv')
test = pd.read_csv('data/test_clean.csv')
y = train.open_channels.values
train = train.drop(columns=['open_channels'])
```

In [13]:

```python
X_train, X_val, y_train, y_val = train_test_split(oof_df, y,
                                                  random_state=17,
                                                  test_size=0.3)

```

In [18]:

```python
%%time
lr = LinearRegression()
lr.fit(X_train, y_train)

print(lr.coef_)
```

```
[0.7904124  0.16517715 0.04447739]
Wall time: 818 ms
```

In [19]:

```python
preds = lr.predict(X_val)
preds = np.round(np.clip(preds, 0, 10)).astype(int)
score = f1_score(y_val, preds, average = 'macro')

print(f'score on val: {score}')
```

```
score on val: 0.9378588402608
```

In [30]:

```python
scores = cv_loop(oof_df, y, LinearRegression())
print(scores)
print(np.mean(scores))
```

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.

```
[0.9378524357477883, 0.9381910752966391, 0.9376301521633792, 0.939490083507193
2, 0.9385950090858696]
0.9383517511601738
```

3. Добавим вероятности из ноутбука по RandomForest

3.1. Метамодель - LinearRegression()

In [8]:

```python
lgb_oof = np.load('for_stacking/lgb13_oof_preds.npy')
cb_oof = np.load('for_stacking/oof_best_catboost.npy')
rf_oof_probs = np.load('for_stacking/rf_train_probs.npy')

oof_df = pd.DataFrame({'lgb_oof': lgb_oof,
                       'cb_oof': cb_oof})
for i in range(11):
    oof_df[f'rf_oof_prob{str(i)}'] = rf_oof_probs[:, i]
```

In [9]:

```python
scores = cv_loop(oof_df, y, LinearRegression())
print(scores)
print(np.mean(scores))
```

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.

```
0.9379607836734138
0.9381939577977526
0.9377431802136612
0.9395857113729804
0.9387336790542435

[0.9379607836734138, 0.9381939577977526, 0.9377431802136612, 0.939585711372980
4, 0.9387336790542435]
0.9384434624224103
```

In [10]:

```python
%%time
X_train, X_val, y_train, y_val = train_test_split(oof_df, y,
                                                  random_state=17,
                                                  test_size=0.3)

lr = LinearRegression()
lr.fit(X_train, y_train)

print(lr.coef_)

preds = lr.predict(X_val)
preds = np.round(np.clip(preds, 0, 10)).astype(int)
score = f1_score(y_val, preds, average = 'macro')

print(f'score on val: {score}')
```

```
[ 7.25332303e-01  1.31141734e-01 -7.20452826e-01 -5.76616182e-01
 -4.33604634e-01 -2.89085199e-01 -1.45589179e-01 -3.09331149e-04
  1.39950744e-01  2.85790302e-01  4.30085623e-01  5.75450079e-01
  7.34380604e-01]
score on val: 0.9378757522818262
Wall time: 3.43 s
```

In [75]:

```python
lgb_preds = np.load('for_stacking/lgb13_test_preds.npy')
cb_preds = np.load('for_stacking/preds_best_catboost.npy')
rf_probs = np.load('for_stacking/rf_test_probs.npy')

test_df = pd.DataFrame({'lgb_oof': lgb_preds, 'cb_oof': cb_preds})
for i in range(11):
    test_df[f'rf_oof_prob{str(i)}'] = rf_probs[:, i]
```

In [12]:

```python
lr = LinearRegression()
lr.fit(oof_df, y)

print(lr.coef_)

preds = lr.predict(test_df)
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('stacking3.1.1.csv', index=False, float_format='%.4f')
```

```
[ 7.23415356e-01  1.32802225e-01 -1.19220250e+10 -1.19220250e+10
 -1.19220250e+10 -1.19220250e+10 -1.19220250e+10 -1.19220250e+10
 -1.19220250e+10 -1.19220250e+10 -1.19220250e+10 -1.19220250e+10
 -1.19220250e+10]
```

3.2. Метамодель - XGBClassifier()

In [7]:

```python
scores = cv_loop(oof_df, y, XGBClassifier(verbosity=1,
                                          n_estimators=1))
print(scores)
print(np.mean(scores))
```

```
...
```

3.3. Метамодель - CatBoostClassifier

In [71]:

```python
from catboost import CatBoostClassifier
```

In [72]:

```python
cb = CatBoostClassifier(iterations=1500,
                        verbose=1)
cb.fit(oof_df, y)
```

```
Learning rate set to 0.086731
0:	learn: 1.7197795	total: 5.87s	remaining: 2h 26m 39s
1:	learn: 1.4436677	total: 10.1s	remaining: 2h 6m 15s
2:	learn: 1.2474957	total: 14.2s	remaining: 1h 58m 20s
3:	learn: 1.0976631	total: 18.4s	remaining: 1h 54m 32s
4:	learn: 0.9609647	total: 22.5s	remaining: 1h 52m 10s
5:	learn: 0.8643736	total: 26.6s	remaining: 1h 50m 25s
6:	learn: 0.7827403	total: 30.8s	remaining: 1h 49m 34s
7:	learn: 0.7139835	total: 35.1s	remaining: 1h 49m
8:	learn: 0.6556434	total: 39.4s	remaining: 1h 48m 42s
9:	learn: 0.6031464	total: 43.7s	remaining: 1h 48m 27s
10:	learn: 0.5563732	total: 47.9s	remaining: 1h 48m 5s
11:	learn: 0.5158112	total: 52.2s	remaining: 1h 47m 56s
12:	learn: 0.4792302	total: 56.4s	remaining: 1h 47m 31s
13:	learn: 0.4472814	total: 1m	remaining: 1h 47m 26s
14:	learn: 0.4181340	total: 1m 4s	remaining: 1h 47m 12s
15:	learn: 0.3918102	total: 1m 9s	remaining: 1h 47m 8s
16:	learn: 0.3652644	total: 1m 13s	remaining: 1h 47m 8s
17:	learn: 0.3438955	total: 1m 18s	remaining: 1h 47m 6s
```

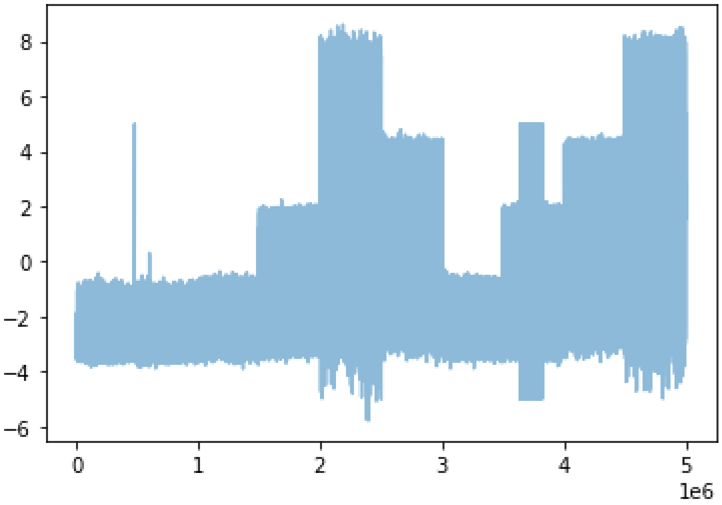
In [79]:

```python
preds = cb.predict(test_df)
preds = pred_proc(preds)
sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str})
sample_df['open_channels'] = preds
sample_df.to_csv("stacking3.3.csv", index=False, float_format='%.4f')
```

4. Разный блендинг на разных батчах

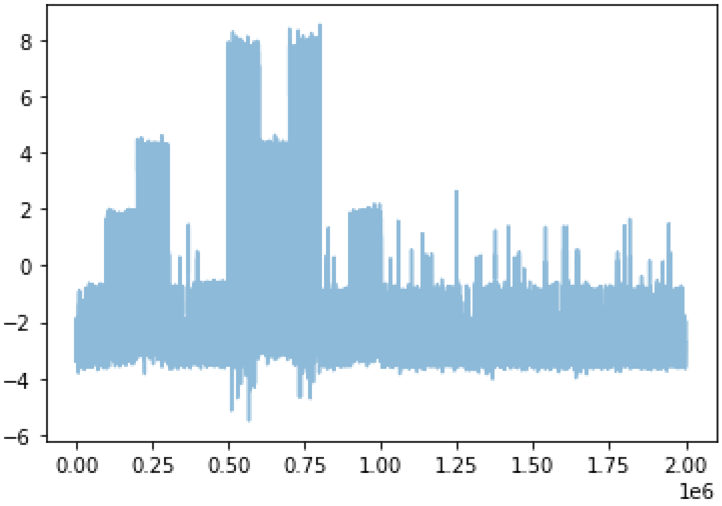
In [31]:
```python
plt.plot(np.arange(0, len(train.signal.values)), train.signal.values,
          alpha=0.5)
```

Out[31]:
```
[<matplotlib.lines.Line2D at 0x1de67580608>]
```



In [32]:
```python
plt.plot(np.arange(0, len(test.signal.values)), test.signal.values,
          alpha=0.5)
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x1de675e2c48>]
```



In [26]:

```python
for i in range(10):
    print(np.mean(train.signal.values[i*500000:(i+1)*500000]))
```

```
-2.681406359199999
-2.6831050761904818
-1.8131248770000001
-0.09298384679999996
3.3578080268
1.6808430502000005
-1.8039282956274345
-0.10786146542743458
1.658963467372566
3.3109071119725657
```

In [30]:

```python
for i in range(4):
    print(np.mean(test.signal.values[i*500000:(i+1)*500000]))
```

```
-1.076758010171455
1.119842871828547
-2.5992788104274345
-2.6210051965999996
```

In [28]:

```python
preds = np.zeros(2*10**6)
```

Для первого батча из теста подбираем коэффициенты по 3, 4, 7, 8 батчам из трейна

In [47]:

```python
train34 = oof_df.iloc[np.arange(2*500000, 4*500000), ]
train78 = oof_df.iloc[np.arange(6*500000, 8*500000), ]
train3478 = pd.concat([train34, train78])
y3478 = y[list(np.arange(2*500000, 4*500000)) + list(np.arange(6*500000, 8*5
```

In [54]:

```python
lr = LinearRegression()
lr.fit(train3478, y3478)
print(lr.coef_)
preds3478 = lr.predict(test_df.iloc[np.arange(0, 500000)])
preds[np.arange(0, 500000)] = preds3478
```

```
[ 6.52878545e-01  9.44760369e-02 -1.73687652e+10 -1.73687652e+10
 -1.73687652e+10 -1.73687652e+10 -1.73687652e+10 -1.73687652e+10
 -1.73687652e+10 -1.73687652e+10 -1.73687652e+10 -1.73687652e+10
 -1.73687652e+10]
```

Для второго батча из теста подбираем коэффициенты по 5,6,9,10 батчам из трейна

In [56]:

```
train56 = oof_df.iloc[np.arange(4*500000, 6*500000), ]
train910 = oof_df.iloc[np.arange(8*500000, 10*500000), ]
train5691 = pd.concat([train56, train910])
y5691 = y[list(np.arange(4*500000, 6*500000)) + list(np.arange(8*500000, 10*
lr = LinearRegression()
lr.fit(train5691, y5691)
print(lr.coef_)
preds5691 = lr.predict(test_df.iloc[np.arange(500000, 2*500000)])
preds[np.arange(500000, 2*500000)] = preds5691
```

```
[ 0.73059474  0.14162184 -0.67248281 -0.51879788 -0.38423512 -0.2542972
 -0.12451996  0.00411493  0.12924003  0.25904027  0.38739255  0.51733709
  0.6572081 ]
```

Для третьего и четвертого батча из теста по 1 и 2 батчам из трейна

In [57]:

```
train12 = oof_df.iloc[np.arange(0*500000, 2*500000), ]
y12 = y[list(np.arange(0*500000, 2*500000))]
lr = LinearRegression()
lr.fit(train12, y12)
print(lr.coef_)
preds12 = lr.predict(test_df.iloc[np.arange(2*500000, 4*500000)])
preds[np.arange(2*500000, 4*500000)] = preds12
```

```
[ 3.93108197e-01  7.63297246e-02 -1.42249995e+10 -1.42249995e+10
 -1.42249995e+10 -1.42249995e+10 -1.42249996e+10 -1.42249995e+10
 -1.42249900e+10  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00]
```

In [59]:

```
preds = pred_proc(preds)
sample_df = pd.read_csv("data/sample_submission.csv", dtype={'time':str})
sample_df['open_channels'] = preds
sample_df.to_csv("stacking4.csv", index=False, float_format='%.4f')
```

Результат: 0.938 на lb

5. Добавим результаты алгоритма Витерби

5.1. Метамодель - линейнеая регрессия

In [78]:

```
# исходные данные
train = pd.read_csv('data/train_clean.csv')
test = pd.read_csv('data/test_clean.csv')
y = train.open_channels.values
train = train.drop(columns=['open_channels'])


# трейн
lgb_oof = np.load('for_stacking/lgb13_oof_preds.npy')
cb_oof = np.load('for_stacking/oof_best_catboost.npy')
rf_oof_probs = np.load('for_stacking/rf_train_probs.npy')
vit_oof_preds = np.load('for_stacking/oof_viterbi.npy')
oof_df = pd.DataFrame({'lgb_oof': lgb_oof,
                       'cb_oof': cb_oof,
                       'viterbi': vit_oof_preds})
for i in range(11):
    oof_df[f'rf_oof_prob{str(i)}'] = rf_oof_probs[:, i]


# тест
lgb_preds = np.load('for_stacking/lgb13_test_preds.npy')
cb_preds = np.load('for_stacking/preds_best_catboost.npy')
rf_probs = np.load('for_stacking/rf_test_probs.npy')
vit_preds = np.load('for_stacking/preds_viterbi.npy')
test_df = pd.DataFrame({'lgb_oof': lgb_preds,
                        'cb_oof': cb_preds,
                        'viterbi': vit_preds})
for i in range(11):
    test_df[f'rf_oof_prob{str(i)}'] = rf_probs[:, i]
```

In [68]:

```
scores = cv_loop(oof_df, y, LinearRegression())
print(scores)
print(np.mean(scores))
```

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.

```
0.9379437431519608
0.9381737172630964
0.9377425770015045
0.9395930654619
0.9387499979561834

[0.9379437431519608, 0.9381737172630964, 0.9377425770015045, 0.9395930654619,
0.9387499979561834]
0.938440620166929
```

In [69]:

```
lr = LinearRegression()
lr.fit(oof_df, y)
```

Out[69]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [70]:

```
lr.coef_
```

Out[70]:

```
array([ 7.23245139e-01,  1.32816043e-01, -4.42761636e-04, -7.24394528e-01,
       -5.79622293e-01, -4.35792810e-01, -2.90513810e-01, -1.46104082e-01,
       -5.90173974e-04,  1.40557225e-01,  2.87431766e-01,  4.32618197e-01,
        5.79539631e-01,  7.36870877e-01])
```