Алгоритм Витерби без обучения на отдельных батчах

В данном ноутбуке рассмотрена реализация алгоритма Витерби без обучения на отдельных схожих сегментах данных.

В данном случае это будет нам полезно возможностью использования данного алгоритма при стекинге с бустингами и лесами.

В данном ноутбуке не так много оригинального кода. В основном использованы результаты отсюда: https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution (https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution)

И отсюда: https://www.kaggle.com/miklgr500/viterbi-algorithm-without-segmentation-on-groups (https://www.kaggle.com/miklgr500/viterbi-algorithm-without-segmentation-on-groups)

В данном случае пока достаточно понимания принципа работы данного алгоритма. Реализовывать его самостоятельно пока не имеет смысла в связи с тем, что есть ненулевая вероятность того, что данный алгоритм ничего не даст при стекинге, т.к. его качество (~ 0.93 - 0.934) недотягивает до качества стекингов, лесов и нейронных сетей, и может быть попросту незамечено метамоделью стекинга.

In [2]:
```
import os

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from tqdm.notebook import tqdm

from sklearn.metrics import f1_score, accuracy_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold, KFold


plt.style.use('dark_background')
```

Попробуем сначала использовать предобработанный датасет с более интелектуальной очисткой данных, рассмотренный в вышеперечисленных публичных ноутбуках.

In [7]:
```
train = pd.read_csv('clean-kalman/train_clean_kalman.csv')
test  = pd.read_csv('clean-kalman/test_clean_kalman.csv')
```

In [8]:

```
class ViterbiClassifier:
    def __init__(self, num_bins=1000):
        self._n_bins = num_bins
        self._p_trans = None
        self._p_signal = None
        self._signal_bins = None
        self._p_in = None

    def fit(self, x, y):
        self._p_trans = self.markov_p_trans(y)
        self._p_signal, self._signal_bins = self.markov_p_signal(true_state, x, self._n_bins)

        self._p_in = np.ones(len(self._p_trans)) / len(self._p_trans)
        return self

    def predict(self, x):
        x_dig = self.digitize_signal(x, self._signal_bins)
        return self.viterbi(self._p_trans, self._p_signal, self._p_in, x_dig)

    @classmethod
    def digitize_signal(cls, signal, signal_bins):
        # https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution
        signal_dig = np.digitize(signal, bins=signal_bins) - 1
        signal_dig = np.minimum(signal_dig, len(signal_bins) - 2)
        return signal_dig

    @classmethod
    def markov_p_signal(cls, state, signal, num_bins = 1000):
        # https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution
        states_range = np.arange(state.min(), state.max() + 1)
        signal_bins = np.linspace(signal.min(),
                                  signal.max(),
                                  num_bins + 1)
        p_signal = np.array([ np.histogram(signal[state == s],
                                           bins=signal_bins)[0] for s in states_range ])
        p_signal = np.array([ p / np.sum(p) if np.sum(p) != 0 else p for p in p_signal ])
        return p_signal, signal_bins

    @classmethod
    def markov_p_trans(cls, states):
        # https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution
```

```
        max_state = np.max(states)
        states_next = np.roll(states, -1)
        matrix = []
        for i in tqdm(range(max_state + 1)):
            current_row = np.histogram(states_next[states == i],
                                       bins=np.arange(max_state + 2))[0]
            if np.sum(current_row) == 0:
                current_row = np.ones(max_state + 1) / (max_state + 1)
            else:
                current_row = current_row / np.sum(current_row)
            matrix.append(current_row)
        return np.array(matrix)

    @classmethod
    def viterbi(cls, p_trans, p_signal, p_in, signal):
        # https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution
        offset = 10**(-20)

        p_trans_tlog  = np.transpose(np.log2(p_trans  + offset))
        p_signal_tlog = np.transpose(np.log2(p_signal + offset))
        p_in_log      =              np.log2(p_in     + offset)

        p_state_log = [ p_in_log + p_signal_tlog[signal[0]] ]

        for s in tqdm(signal[1:]):
            p_state_log.append(np.max(p_state_log[-1] + p_trans_tlog, axis=1)
                               + p_signal_tlog[s])

        states = np.argmax(p_state_log, axis=1)

        return states
```

In [9]:

```
true_state = train.open_channels.values
signal = train.signal.values
```

Обучение

In [10]:
```
viterbi = ViterbiClassifier().fit(signal, true_state)
train_prediction = viterbi.predict(signal)
```

```
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=4818797.0), HTML(value='')))
```

In [11]:
```
print("Accuracy =", accuracy_score(y_pred=train_prediction, y_true=true_state))
print("F1 macro =", f1_score(y_pred=train_prediction, y_true=true_state, average='macro'))
```

```
Accuracy = 0.9641219656852186
F1 macro = 0.9320404278331105
```

In [12]:

```
X_train = train.signal
y_train = train.open_channels

X_test = test.signal

n_fold = 5
folds = KFold(n_splits=n_fold, shuffle=True, random_state=17)

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

for training_index, validation_index in tqdm(folds.split(X_train), total=n_fold):
        # 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]

        true_state = y_train_.values
        signal = X_train_.values

        model = ViterbiClassifier().fit(signal, true_state)

        # скор на валидации
        preds = model.predict(X_valid.values)
        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
        preds = model.predict(X_test)
        prediction += preds

        print(f'score: {score}')

prediction /= n_fold
```

```
HBox(children=(FloatProgress(value=0.0, max=5.0), HTML(value='')))
```

```
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=963759.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))

score: 0.9293983141212808
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=963759.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))

score: 0.9294417188919251
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=963759.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))

score: 0.9298128243081841
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, max=963758.0), HTML(value='')))
```

```
HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))

score: 0.9299081397497022
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=963758.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=1999999.0), HTML(value='')))

score: 0.930220770278687
```

Алгоритм показал сравнительно высокий скор на валидации. Учитывая, что никаких дополнительных фичей здесь не используется, только непосредственное приближение восстановления скрытых состояний по видимым, это очень хороший результат.

Сохраним результат работы и oof предсказания для стекинга.

In [69]:
```python
def pred_proc(pred):
    pred = np.round(np.clip(pred, 0, 10))
    return pred.astype(int)
```

In [70]:
```python
y_catboost_pred = pred_proc(prediction)

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

In [71]:
```python
np.save('preds_viterbi_best', prediction)
np.save('oof_viterbi_best', oof)
```

Посмотрим теперь, какое качество алгоритм даёт на наших данных.

In [15]:
```
train = pd.read_csv('data-without-drift/train_clean.csv')
test  = pd.read_csv('data-without-drift/test_clean.csv')

true_state = train.open_channels.values
signal = train.signal.values

viterbi = ViterbiClassifier().fit(signal, true_state)
train_prediction = viterbi.predict(signal)
```

```
HBox(children=(FloatProgress(value=0.0, max=11.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=4999999.0), HTML(value='')))
```

In [16]:
```
print("Accuracy =", accuracy_score(y_pred=train_prediction, y_true=true_state))
print("F1 macro =", f1_score(y_pred=train_prediction, y_true=true_state, average='macro'))
```

```
Accuracy = 0.9117018
F1 macro = 0.8518252491783009
```

Качество даже на обучающей выборке совсем далеко от оптимального, поэтому будем использовать предыдущий датасет.

In [ ]:
```

```