Алгоритм Витерби для скрытого марковского процесса.

Не секрет, что данные в соревновании получены искусственно.

Утверждается, они были сгенерированы с помощью некоего марковского процесса, затем к ним был добавлен шум (сгенерированный, потом прогнанный через физический усилитель и записанный на микрофон), затем сдвиг (drift).

Следовательно, возникает задача получения скрытых состояний(количества открытых ионных каналов) цепи по видимым(сигналу).

В данном ноутбуке проведён эксперимент с использованием алгоритма Витерби поиска наиболее подходящего списка состояний.

https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%
(https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%

In [3]:

```
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

from tqdm.notebook import tqdm

from sklearn.metrics import f1_score

from scipy.stats import norm
```

In [4]:

```
train = pd.read_csv('data-without-drift/train_clean.csv')
test  = pd.read_csv('data-without-drift/test_clean.csv')
```

Прежде чем реализовывать алгоритм самостоятельно, посмотрим, осмысленно ли это делать вообще.

Получим предсказания с использованием kfold кросс-валидации, и посмотрим на качество на валидации.

Реализацию алгоритма на данном этапе возьмем из паблик ноутбуков соревнования.

https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution
(https://www.kaggle.com/friedchips/the-viterbi-algorithm-a-complete-solution)

In [5]:

```
class ViterbiClassifier:
    def __init__(self):
        self._p_trans = None
        self._p_signal = None
        self._p_in = None

    def fit(self, x, y):
        self._p_trans = self.markov_p_trans(y)
        self._dists = []
        self._states = len(np.unique(y))
        for s in np.arange(y.min(), y.max() + 1):
            self._dists.append((np.mean(x[y == s]), np.std(x[y == s])))

        return self

    def predict(self, x):
        p_signal = self.markov_p_signal(x)
        return self.viterbi(self._p_trans, p_signal, x)

    def markov_p_signal(self, signal):
        p_signal = np.zeros((self._states, len(signal)))
        for k, dist in enumerate(self._dists):
            p_signal[k, :] = norm.pdf(signal, *dist)

        return p_signal

    def markov_p_trans(self, states):
        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.arang
            if np.sum(current_row) == 0:
                current_row = np.ones(max_state + 1) / (max_state + 1) # ...us
            else:
                current_row = current_row / np.sum(current_row) # normalize to
            matrix.append(current_row)
        return np.array(matrix)

    def viterbi(self, p_trans, p_signal, signal):
        offset = 10**(-20)

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

        T1 = np.zeros(p_signal.shape)
        T2 = np.zeros(p_signal.shape)

        T1[:, 0] = p_signal_tlog[0, :]
        T2[:, 0] = 0

        for j in tqdm(range(1, p_signal.shape[1])):
            for i in range(len(p_trans)):
                T1[i, j] = np.max(T1[:, j - 1] + p_trans_tlog[:, i] + p_signal
                T2[i, j] = np.argmax(T1[:, j - 1] + p_trans_tlog[:, i] + p_sig

        x = np.empty(p_signal.shape[1], 'B')
        x[-1] = np.argmax(T1[:, p_signal.shape[1] - 1])
        for i in reversed(range(1, p_signal.shape[1])):
```

```
            x[i - 1] = T2[x[i], i]

        return x
```

Проведём кросс-вадидацию и получим out of fold предскзаания для данного алгоритма.

In [7]:

```
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_fol
        # 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]

        model = ViterbiClassifier().fit(X_train_, y_train)


        # скор на валидации
        preds = model.predict(X_valid)
        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
```

40% 2/5 [27:32<34:19, 686.34s/it]

100% 11/11 [00:02<00:00, 3.78it/s]

100% 999999/999999 [11:23<00:00, 1463.31it/s]

100% 1999999/1999999 [07:30<00:00, 4443.66it/s]

score: 0.7533647190608334

100% 11/11 [00:03<00:00, 3.09it/s]

```
100%                                    999999/999999 [03:45<00:00, 4434.41it/s]


100%                                    1999999/1999999 [11:34<00:00, 2877.98it/s]


score: 0.7530597098194463

100%                                    11/11 [03:53<00:00, 21.26s/it]


100%                                    999999/999999 [03:47<00:00, 4390.93it/s]


9%                                      184787/1999999 [00:52<06:54, 4380.43it/s]

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
 last)
<ipython-input-7-9ad60175c2dd> in <module>
     30
     31         # предсказание на тесте
---> 32         preds = model.predict(X_test)
     33         prediction += preds
     34

<ipython-input-5-245828350e11> in predict(self, x)
     16     def predict(self, x):
     17         p_signal = self.markov_p_signal(x)
---> 18         return self.viterbi(self._p_trans, p_signal, x)
     19
     20     def markov_p_signal(self, signal):

<ipython-input-5-245828350e11> in viterbi(self, p_trans, p_signal, sig
nal)
     55             for i in range(len(p_trans)):
     56                 T1[i, j] = np.max(T1[:, j - 1] + p_trans_tlog[
:, i] + p_signal_tlog[j, i])
---> 57                 T2[i, j] = np.argmax(T1[:, j - 1] + p_trans_tl
og[:, i] + p_signal_tlog[j, i])
     58
     59         x = np.empty(p_signal.shape[1], 'B')

KeyboardInterrupt:
```

Можно видеть, что качество на валидации очень низкое, поэтому не стоит ожидать высокого качества на LB, и использовать в стекинге, т.к. модель верхнего уровня в стекинге/блендинге, скорее всего, просто возьмёт ответы данной модели с коэффициентом 0.

В паблик ноутбуках с помощью этого алгоритма получают достаточно высокое качество, т.к. используют его для схожих батчей на тестовой и обучающей выборке (предполагается, что если значения сигналов в различных батчах схожи, то для генерации данных в них используются схожие макровские процессы).

Поэтому, использовать данную модель в стекинге не представляется возможным.

Стоит отметить, что в результате работы алгоритма на выходе мы получаем матрицу вероятностей переходов. Её можно использовать в качестве новой фичи. Также, данная модель будет полезна при обучении остальных рассмеотренных моделей на отдельных батчах.

In [ ]:

```
1
```