Формирование обучающей выборки для predictive clustering

In [1]:
```python
import json
import numpy as np
import pandas as pd
import os
import time
import glob
import datetime
import itertools
from tqdm import tqdm
from datetime import datetime
import matplotlib.pyplot as plt
from itertools import product
from scipy.special import gamma
from sklearn.neighbors import KDTree
from scipy.spatial import cKDTree
from collections import defaultdict
```

Формирование обучающей выборки для кластеризации по Уишарту

In [4]:
```python
new_train_tot3 = np.load('new_train_tot3.npy')
print(new_train_tot3.shape)
new_train_tot3 # список наблюдений для обучающей выборки
```

```
(3912, 5)
```

Out[4]:
```
array([['SPI', '2020-09-23', '720', '34', '1'],
       ['GLSI', '2020-12-09', '666', '54', '1'],
       ['AACG', '2021-02-04', '627', '55', '1'],
       ...,
       ['EVH', '2020-08-20', '486', '28', '0'],
       ['MXIM', '2019-10-29', '488', '30', '0'],
       ['PEI', '2020-10-08', '519', '28', '0']], dtype='<U11')
```

Нельзя, чтобы обучающие выборки по каждому наблюдению слиплись, иначе шаблоны пройдутся не только по своим наблюдениям но и по чужим, выходя за границы, поэтому будем резать и сохранять в файлы (опер памяти, чтобы все это делать сразу не хватит).

In [5]:
```python
from itertools import product
patterns1=np.array(list(product(np.arange(1, 11, 1),repeat=4)),dtype=np.uint8)
np.random.seed(123)
# возьмем 1000 шаблонов, слишком долго будет вычисляться, если возьмем все 10 тыс.
patterns = patterns1[np.random.choice(patterns1.shape[0], 1000, replace=False)]
patterns
```

Out[5]:
```
array([[ 3,  7,  6,  7],
       [ 1,  5,  5,  6],
       [10,  6,  1,  6],
       ...,
       [ 3,  7,  5,  8],
       [ 1,  5,  4, 10],
       [ 9,  5,  1,  5]], dtype=uint8)
```

In [11]:
```python
#np.save('patterns_1000_patterns_for_wishart.npy', patterns)
```

In [7]:
```python
# отдельные суммы расстояний для каждого шаблона
q = np.sum(patterns[:,:0],axis=1).reshape((patterns.shape[0],1))
a = np.sum(patterns[:,:1],axis=1).reshape((patterns.shape[0],1))
b = np.sum(patterns[:,:2],axis=1).reshape((patterns.shape[0],1))
c = np.sum(patterns[:,:3],axis=1).reshape((patterns.shape[0],1))
d = np.sum(patterns[:,:4],axis=1).reshape((patterns.shape[0],1))
```

In [8]:
```python
np.max(d) # макс обрезка по шаблону с конца ряда
```

Out[8]:
```
39
```

In [10]:
```python
# обучающая выборка для каждого дневного наблюдения из списка класса

min_ts = -0.9274193548387099 # для сдвига всей выборки вверх, отриц значения здесь недопустимы

for i in tqdm(range(len(new_train_tot3))):

    # откроем каждое наблюдение из обуч выборки
    ticker_name = new_train_tot3[i][0]
    day = new_train_tot3[i][1]
    parent_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/train3'
    file_min = os.path.join(parent_dir, ticker_name + '_'+ day +'.csv')
    train_data = pd.read_csv(file_min).to_numpy().ravel()

    # сделаем ряд стационарным через np.diff, очень важно ее сделать
    ts_diff = np.diff(train_data)
    ts_diff = ts_diff - np.full((ts_diff.shape[0], ), min_ts) # но также важно сдвинуть все ряды вверх

    # запишем наблюдения по каждому паттерну в тензор и правильно переформатируем (спец трюк)
    sds=[]
    for i in range(len(ts_diff)-np.max(d)):
        sd=np.hstack((ts_diff[q+i], ts_diff[a+i], ts_diff[b+i], ts_diff[c+i], ts_diff[d+i]))
        sds.append(sd)
    sds=np.array(sds)
    sds=np.stack(sds,1)

    # запишем эти тензоры в файлы
    parent_dir2='C:/Users/Kuanysh/Downloads/pump_and_dump/train_npys'
    np_dir = os.path.join(parent_dir2, ticker_name + '_'+ day +'.npy')
    with open(np_dir, 'wb') as f:
         np.save(f, sds)
```

```
100%|██████████████████████████████████████████████████| 3912/3912 [02:52<00:00, 2
2.65it/s]
```

In [11]:
```python
# создадим подпапки для кадого из 1000 шаблонов, чтобы туда собирать срезанные по каждому шаблону обуч-ие выбор
for n in range(1000): # 1000 шаблонов
    parent_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/1000patterns'
    path = os.path.join(parent_dir, str(n))
    os.makedirs(path)
```

In [12]:
```python
for i in tqdm(range(len(new_train_tot3))):

    # вскроем обуч выборки по каждому наблюдению
    ticker_name = new_train_tot3[i][0]
    day = new_train_tot3[i][1]
    parent_dir2='C:/Users/Kuanysh/Downloads/pump_and_dump/train_npys'
    np_dir = os.path.join(parent_dir2, ticker_name + '_'+ day +'.npy')
    train_ts = np.load(np_dir, mmap_mode='r+')
    sh = train_ts.shape[0] # 1000 шаблонов

    # теперь поместим  срезанные по каждому шаблону обуч-ие выборки в их подпапки
    parent_dir3 = 'C:/Users/Kuanysh/Downloads/pump_and_dump/1000patterns'
    for k in range(sh):
        path1 = os.path.join(parent_dir3, str(k))
        path2 = os.path.join(path1, ticker_name + '_' + day + '_' + str(k) +'.npy')
        with open(path2, 'wb') as f:
            np.save(f, train_ts[k])
```

```
100%|██████████████████████████████████████████████████| 3912/3912 [28:59<00:00,
2.25it/s]
```

In [13]:
```python
for i in tqdm(range(1000)): # 1000 шаблонов

    # вскроем каждую подпаку и соберем все обучающие выборки и объеденим наблюдения по каждому шаблону
    parent_dir = 'C:/Users/Kuanysh/Downloads/pump_and_dump/1000patterns'
    path = os.path.join(parent_dir, str(i))
    files = glob.glob(path + '/*') # обнаружим все файлы
    list_pats = []
    for file in files:
        one_train_pat = np.load(file, mmap_mode='r+').tolist()
        list_pats.extend(one_train_pat) # объединение по каждому шаблону

    # объединенные наблюдения поместим в отдельную папку
    directory = 'C:/Users/Kuanysh/Downloads/pump_and_dump/total_patterns'
    with open(os.path.join(directory, 'tot'+ str(i)), "w") as fp:
         json.dump(list_pats, fp)
```

```
100%|█████████████████████████████████████████████████| 1000/1000 [4:21:50<00:00, 1
5.71s/it]
```

In [2]:
```python
np.random.seed(12345)
choice = np.random.choice(1473014, 400000, replace=False)
#np.save('choice_500000.npy', choice)
choice
```

Out[2]:
```
array([ 727536,  633366,   16414, ...,  940512, 1279791,  252998])
```

In [3]:
```python
# теперь наконецто объединим все шаблоны воедино, и таким образом, вся обучающая выборка будет правильно сформи
list_tot = []
for i in tqdm(range(1000)): # 1000 шаблонов
    parent_dir = "C:/Users/Kuanysh/Downloads/pump_and_dump/total_patterns"
    path = os.path.join(parent_dir, 'tot' + str(i))
    with open(path, "r") as fp:
         tot = json.load(fp)
    tot_ = np.array(tot)
    rnd_tot = tot_[choice]
    list_tot.append(rnd_tot)
```

```
100%|█████████████████████████████████████████████████| 1000/1000 [1:11:52<00:00,
4.31s/it]
```

In [4]:
```python
list_tot_=np.array(list_tot, dtype=np.float32)
#np.save('tot_all.npy', list_tot_)
```

In [2]:
```python
list_tot_all = np.load("tot_all.npy", mmap_mode='r+')
```

Кластеризация по Уишарту:

In [3]:
```python
# кластеризуем обучающую выборку по Уишарту:
class Wishart:
    def __init__(self, wishart_neighbors, significance_level):
        self.wishart_neighbors = wishart_neighbors  # Number of neighbors
        self.significance_level = significance_level  # Significance level

    def fit(self, X):
        #kdt = KDTree(X, metric='euclidean')
        kdt = cKDTree(X, leafsize=40)
        #add one because you are your neighb.

        distances, neighbors = kdt.query(X, k = self.wishart_neighbors + 1, p=2, n_jobs=-1)
        #distances, neighbors = kdt.query(X, k = self.wishart_neighbors + 1, return_distance = True)
        neighbors = neighbors[:, 1:]

        distances = distances[:, -1]
        indexes = np.argsort(distances)

        size, dim = X.shape

        self.object_labels = np.zeros(size, dtype = int) - 1

        #index in tuple
        #min_dist, max_dist, flag_to_significant
        self.clusters = np.array([(1., 1., 0)])
        self.clusters_to_objects = defaultdict(list)
        #print('Start clustering')

        for index in indexes:
            neighbors_clusters =\
                np.concatenate([self.object_labels[neighbors[index]], self.object_labels[neighbors[index]]])
            unique_clusters = np.unique(neighbors_clusters).astype(int)
            unique_clusters = unique_clusters[unique_clusters != -1]


            if len(unique_clusters) == 0:
                self._create_new_cluster(index, distances[index])
            else:
                max_cluster = unique_clusters[-1]
                min_cluster = unique_clusters[0]
                if max_cluster == min_cluster:
                    if self.clusters[max_cluster][-1] < 0.5:
                        self._add_elem_to_exist_cluster(index, distances[index], max_cluster)
                    else:
                        self._add_elem_to_noise(index)
                else:
                    my_clusters = self.clusters[unique_clusters]
                    flags = my_clusters[:, -1]
                    if np.min(flags) > 0.5:
                        self._add_elem_to_noise(index)
                    else:
                        significan = np.power(my_clusters[:, 0], -dim) - np.power(my_clusters[:, 1], -dim)
                        significan *= self.wishart_neighbors
                        significan /= size
                        significan /= np.power(np.pi, dim / 2)
                        significan *= gamma(dim / 2 + 1)
                        significan_index = significan >= self.significance_level

                        significan_clusters = unique_clusters[significan_index]
                        not_significan_clusters = unique_clusters[~significan_index]
                        significan_clusters_count = len(significan_clusters)
                        if significan_clusters_count > 1 or min_cluster == 0:
                            self._add_elem_to_noise(index)
                            self.clusters[significan_clusters, -1] = 1
                            for not_sig_cluster in not_significan_clusters:
                                if not_sig_cluster == 0:
                                    continue

                                for bad_index in self.clusters_to_objects[not_sig_cluster]:
                                    self._add_elem_to_noise(bad_index)
                                self.clusters_to_objects[not_sig_cluster].clear()
                        else:
                            for cur_cluster in unique_clusters:
                                if cur_cluster == min_cluster:
                                    continue

                                for bad_index in self.clusters_to_objects[cur_cluster]:
                                    self._add_elem_to_exist_cluster(bad_index, distances[bad_index], min_cluste
                                self.clusters_to_objects[cur_cluster].clear()

                            self._add_elem_to_exist_cluster(index, distances[index], min_cluster)

        return self.clean_data()

    def clean_data(self):
        unique = np.unique(self.object_labels)
        index = np.argsort(unique)
        if unique[0] != 0:
            index += 1
        true_cluster = {unq :  index for unq, index in zip(unique, index)}
        result = np.zeros(len(self.object_labels), dtype = int)
        for index, unq in enumerate(self.object_labels):
            result[index] = true_cluster[unq]
        return result

    def _add_elem_to_noise(self, index):
        self.object_labels[index] = 0
        self.clusters_to_objects[0].append(index)

    def _create_new_cluster(self, index, dist):
        self.object_labels[index] = len(self.clusters)
        self.clusters_to_objects[len(self.clusters)].append(index)
        self.clusters = np.append(self.clusters, [(dist, dist, 0)], axis = 0)

    def _add_elem_to_exist_cluster(self, index, dist, cluster_label):
        self.object_labels[index] = cluster_label
        self.clusters_to_objects[cluster_label].append(index)
        self.clusters[cluster_label][0] = min(self.clusters[cluster_label][0], dist)
        self.clusters[cluster_label][1] = max(self.clusters[cluster_label][1], dist)
```

In [6]:
```python
claster_labs_Wish2 = [Wishart(22, 0.2).fit(list_tot_all[i]) for i in tqdm(range(len(patterns)))]
```

```
  0%|                                                              | 0/1000 [00:00
<?, ?it/s]<ipython-input-3-59bae2c59e79>:12: DeprecationWarning: The n_jobs argument has been renamed "worker
s". The old name "n_jobs" will stop working in SciPy 1.8.0.
  distances, neighbors = kdt.query(X, k = self.wishart_neighbors + 1, p=2, n_jobs=-1)
<ipython-input-3-59bae2c59e79>:52: RuntimeWarning: divide by zero encountered in power
  significan = np.power(my_clusters[:, 0], -dim) - np.power(my_clusters[:, 1], -dim)
<ipython-input-3-59bae2c59e79>:52: RuntimeWarning: invalid value encountered in subtract
  significan = np.power(my_clusters[:, 0], -dim) - np.power(my_clusters[:, 1], -dim)
100%|█████████████████████████████████████████████████| 1000/1000 [2:19:03<00:00,
8.34s/it]
```

In [10]:
```python
np.save('claster_labs_Wish2.npy', claster_labs_Wish2)
```

In [8]:
```python
# получим центры кластеров
sds = list_tot_all
sds_Wishart = []
for i in range(len(patterns)):
    sds_i_W = []
    for cl in set(claster_labs_Wish2[i]):
        if cl != 0:
            sds_i_W.append(np.mean(sds[i][claster_labs_Wish2[i] == cl], axis=0))
    sds_i_W = np.array(sds_i_W)
    sds_Wishart.append(sds_i_W)
```

In [11]:
```python
# выровняем ряды по одной длине и заполним ненулевыми значениями
from tslearn.utils import to_time_series_dataset
N = np.array([0.6666], dtype=np.float32)
train_pat = to_time_series_dataset(sds_Wishart)
train_pat[np.isnan(train_pat)]=N[0]
train_pat.shape
```

Out[11]:
```
(1000, 2441, 5)
```

In [16]:
```python
train_pat[200][200:500]
```

Out[16]:
```
array([[0.88575757, 0.9274323 , 0.92741054, 0.92741394, 0.92739296],
       [0.92738473, 0.93124068, 0.93088257, 0.92741936, 0.92741936],
       [0.92213601, 0.92744416, 0.92741948, 0.92741531, 0.93233728],
       ...,
       [0.99365377, 0.91254997, 0.92790794, 0.92684883, 0.92741936],
       [0.92740697, 0.92776537, 0.92739129, 1.03847671, 0.92732012],
       [0.89711171, 0.98803461, 0.92741936, 0.92741936, 0.92726785]])
```

In [17]:
```python
np.save('train_pat_1000_wishart_totlist3.npy', train_pat)
```

In [ ]:
```python

```