Соревнование по курсу «Байесовский выбор моделей»

Общая информация

- Участие в соревновании по командам. В каждой команде от 3 до 5 человек;
- Каждый участник победившей команды получает 150 баллов, баллы последующих участвовавших команд убывают до 20 от 100 баллов для второй;
- Второй этап соревнования состоит в построении автономного алгоритма анализа временных рядов для детектирования разладки;
- Код разработанного алгоритма, соответствующего заданному интерфейсу, требуется отправить на почту aduenko1@gmail.com и iakovlev.kd@phystech.edu до 19 мая 23:59 по Москве (можно в виде ссылки на репозиторий на github);
- Алгоритм может содержать ветвление для проверки некоторых свойств входного временного ряда и преобразования временного ряда для лучшей работы;
- Предоставленный код будет тестироваться на разных выборках, похожих на 4 выборки, предоставленных на первом этапе соревнования;
- Вместе с алгоритмом должен быть предоставлен отчет его тестирования на обучающих выборках, предоставленных ранее, и сравнение с истинными моментами разладки. Плюсом будет тестирование на собственных выборках с известной разладкой или реальных данных.

Описание процедуры соревнования

Прогноз временного ряда. Имеется временной ряд $\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{x}_i \in \mathbb{R}^D$, сгенерированный по некоторому правилу. При этом в этом правиле могла произойти разладка в некоторый момент времени T.
Задача: Найти момент разладки $100 \leq T \leq N - 100$ или указать, что разладки не было $T = N + 1$.
Условие победы: Побеждает та команда, прогноз $\hat{T}$ которой окажется наиближайшим к истинному моменту разладки T.
Описание игры:

- Перед началом игры у каждой команды есть $S_0 = 1000000$ конфет (в каждом из наборов временных рядов);
- Игра состоит из поочередного анализа K временных рядов, для каждого из которых требуется построить прогноз $\hat{T}_k$;
- Каждая команда должна выбрать размер вклада D_k для каждого временного ряда, $\sum_k D_k \leq S$;
- Вклад победившей команды удваивается ($S_k = S_{k-1} + D_k$), проигравшая – теряет все конфеты ($S_k = S_{k-1} - D_k$). Если команд Z больше 2, то команды, занявшие места z со второго до предпоследнего, получают $S_k = S_{k-1} + D_k(1 - 2\frac{z-1}{Z-1})$ конфет;
- Переход на шаг $k + 1$.

Замечание 1: Требуется предоставить алгоритм выбора D_k, T_k для временных рядов из контрольной выборки;
Замечание 2: Алгоритм должен быть выполнен на языке Python в виде класса с заданным интерфейсом и предобучен по данным из обучающей выборки (см. ниже);
Замечание 3: Время работы алгоритма на всей контрольной выборке (равномощна обучающей) - не более 30 минут на среднем ноутбуке.

Интерфейс алгоритма

Ниже приведен базовый класс для анализа и поиска разладки в наборе временных рядов, предоставленных в `pd.DataFrame data_df` при полном бюджете на всю выборку S_0 конфет. Ваш класс для работы с рядами должен быть отнаследован от данного и должен содержать метод predict, который и будет вызываться.

```python
class DummyDetector(object):
    def __init__(self, S0: float, data_df: pd.DataFrame):
        self.S0 = S0
        self.data_df = data_df

    @property
    def T(self) -> int:
        return len(self.data_df)

    @property
    def feature_names(self):
        return self.data_df.columns

    @property
    def num_series(self) -> int:
        return np.array([int(item.split("_")[1])
        for item in self.feature_names]).max() + 1

    def get_data_by_series_index(self, index) -> np.ndarray:
        assert (index >= 0) and (index < self.num_series)
        cur_feature_names = [item for item in self.feature_names
        if item.startswith(f"TS_{index}_coord_")]
        return data_df[cur_feature_names].values

    def analyze_series(self, index) -> Tuple[int, float]:
        '''Returns the breakpoint location and confidence score'''
        data = self.get_data_by_series_index(index)
        ## Some analysis here, in the dummy one - none
        return int(self.T / 2), 0.5

    def predict(self) -> Tuple[np.ndarray, np.ndarray]:
        predicted_breakpoint_list = []
        confidence_list = []
        for index in range(self.num_series):
            cur_breakpoint, cur_confidence = self.analyze_series(index)
```

```
            predicted_breakpoint_list.append(cur_breakpoint)
            confidence_list.append(cur_confidence)
        confidence_list = np.array(confidence_list)
        return np.array(predicted_breakpoint_list),\
        self.S0 * confidence_list / np.sum(confidence_list)
```