Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б10-мм

Разработка ядра пакета PySATL-CPD

Романюк Артем Дмитриевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н. Гориховский В. И.

Консультант:
инженер-исследователь лаборатории «YADRO» Кутуев В.А.

Санкт-Петербург
2025

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор инструментов обнаружения разладки **5**

2.1. Необходимые сведения 5

2.2. Обзор алгоритмов 5

2.3. Обзор библиотек 7

2.4. Выводы 8

3. Проектирование **9**

3.1. DataController 9

3.2. Scrubber 9

3.3. ScrubberScenario 10

3.4. Algorithm 10

3.5. CPDCore 10

4. Реализация **12**

4.1. Реализация ядра 12

4.2. Генератор данных 13

Заключение **18**

Список литературы **19**

Введение

Математическая статистика является важной областью математики, создающей методы и инструменты для анализа данных и обработки информации. Сегодня, когда данные играют ключевую роль практически во всех сферах деятельности, методы математической статистики находят широкое применение в самых разных областях: от экономики и медицины до инженерии и социальных наук.

Одним из таких применений является нахождение точек разладки (change point detection). В таких точках поведение системы резко изменяется. Один из примеров использования — изучение изменений климата[3]. Есть множество методов нахождения разладки, например графовый и Байесовский методы. Каждый из них лучше работает в разных условиях и требует разные параметры для запуска, при этом решая одну задачу.

В рамках модуля PySATL-CPD [1] уже были реализованны графовый и Байесовский методы на языке программирования Python. Авторами проекта было принято решение разделить данный модуль на ядро, занимающееся обнаружением разладки и экспериментальное окружение, для изучения эффективности и сравнения алгоритмов.

В данной работе ставится задача реализовать ядро библиотеки для работы с различными методами определения точек разладки, обеспечив возможность интеграции новых методов в будущем.

[1] https://github.com/PySATL/PySATL-CPD-Module. Дата обращения: 17 ноября 2024 г.

1. Постановка задачи

Целью работы является проектирование и создание ядра для модуля определения разладки. Для её выполнения были поставлены следующие задачи:

1. выполнить обзор существующих инструментов обнаружения разладки;
2. сформулировать требования к архитектуре разрабатываемого модуля;
3. разработать архитектуру;
4. реализовать данный компонент в проекте PySATL-CPD-Module [2].

[2] https://github.com/PySATL/PySATL-CPD-Module. Дата обращения: 17 ноября 2024 г.

2. Обзор инструментов обнаружения разладки

Будут рассмотрены различные подходы для обнаружения разладки и существующие библиотеки, позволяющие искать разладки, для выявления функциональных требований разрабатываемой библиотеки. Ядро модуля разрабатывается с учетом использования нескольких методов, поэтому будут рассмотрены только библиотеки, содержащие несколько алгоритмов и позволяющие запускать их единообразным образом.

2.1. Необходимые сведения

Алгоритмы обнаружения разладки могут выполнять две задачи: детекция (обнаружение наличия разладки) и локализация (обнаружение точки разладки). Задачи обнаружения разладки делятся на два типа:

- Offline.
  - Все значения временного ряда известны;
- Online.
  - Известны значения временного ряда только до текущего момента времени;
  - Необходимо быстро обрабатывать поступающие значения.

2.2. Обзор алгоритмов

1. **Байесовский метод [1]**
   Главная идея алгоритма заключается в расчете вероятности нахождения разладки. Если эта вероятность больше заданного порогового значения, то считаем разладку найденой. Так же рассчитывает «длину пробега» (число шагов, пройденных с последней разладки) для определения точного места разладки. Алгоритм

работает в онлайн режиме (постепенно получая данные обрабатывать их). Определение наличия разладки и ее локализация занимают одинаковое время. Плохо работает на большом временном ряду.

2. **Графовый метод [2]**
   Задается функция, определяющая похожие значения (например разница в значениях меньше 5) и строится граф, где вершины – значения временного ряда, а ребром соединены похожие вершины. Проходясь по временной оси, в каждой точке считаются значения, зависящие от количества проходящих через нее ребер, и сравниваются с пороговым значением. Работает только в офлайн режиме. Определение наличия разладки и ее локализация занимают одинаковое время. Занимает много памяти при работе с большим временным рядом.

3. **Плотностные методы**
   RULSIF [5] и KLIEP [4] – примеры плотностных методов. Основная идея заключается в оценке отношений плотностей распределений данных до и после предполагаемой точки разладки с целью выявления изменений в этих распределениях. Могут работать в онлайн режиме. Это достигается использованием «скользящего окна». Определяется окно (часть данных) и учитываются только данные, входящие в него, остальные забываются. По мере добавления новых данных окно передвигается. Определение наличия разладки и ее локализация занимают одинаковое время.

Идею «скользящего окна» плотностных алгоритмов можно обобщить для всех алгоритмов, работающих только в офлайн режиме, для решения онлайн задач.

Scrubber – программный компонент, отвечающий за разбиение входных данных на куски определенным способом с целью дальнейшей передачи их в алгоритм поиска разладки. Позволяет использовать Offline алгоритм в Online задачах и сокращать время выполнения Offline алгоритма при работе с большим объемом данных.

2.3. Обзор библиотек

1. **Ruptures**

   Python библиотека для анализа временных рядов. Нет встроенного скраббера для деления данных на окна для будущей обработки алгоритмом. Поддерживает только работу в офлайн режиме. Есть генератора данных для тестирования параметров алгоритмов. Генерация происходит в коде и нет способа сохранить данные для будущей обработки. Нет возможности выполнять нахождение разладки без локализации. GitHub: https://github.com/deepcharles/ruptures/ [3].

2. **Kats**

   Python библиотека для анализа временных рядов. Нет встроенного скраббера, но можно вручную выбирать окно. Поддерживает работу только в офлайн режиме. Так же нет генератора данных для тестирования параметров алгоритмов. Можно выбирать разные сценарии нахождения точек разладки. Нет возможности выполнять нахождение разладки без локализации. GitHub: https://github.com/facebookresearch/Kats/ [4].

Составлена сравнительная таблица.

Критерии сравнения	ruptures	Kats
Режим работы	Оффлайн	Оффлайн
Генератор данных	Да	Нет
Сохранение сгенерированных данных	Нет	Нет
Поддержка скраббирования	Нет	Нет
Ограничение точек разладки	Нет	Да
Нахождение разладки без локализации	Нет	Нет

[3] Дата обращения: 17 ноября 2024 г.

[4] Дата обращения: 17 ноября 2024 г.

2.4. Выводы

Исходя из обзора методов обнаружения разладки, можно сделать вывод, что не так много подходов имеют возможность решения онлайн задач. Для чего им нужен скраббер. Так же скраббер даст возможность работать с меньшим объемом данных. Время детекции не всегда совпадает с временем локализации, следовательно, стоит выделить их как два разных режима.

Исходя из обзора библиотек: Для проверки качества работы алгоритмов, важен генератор данных, для быстрого создания датасетов. При плохой работе алгоритма на части данных могут возникать случайные разладки. Для их минимизации нужно ограничение разладок на окне скраббера.

Требования к ядру модуля:

1. два режима работы алгоритма: нахождение и локализация точек разладки;
2. наличие скраббера;
3. возможность ограничения количества точек разладки на окне скраббера.

3. Проектирование

В данном разделе будут описаны все классы архитектуры ядра и причины их создания.

3.1. DataController

Так как программа может работать с большими объемами данных, требуется класс для их управления. Главной задачей DataController является инкапсуляция работы с данными. Выделение отдельного класса для этой задачи упрощает доступ к данным и в дальнейшем позволяет внедрять различные способы их передачи ядру, например, ленивое чтение больших файлов.

3.2. Scrubber

У многих алгоритмов возникают проблемы при работе с большими объемами данных. Например, графовый алгоритм будет использовать много памяти. Тогда нужно разделить данные на окна (меньшие части) и уже окна передавать в алгоритм.

Также необходимо сопоставлять точки разладки, обнаруженные на отдельном окне, с общими данными. Это важно для корректного позиционирования, так как разладка, найденная в пределах окна, имеет относительные координаты, которые нужно преобразовать в абсолютные координаты исходного набора данных.

Важно поддерживать возможность перехода в онлайн-режим. Это достигается возможностью постепенно добавлять данные в скраббер и когда новых данных будет достаточно для нового окна, передать алгоритму.

Есть разные виды скрабберов (например лнейный или с логарифмическим забыванием) следовательно стоит выделить абстрактный класс скраббера.

3.3. ScrubberScenario

Алгоритмы и их параметры сильно зависят от данных. При неудачном подборе параметров, на промежутке может быть определено много ложно положительных точек. Что бы не засорять результат работы алгоритма очевидно неверными данными стоит дать возможность пользователю ставить ограничение на максимальное количество точек на одном окне скраббера. Потребность локализации так же будет фиксировать ScrubberScenario.

3.4. Algorithm

Алгоритм — основная часть ядра. У каждого алгоритма есть свои детали реализации, из-за чего например время определения наличия разладки может занимать меньше времени, чем нахождение точных точек. Следовательно, в абстрактный класс алгоритма стоит вынести два метода: локализация и просто нахождение наличия разладки.

3.5. CPDCore

Все выше перечисленные компоненты необходимо объединить и запускать в правильном порядке. В роли оркестратора будет выступать CPDCore, реализующий паттерн Посредник. У CPDCore есть один метод run, который будет возвращать найденые точки разладки. Сначала данные делятся на части с помощью DataController. Далее части данных передаются в скраббер, который делит их на окна и передает в алгоритм. Алгоритм возвращает разладки в скраббер, скраббер сопоставляет их с данными в окне и передает в DataController. DataController так же сопоставляет данные и возвращает результат работы. Диаграмма классов ядра представлена на рис. 1.

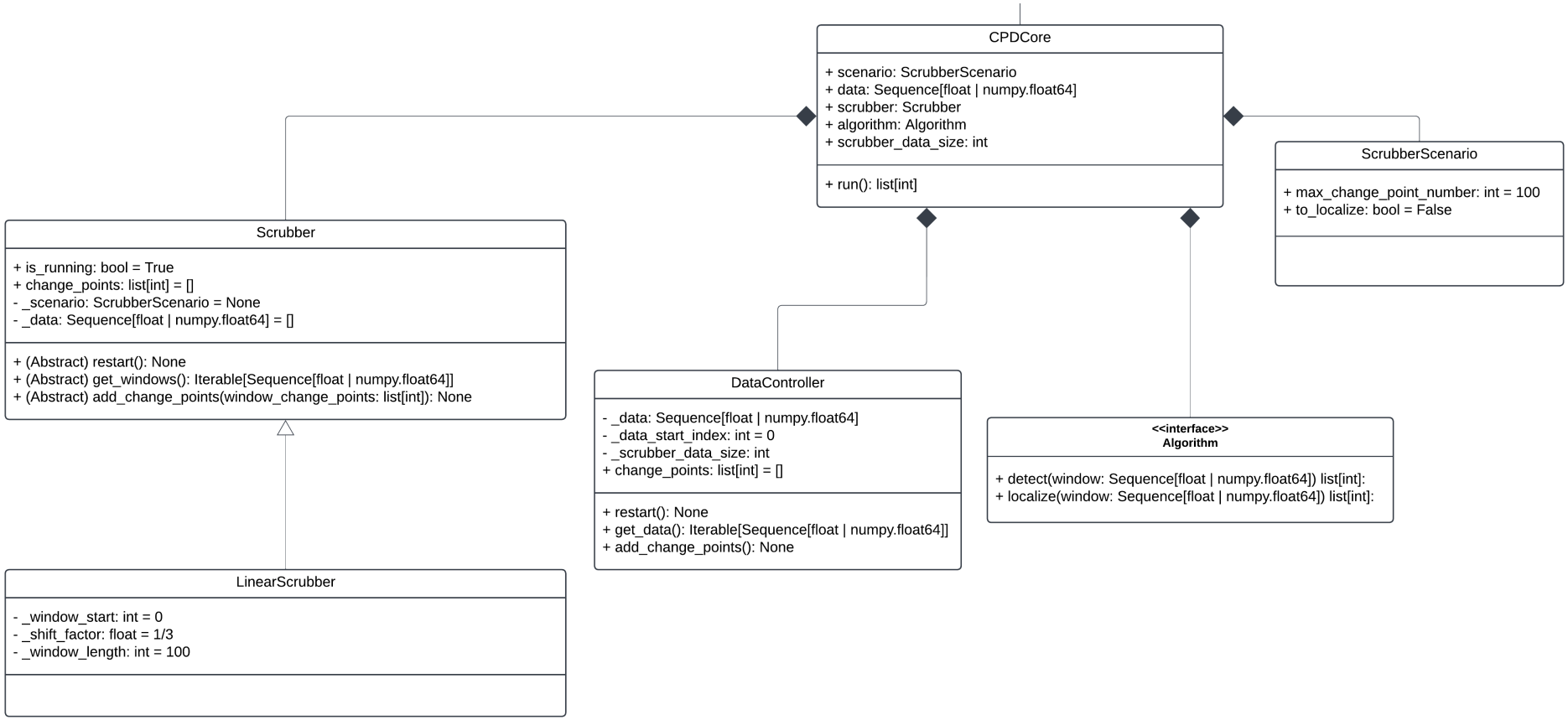

Рис. 1: Диаграмма классов ядра.

4. Реализация

4.1. Реализация ядра

Для представления данных в модуле было выбрано использовать тип данных Sequence из библиотеки collections. Он предоставляет доступ к количеству данных и элементам по индексу. Это дает возможность делить данные, основываясь на их количестве. В будущем при расширении до онлайн режима планируется использовать MutableSequence для возможности обновления данных.

1. **Scrubber**
   Абстрактный класс Scrubber имеет методы для приведения скраббера к изначальному состоянию, генерации окон и сохранения точек разладки. При изменении данных в скраббере он возвращается к изначальному состоянию. Был реализован класс LinearScrubber. Задается длина окна и коэффициент сдвига, отвечающий за то, на сколько будет сдвигаться окно. При неудачном делении данных разладка может оказаться в начале окна, и алгоритму будет сложнее ее определить. При коэффициенте сдвига меньше 1 эта проблема будет частично решена.

2. **DataController**
   Сейчас задачей DataController является деление данных на большие части, благодаря чему можно избавится от коэффициента сдвига. DataController будет делить данные на равные части друг за другом. Имеет те же методы, что и Scrubber, так же есть setter для data.

3. **CPDCore**
   CPDCore имеет один метод run. Так как сценарий может быть изменен в оболочке, то при вызове run первым делом подставляется новый сценарий в скраббер и вызывается restart у scrubber и data_controller. Благодаря реализации DataController и Scrubber,

которые предоставляют итератор окон. Далее проверяется потребность в локализации и алгоритм высчитывает разладки.

4.2. Генератор данных

Важной частью работы с алгоритмами поиска разладкок являются данные. Они нужны для проверки работоспособности алгоритмов и качества работы CPDCore. Для этого в проекте нужен собственный генератор данных. За основу был взят уже реализованный в рамках проекта генератор [5].

Основной проблемой был неудобный конфигурационный файл (рис. 2). В нем задавалось название датасета, а потом указывались данные: список типов распределений, список длин и параметры распределений. При работе с большим количеством распределений становится проблематичным изменение параметров каждого. Также такой конфиг проблемно быстро прочитать.

[5]https://github.com/vkutuev/cpd-datasets. Дата обращения: 3 января 2025 г.

```yaml
- name: 20-normal-0-1-no-change-point
  distributions: [normal]
  lengths: [20]
  parameters:
    - mean: 0
      variance: 1
- name: 100-normal-0-1-no-change-point
  distributions: [normal]
  lengths: [100]
  parameters:
    - mean: 0
      variance: 1
- name: 20-normal-0-1-20-normal-10-1
  distributions: [normal, normal]
  lengths: [20, 20]
  parameters:
    - mean: 0
      variance: 1
    - mean: 10
      variance: 1
```

Рис. 2: Изначальный вид конфигурационного файла генератора данных.

Было принято решение перейти на другой вид конфига (рис. 3). Теперь после названия датасета сразу перечисляются распределения. Описание распределения содержит в себе данные о типе распределения, длине и параметрах. Это помогает проще читать и рефакторить файл.

```yaml
- name: 20-normal-0-1-no-change-point
  distributions:
    - type: normal
      length: 20
      parameters:
        mean: 0
        variance: 1
- name: 100-normal-0-1-no-change-point
  distributions:
    - type: normal
      length: 100
      parameters:
        mean: 0
        variance: 1
- name: 20-normal-0-1-20-normal-10-1
  distributions:
    - type: normal
      length: 20
      parameters:
        mean: 0
        variance: 1
    - type: normal
      length: 20
      parameters:
        mean: 10
        variance: 1
```

Рис. 3: Новый вид конфигурационного файла генератора данных.

Также при генерации данных есть возможность их сохранить. Для того чтобы открыть сгенерированные данные, создан статический метод read_generated_datasets у класса размеченных данных (LabeledCPData). Этот метод возвращает словарь, где по названию датасета лежит объект LabeledCPData.

Для упрощения анализа результатов работы ядра нужна визуализация в виде графика. В проекте есть два вида данных: размеченные (LabeledCPData) и неразмеченные (Sequence). Разница в том, что у размеченных данных заранее известны разладки. Для визуализации неразмеченных данных достаточно показать сами данные и найденные разладки (рис. 4). Для размеченных данных можно показывать не просто найденные разладки, а разделить их на правильно найденные, непра-

вильно найденные и не найденные (рис. 5, рис. 6).

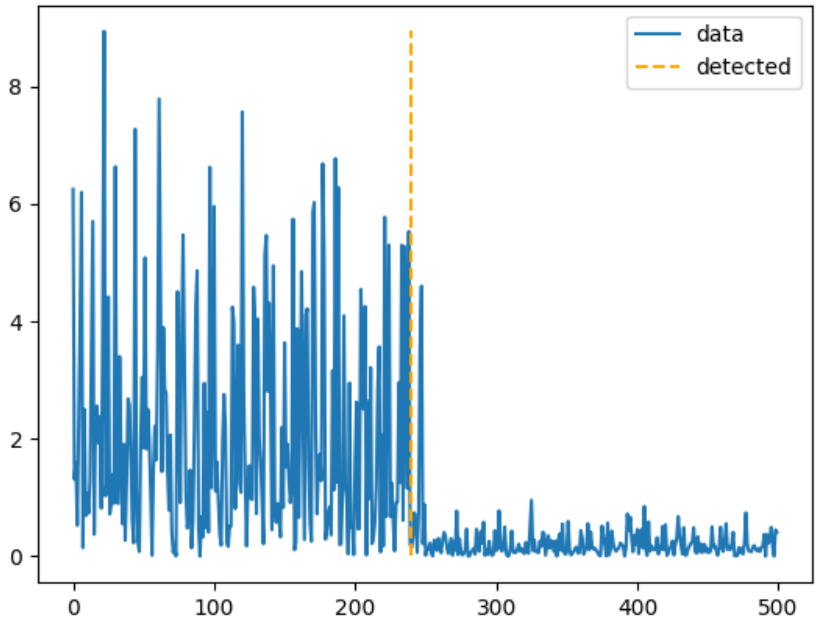

Рис. 4: Разладка в неразмеченных данных.

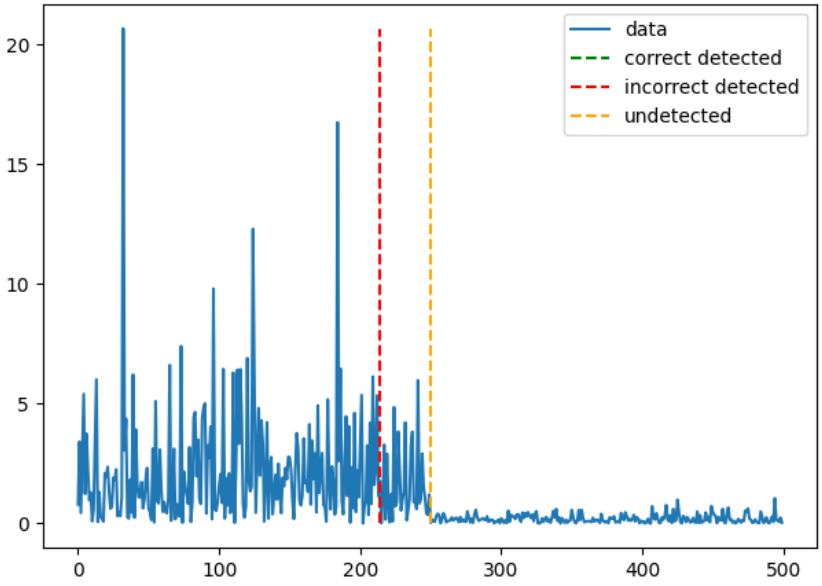

Рис. 5: Неправильно найденная и не найденная разладки.

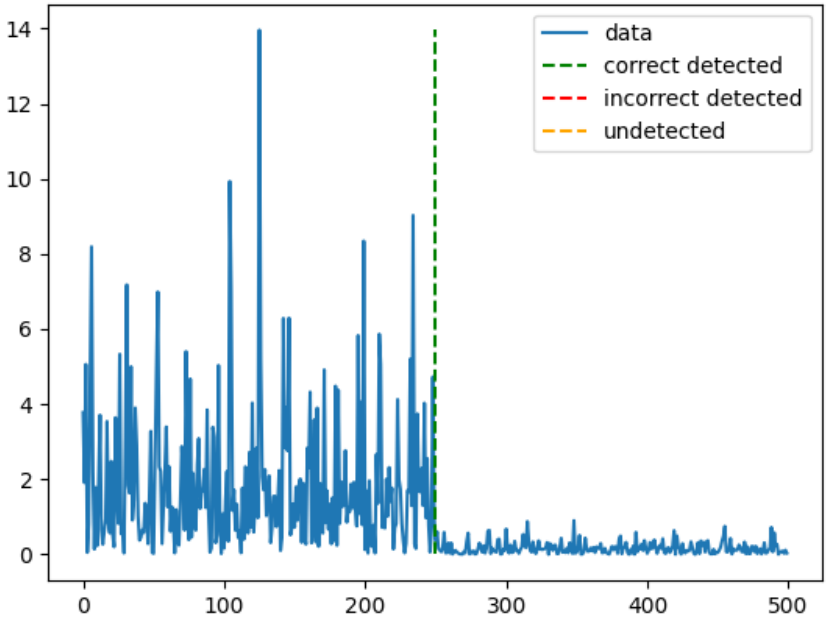

Рис. 6: Правильно найденная разладка.

Заключение

Результаты работы:

1. Проанализированы существующие python библиотеки определения разладки (ruptures, Kats), алгоритмы определения разладки (Байесовски, графовый, плотностные) и выделены требования к разрабатываемому продукту. (Разделы 2.2, 2.3, 2.4);

2. Спроектирована архитектура ядра модуля определения разладки (Раздел 3);

3. Реализовано ядро модуля определения разладки (Раздел 4.1);

   - Ссылка на репозиторий: https://github.com/PySATL/PySATL-CPD-Module;
   - Логин коммитера — wrdxwrdxwrdx (https://github.com/wrdxwrdxwrdx);
   - Ссылки на пулл реквесты [6] [7] [8] [9] [10] [11];

4. Написаны unit-тесты ко всем компонентам ядра (покрытие 96%).

[6] https://github.com/PySATL/PySATL-CPD-Module/pull/7
[7] https://github.com/PySATL/PySATL-CPD-Module/pull/8
[8] https://github.com/PySATL/PySATL-CPD-Module/pull/12
[9] https://github.com/PySATL/PySATL-CPD-Module/pull/14
[10] https://github.com/PySATL/PySATL-CPD-Module/pull/19
[11] https://github.com/PySATL/PySATL-CPD-Module/pull/30

Список литературы

[1] Adams Ryan Prescott, MacKay David J. C. Bayesian Online Changepoint Detection. — 2007. — 0710.3742.

[2] Chen Hao, Zhang Nancy. Graph-Based Change-Point Detection. — 2014. — 1209.1625.

[3] Jaxk Reeves Jien Chen Xiaolan L. Wang Robert Lund, Lu Qi Qi. A Review and Comparison of Changepoint Detection Techniques for Climate Data // Journal of Applied Meteorology and Climatology. — 2007. — P. 900–915.

[4] Kawahara Y., Sugiyama M. Sequential Change-Point Detection Based on Direct Density-Ratio Estimation // SIAM International Conference on Data Mining. — 2009. — P. 389–400.

[5] Relative Density-Ratio Estimation for Robust Distribution Comparison / Makoto Yamada, Taiji Suzuki, Takafumi Kanamori et al. // Advances in Neural Information Processing Systems. — Vol. 24. — Curran Associates, Inc., 2011.