

Санкт-Петербургский государственный университет

Патов Артемий Сергеевич

Выпускная квалификационная работа

Классификационные методы обнаружения разладки

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

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

Консультант:
инженер-исследователь, Лаборатория технологий программирования инфраструктурных
решений СПбГУ, В. А. Кутуев

Рецензент:
Старший инженер, ООО «Техкомпания Хуавэй», к.ф.-м.н., М. С. Куля

Санкт-Петербург
2025

Saint Petersburg State University

Artemii Patov

Bachelor's Thesis

Classification based change point detection methods

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2021 "Programming Technologies"*

Scientific supervisor:
Docent, C.Sc, V. I. Gorikhovskii

Consultant:
Research engineer at "St. Petersburg University Infrastructure Solutions Programming
Technologies Lab", V. A. Kutuev

Reviewer:
Senior Engineer at "Huawei Technologies", C.Sc., M. S. Coulas

Saint Petersburg
2025

Оглавление

1. Введение	5
2. Задачи обнаружения и локализации разладки	7
2.1. Статистическая постановка задач обнаружения и локализации разладки	7
2.2. Офлайн и онлайн-задачи	8
3. Обзор методов и инструментов для обнаружения разладки	9
3.1. Существующие методы обнаружения разладки	9
3.1.1. Метод обнаружения разладки с использованием ближайших соседей	10
3.2. Существующие инструменты для решения задачи обнаружения разладки	12
3.3. pysatl-cpd	12
4. Постановка задачи	14
5. Классификационные барьерные методы обнаружения разладки	15
5.1. Алгоритм скрабирования	15
5.2. Классификационные методы обнаружения разладки . . .	16
5.2.1. Классификаторы	18
5.2.2. Функции оценивания качества классификации . .	19
5.2.3. Критерии обнаружения разладки	19
6. Реализация методов	20
6.1. Архитектура алгоритма поиска разладки с использованием классификаторов	20
6.2. Реализация классификационных методов	21
6.3. Реализация метода с использованием ближайших соседей	21
7. Фреймворк для бенчмаркинга	23

7.1.	Схема фреймворка	23
7.2.	Требования к фреймворку	23
7.3.	Архитектура системы бенчмаркинга для скрабирующих алгоритмов	24
7.4.	Реализация фреймворка для бенчмаркинга	25
8.	Экспериментальное исследование реализованных алгоритмов	27
8.1.	Исследовательские вопросы	27
8.2.	RQ1: сравнительный анализ алгоритмов на основе синтетических данных	28
8.2.1.	Монте-Карло моделирование для подсчета порогов	29
8.2.2.	Монте-Карло моделирование для анализа мощностей	31
8.3.	RQ2: исследование относительной производительности реализованных алгоритмов	33
8.4.	RQ3: сравнение с оригинальной реализацией метода обнаружения разладки на основе графа ближайших соседей на многомерных данных	34
8.5.	RQ4: валидация реализованных алгоритмов на реальных данных	35
8.5.1.	Описание данных	35
8.5.2.	Результаты классификационных алгоритмов . . .	37
8.6.	Замечание о скорости обнаружения разладки	41
8.7.	Выводы по результатам экспериментального исследования	42
9.	Заключение	44
	Список литературы	47
А.	Приложение: Архитектура фреймворка для бенчмаркинга	50

1. Введение

Анализ временных рядов приобретает все большее значение в различных областях, включая медицину, аэрокосмическую отрасль, финансы, метеорологию. Временные ряды представляют собой последовательности измерений во времени, описывающие поведение систем, которое в силу различных обстоятельств может меняться с течением времени. Обнаружение точки разладки в последовательности наблюдений представляет собой задачу выявления резких изменений во входящем наборе данных, где могут происходить не только фундаментальные перемены, но и регулярные случайные колебания, которые не стоит принимать во внимание. Эта задача возникает в совершенно разных сферах: финансы [16, 21], где, например, необходимо оперативно регистрировать изменения обменного курса; мониторинг распространения эпидемических заболеваний [26], где важно отслеживать динамику развития эпидемии; безопасность [11], требующая немедленного реагирования на взломы.

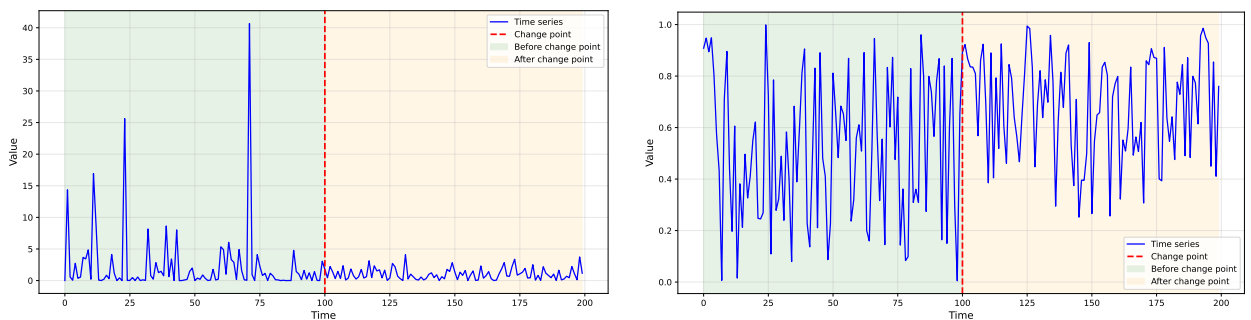


Рис. 1: Примеры данных с разладкой

В представленной работе исследуется применение простых классификаторов для решения задачи обнаружения разладки: разрабатывается обобщённый классификационный подход, использующий барьерную технику, заключающуюся в применении двухвыборочного тестирования к каждой точке входной выборки. Получившийся шаблонный метод обладает большой гибкостью с точки зрения конфигурирования: классификаторы, как и статистические критерии, взаимозаменяемы. Таким образом, число методов, получающихся в ходе подстановки различных

компонентов, выходит довольно большим.

В своем первоначальном виде классификационные методы решают офлайн-задачу (т. е. для известной выборки фиксированного размера). Однако алгоритмы, реализующие данный подход, возможно применять и в онлайн-режиме за счет использования техники скрабирования, позволяющей обрабатывать временной ряд по частям без требования его полной загрузки. Предлагаемый подход непараметрический, следовательно, не требует априорных знаний о свойствах временного ряда.

Для эмпирического изучения свойств реализованных алгоритмов разрабатывается система бенчмаркинга, позволяющая автоматически проводить эксперименты на синтетических наборах данных и сравнивать все алгоритмы на основе одних метрик. Благодаря тому, что алгоритмы интегрируются в проект `pysat1-cpd`, где другими участниками разрабатываются алгоритмы, основанные на различных подходах, появляется возможность сравнения алгоритмов на одинаковых данных. Также проводится апробация на реальных данных и сравнение с результатами алгоритмов из различных статей. Техники, методологии и инструменты, разработанные в представленной работе, применимы не только к классификационным алгоритмам, но и ко всему классу офлайн алгоритмов.

2. Задачи обнаружения и локализации разладки

Определения онлайн и офлайн-задач обнаружения разладки были сформулированы в обзорной статье [2] в виде проверки статистической гипотезы. Они представлены в настоящем разделе вместе с определением задачи локализации точки разладки.

2.1. Статистическая постановка задач обнаружения и локализации разладки

$Y = \{Y_1, \dots, Y_n\}$ — набор наблюдений. Y_i принимает значения в $\mathbb{R}^d$ ($d \geq 1$).

Нулевая гипотеза H_0 . Все наблюдения $Y = \{Y_1, \dots, Y_n\}$ следуют одному и тому же распределению F_0

$$H_0 : Y_i \sim F_0, i = 1, 2, \dots, n.$$

Альтернативная гипотеза H_1 . Существует момент разладки $\tau \in \{2, \dots, n-1\}$, такой что:

$$H_1 : Y_i \sim \begin{cases} F_0, i < \tau, \\ F_1, i \geq \tau, \end{cases}$$

где $F_0 \neq F_1$ (распределения до и после разладки различны).

Для проверки H_0 против H_1 вводится статистика разладки $S_n(\tau)$, которая оценивает различимость распределений до и после потенциальной точки τ .

Критерий обнаружения разладки. Гипотеза H_0 отвергается, если $S_n(\tau)$ превышает порог h_n , заданный уровнем значимости α .

Кроме того, в случае обнаружения разладки τ_{true} может рассматриваться дополнительная задача её *корректной локализации*. Точка разладки τ_{true} считается корректно локализованной оценкой $\tilde{\tau}$ с точностью

k , если выполнено условие

$$|\tilde{\tau} - \tau_{true}| \leq k.$$

В настоящей работе для локализации точки разладки используется следующий эвристический метод:

$$\tilde{\tau} = \arg \max_{1 < \tau < n} S_n(\tau)$$

Параметрические подходы используют предположения о входных данных: распределение F_0 должно быть заранее известно с точностью до параметров. *Непараметрические подходы* не требуют подобных предположений.

2.2. Офлайн и онлайн-задачи

В офлайн-задаче предполагается, что для анализа доступен полный набор наблюдений фиксированного объема.

В отличие от офлайн-задачи, в онлайн-задаче данные $Y = \{Y_1, Y_2, \dots, Y_t, \dots\}$ поступают последовательно во времени. Эта последовательность является потенциально бесконечной. Ключевая цель онлайн-обнаружения — выявить факт изменения статистических свойств ряда (точку разладки τ) как можно быстрее после того, как оно произошло, при этом поддерживая приемлемый уровень ложных тревог. На каждом временном шаге t , после поступления нового наблюдения (или блока наблюдений), система должна принять решение о наличии или отсутствии разладки, основываясь на доступной к этому моменту информации $Y = \{Y_1, \dots, Y_t\}$. Особенностью онлайн-режима является ограничение на время анализа на каждом шаге. Поэтому с поступлением новых наблюдений анализируются не все накопленные наблюдения, но только их релевантная часть.

3. Обзор методов и инструментов для обнаружения разладки

Область, связанная с решением задачи поиска разладки, активно развивается. Обзоры [2, 24] обладают подробным описанием терминологии, используемой во всей предметной области, и, помимо всего прочего, рассматривают возможные способы оценки качества работы методов и предлагают свои классификации методов. Первая работа концентрируется на офлайн-методах, вторая дает общую классификацию.

3.1. Существующие методы обнаружения разладки

Для решения задачи обнаружения разладки существует множество методов, вот лишь некоторые из них:

Байесовский [1] — параметрический онлайн-метод, анализирующий распределения длин пробегов (времени с момента последней обнаруженной точки разладки) с учетом априорных параметров и входных наблюдений.

CUSUM [13, 17] — параметрический онлайн-метод, основанный на накоплении отклонений наблюдаемых значений от ожидаемого (например, среднего).

Графовые — непараметрические офлайн-методы, использующие графы и алгоритмы для их анализа. Например, в работах [4, 28] строится граф расстояний на основе заданных наблюдений.

AR-Based (от англ. autoregression) [8, 10, 22] — класс алгоритмов, использующих авторегрессионные модели, рассматривающие каждую точку временного ряда вместе с множеством предшествующих точек.

DMD (от англ. dynamic mode decomposition — динамическая модовая декомпозиция) [25, 27] — метод, разлагающий временной ряд на

его доминирующие частотные компоненты (моды), описываемые их колебаниями и величинами, и отслеживающий их изменения во времени.

Говоря о методах, близких к классификационным, отметим несколько статей. Основой для представленной работы является публикация [3]: на основе описанного в ней метода в текущей работе строится его реализация, а затем его обобщение. С эмпирическими результатами, полученными в вышеупомянутой работе, будет проведено сравнение. Помимо этого, в статье представлены асимптотические оценки, которые полезно принять во внимание. В статье [12] приводятся непараметрический и параметрический методы, основанные на бинарной сегментации, производимой над всей выборкой. У метода есть ряд ограничений — во-первых, метод работает только в офлайн-режиме, во-вторых, для его применения необходимо априорное знание о количестве точек разладки. Благодаря технике скрабирования метод, предлагаемый в нынешней работе, сможет избавиться от первого ограничения, и он не использует подходов, накладывающих второе.

3.1.1. Метод обнаружения разладки с использованием ближайших соседей

Один из методов, реализуемых в представленной работе, основан на графе ближайших соседей [14, 19]. Далее представлен двухвыборочный тест на основе данного метода, который можно расширить до онлайн-метода, путем выделения выборок конечного размера из временного ряда.

Пусть $\{Y_1, \dots, Y_{n_1}\}$ и $\{Y_{n_1+1}, \dots, Y_{n_1+n_2}\}$ — случайные выборки из двух распределений, их размер $n = n_1 + n_2$. Пусть

$$b_{ij} = \mathbf{I}((i \leq n_1, j > n_1) \text{ или } (i > n_1, j \leq n_1)).$$

Пусть A_{ij} принимает значение 1, если $\mathbf{Y}_j$ среди первых k ближайших соседей $\mathbf{Y}_i$, в ином случае — 0. Тогда в качестве статистики для проверки

основной гипотезы может использоваться следующая функция

$$X = \sum_{i=1}^n \sum_{j=1}^n (A_{ij} + A_{ji})b_{ij}.$$

Её стандартизированная оценка имеет следующий вид:

$$S = \frac{X - E(X)}{\sqrt{\text{Var}(X)}},$$

где $E(X)$ и $\text{Var}(X)$ — математическое ожидание и дисперсия соответственно, которые можно рассчитать по формулам:

$$E(X) = \frac{4kn_1n_2}{n-1},$$

$$\begin{aligned} \text{Var}(X) = \frac{4n_1n_2}{n-1} & \left(h(n_1, n_2) \left(\frac{1}{n} \sum_{i,j=1}^n A_{ij}A_{ji} + k - \frac{2k^2}{n-1} \right) \right. \\ & \left. + (1 - h(n_1, n_2)) \left(\frac{1}{n} \sum_{i,j,l=1}^n A_{ji}A_{li} - k^2 \right) \right), \end{aligned}$$

где $h(n_1, n_2) = \frac{4(n_1-1)(n_2-1)}{(n-2)(n-3)}$.

Приведенная выше стандартизированная оценка S сравнивается с порогом, преодоление которого говорит о наличии разладки.

В рассматриваемом методе можно выделить 4 основных этапа: разбиение выборки барьером, применение графа ближайших соседей, построение статистики и применение критерия обнаружения разладки. В разделе 5.2 рассматривается обобщение данного подхода: вместо графа ближайших соседей применяются различные классификаторы. Статистика строится на основе предсказаний классификатора.

3.2. Существующие инструменты для решения задачи обнаружения разладки

Авторы исследования [2] предлагают свой пакет для обнаружения точек разладки `ruptures`¹. Однако он имеет ряд ограничений: отсутствует функциональность для обработки временных рядов по частям, поддерживается только офлайн-режим обнаружения разладки, отсутствует необходимая инфраструктура для проведения экспериментов. Кроме того, существует известный пакет `kats` [9], который обладает точно такими же ограничениями. Вышеупомянутые пакеты сконцентрированы на конкретных классах методов, их инфраструктура не обладает достаточной гибкостью для исследования других семейств методов. В проекте `PySATL` проводится комплексное исследование статистических алгоритмов. В частности, задача обнаружения разладки рассматривается в `pysatl-cpd`². Цель `pysatl-cpd` — создание инфраструктуры для поддержания широкого класса методов и сравнения их между собой. Следовательно, алгоритмы, реализуемые в представленной работе, стоит внедрять в `pysatl-cpd`.

3.3. `pysatl-cpd`

Код проекта `pysatl-cpd` написан на языке Python, выбор языка вполне обоснован: разработка ориентирована не на эффективность самих реализаций, а на возможность быстро провести всестороннее изучение статистических свойств методов при помощи инструментов, предоставляемых языком.

В `pysatl-cpd` на момент написания работы реализованы некоторые алгоритмы обнаружения разладки, например, Байесовский (онлайн, параметрический) и графовый (офлайн, непараметрический), с которыми имеется возможность сравнить реализуемые в рамках текущей работы классификационные методы.

Помимо этого, в `pysatl-cpd` поддержан скрабер, позволяющий ра-

¹`ruptures`: <https://github.com/deepcharles/ruptures> (Дата обращения: 24.05.2025)

²`pysatl-cpd`: <https://github.com/PySATL/pysatl-cpd> (Дата обращения: 24.05.2025)

ботать с офлайн-алгоритмом в онлайн-режиме, подавая в алгоритм небольшими порциями поступающий временной ряд. Также в проекте реализована базовая инфраструктура для проведения экспериментов, которой недостаточно для полноценного исследования реализуемых алгоритмов, однако её несложно расширить. Говоря конкретно, поддерживаются генераторы для сэмплирования различных распределений, однако отсутствует система для проведения бенчмаркинга алгоритмов на основе синтетических данных.

4. Постановка задачи

Цель — исследование классификационного подхода для решения задачи обнаружения разладки в рамках проекта `pysatl-cpd`.

Для достижения поставленной цели были выделены следующие задачи:

- Разработка обобщенного классификационного барьерного метода обнаружения разладки.
- Реализация алгоритмов обнаружения разладки, основанных на обобщенном классификационном подходе, с использованием классификаторов KNN, Kernel SVM, Decision Tree, Random Forest.
- Реализация фреймворка для бенчмаркинга, предоставляющего возможность сэмплировать данные с различными распределениями и проводить на основе сгенерированных датасетов оценивание эффективности алгоритмов по определенным метрикам.
- Оценка эффективности реализованных алгоритмов с точки зрения статистической мощности и производительности для определенного набора распределений.
- Анализ зависимости качества обнаружения разладки от выбора классификатора и параметров выборки.
- Сравнение эффективности реализованных алгоритмов на реальных данных с существующими решениями.

5. Классификационные барьерные методы обнаружения разладки

В данном разделе предлагается оригинальный подход к обнаружению разладки в онлайн-режиме с использованием классификаторов и барьерной техники. В работе рассматриваются различные комбинации классификаторов (KNN, SVM, Decision Tree, Random Forest) и метрик качества (F1, MCC и др.), что обеспечивает гибкость подхода к различным типам распределений. Для обработки потоковых данных в онлайн-режиме используется алгоритм скрабирования (см. раздел 5.1), который адаптивно управляет окном анализа, сводя задачу к офлайн-постановке. В частности, техника скрабирования применима к классификационным алгоритмам обнаружения разладки, представленным в разделе 5.2.

5.1. Алгоритм скрабирования

Задача обнаружения разладки может решаться в совершенно разных контекстах. В случае офлайн-задачи выборка может обрабатываться как единым целым, так и по частям, при помощи скользящего окна: рассматривается ограниченное количество подряд идущих точек, и когда обработка такого интервала заканчивается, он сдвигается далее по временному ряду, захватывая новые точки и забывая о таком же количестве старых точек.

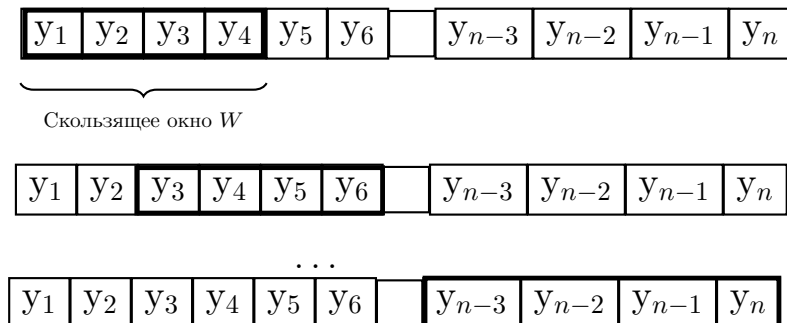


Рис. 2: Скользящее окно. Частный случай техники скрабирования

В случае решения онлайн-задачи обработка временного ряда про-

исходит при помощи алгоритма скрабирования: в отличие от скользящего окна, скрабирование, в случае нехватки количества имеющихся наблюдений для их анализа, предполагает ожидание, пока не поступит достаточное число точек, чтобы их можно было анализировать. Помимо этого, скрабирование может иметь несколько сценариев: например, возможна линейная обработка имеющихся данных и, соответственно, линейное забывание. В таком случае скрабирование будет отличаться от скользящего окна только ленивой подгрузкой данных. Скрабер может быть настроен на логарифмическое забывание данных и на линейную подгрузку новых точек. В таком случае множество точек для обработки с каждым шагом будет увеличиваться. Помимо этого, рассматриваемые скрабером наблюдения могут быть не подряд идущими, возможно брать совершенно разные части временного ряда и так далее.

Алгоритмы, рассматриваемые в данной работе, применяются совместно со скрабером. Окна скрабера имеют небольшой размер, что позволяет с применением легковесных методов быстро выдавать решение.

Специфика работы алгоритма в рамках окна состоит в том, что крайние части окна рассматривать не стоит: в этих частях недостаточно наблюдений для анализа. Поэтому в окне выделяется рабочая область: точки, в которых проверяется гипотеза о наличии разладки.

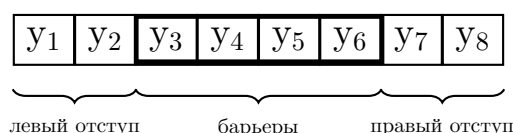


Рис. 3: Пример окна скрабера. Длина отступов от краев окна — 2 элемента. Рабочая область состоит из 4 центральных элементов.

5.2. Классификационные методы обнаружения разладки

Основанием для концепции обобщенного классификационного барьерного подхода послужили идеи из статьи [3]. А именно, выбор барьера во временном ряду и оценка качества разбиения им временного ряда

на две части. На основе рассмотренного в разделе 3.1.1 метода строится обобщенный классификационный метод обнаружения разладки. Идея обобщенного метода состоит в том, что для оценки качества разбиения временного ряда можно использовать классификатор и функцию для оценки качества классификации, которые можно варьировать. Конкретно в данной работе будут использоваться KNN, SVM, Decision Tree и Random Forest классификаторы. Что касается функций оценки качества, используются F1, MCC. Псевдокод алгоритма можно увидеть ниже (Алгоритм 1).

Algorithm 1 CPD-алгоритм с использованием классификатора

```

1: Input: sample  $y_1, \dots, y_n$ , classifier, quality_metric, threshold,
2: left indent, right indent
3: Output: Detected change_points
4:
5: Split sample into train_sample and test_sample
6: change_points  $\leftarrow []$ 
7:
8: for barrier  $\leftarrow$  left to right do
9:   classifier.fit(barrier, train_sample)
10:  classes  $\leftarrow$  classifier.predict(test_sample)
11:  statistics  $\leftarrow$  quality_metric(barrier, classes)
12:
13:  if statistics  $>$  threshold then
14:    change_points.append(barrier)
15: return change_points

```

Алгоритм разбивает входные данные на две части: *train* и *test* (Строка 5). Далее алгоритм итерируется по центральной части (рабочей области) входных данных. Каждый элемент рабочей области рассматривается в качестве барьера. Классификатор *classifier* обучается на точках, не являющихся барьерами (Строка 9), а затем предсказывает классы элементов из выборки *test* (Строка 10). Далее оценивается качество классификации при помощи метрики *quality_metric* (Строка 11). На строке 13 полученная оценка сравнивается с порогом *threshold*. Если значение метрики больше порога, то точка, рассматриваемая в качестве барьера, считается разладкой (см. рис. 4). Видно, что все главные

составляющие параметризуются.

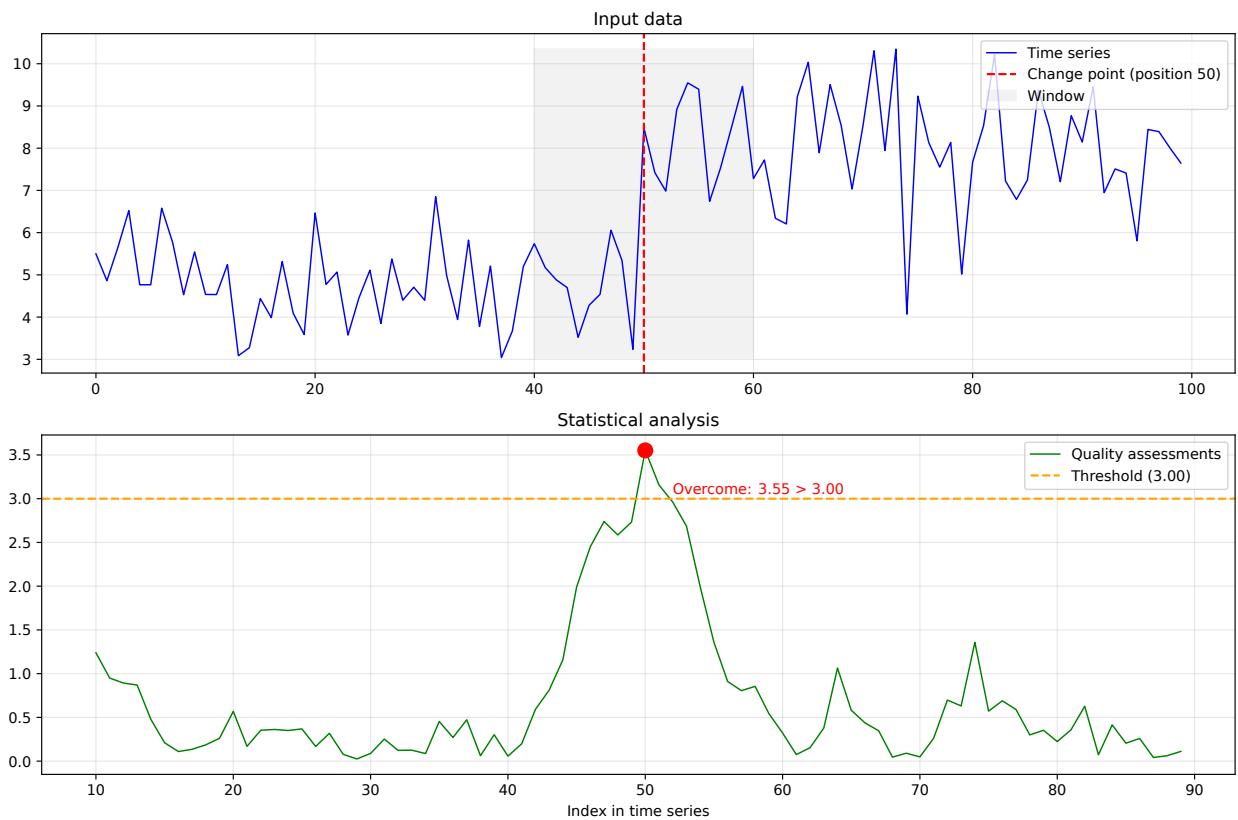


Рис. 4: Пример работы алгоритма. Точка с максимальным значением статистики ($3.55 > 3.0$, порог преодолен) является точкой разладки

5.2.1. Классификаторы

Вследствие того, что метод предполагается использовать в условиях ограниченных ресурсов, рассматривались только легковесные классификаторы: SVM³, Random Forest⁴, Decision Tree⁵, KNN [3]. Вышеперечисленные классификаторы достаточно чувствительны, чтобы не упустить изменения в окне небольшого размера. Все из них имеют свои области применимости, одна из целей данной работы — определить эти области.

³SVM: <https://scikit-learn.org/stable/modules/svm.html> (Дата обращения: 24.05.2025)

⁴Random Forest: <https://scikit-learn.org/1.5/modules/ensemble.html#random-forests> (Дата обращения: 24.05.2025)

⁵Decision Tree: <https://scikit-learn.org/1.5/modules/tree.html> (Дата обращения: 24.05.2025)

5.2.2. Функции оценивания качества классификации

Функции оценивания качества классификации были выбраны стандартные, основанные на матрице несоответствий. Однако в данном случае отрицательные значения будут соответствовать классу 0, положительные — классу 1. Как и в случае с классификаторами, функции имеют свои границы применимости.

$$F_1(W, \tau) = \frac{2TP}{2TP + FP + FN}$$

Важно отметить, что F_1 несимметрична относительно перестановки классов (т.е. её значение изменится, если поменять местами метки классов) и не учитывает истинно отрицательные значения (TN).

$$MCC(W, \tau) = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Подобно F_1 , MCC также несимметрична при простой перестановке меток классов. Данная метрика чувствительна к несбалансированным данным. Однако, поскольку MCC учитывает все значения матрицы несоответствий, во многих ситуациях она способна предоставлять более близкие к истинным оценки [6, 5] по сравнению с F_1 -мерой, особенно при работе с несбалансированными наборами данных.

Были опробованы и другие меры: дискриминантные [20], ассигасу, AGF, Youden, AGM [23], однако все они оказались менее эффективными, чем MCC.

5.2.3. Критерии обнаружения разладки

Статистический критерий принимает решение о наличии разладки на основе оценок, полученных после применения метрики качества к точкам из входного окна. Его реализации состоят из двух частей: обработка массива со значениями метрики для точек из рабочей области, сравнение с порогом. Пороги могут быть подсчитаны на основе Монте-Карло моделирования в офлайн-режиме.

6. Реализация методов

Код реализованных методов находится в главной ветке проекта `pysatl-cpd`⁶.

6.1. Архитектура алгоритма поиска разладки с использованием классификаторов

На рис. 5 приведена архитектура метода.

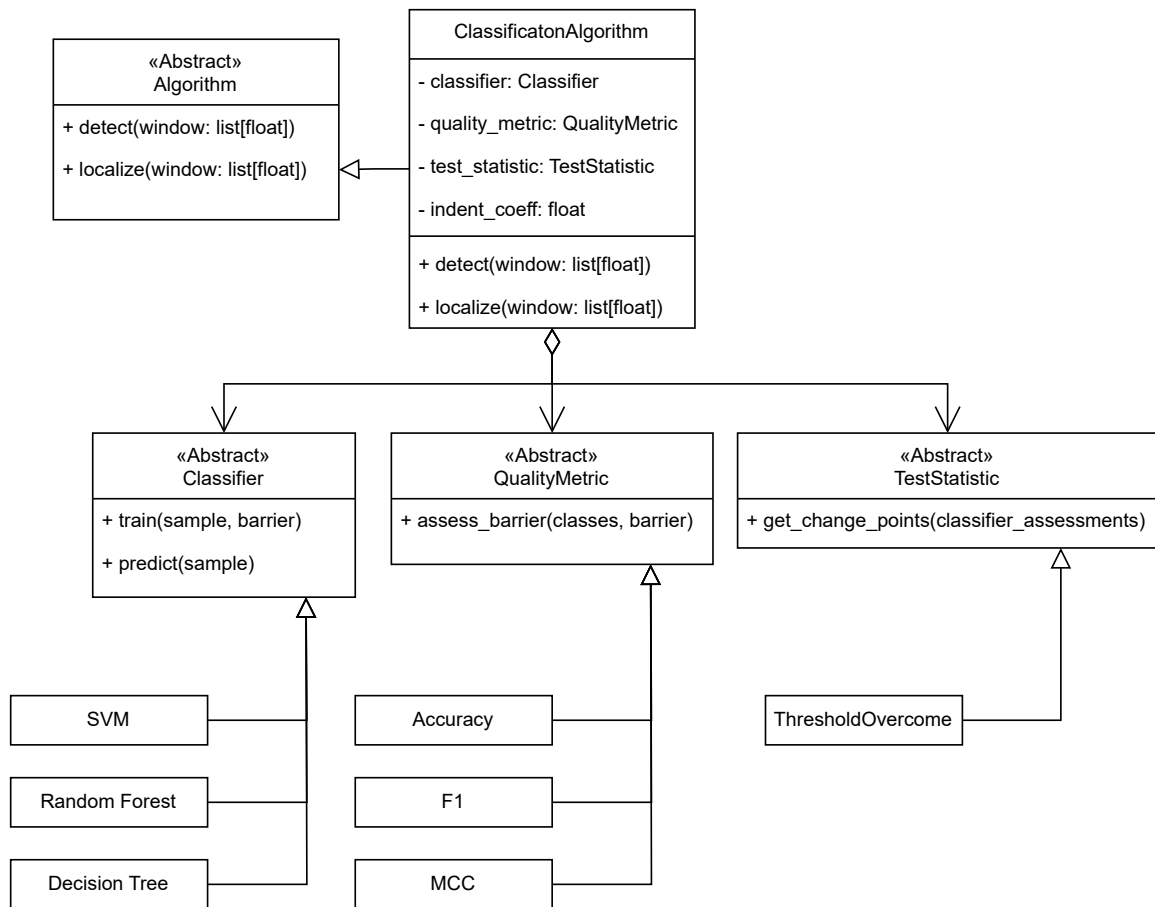


Рис. 5: Архитектура обобщенного метода поиска разладки с использованием классификаторов.

Классы `Classifier`, `QualityMetric` и `TestStatistic` являются абстракциями над классификатором, метрикой качества классификации и статистическим критерием соответственно.

⁶`pysatl-cpd`: <https://github.com/PySATL/pysatl-cpd> (Дата обращения: 24.05.2025)

Класс `ClassificationAlgorithm` реализует паттерн "Шаблонный метод": переданные параметры (классификатор, метрика качества, статистический критерий) собираются в общую для всех классификационных алгоритмов последовательность действий: вызывается классификатор, после чего оценивается качество классификации, на множестве агрегированных значений метрики применяется статистический критерий.

6.2. Реализация классификационных методов

Библиотека `scikit`⁷ предоставляет все необходимые классификаторы (Раздел 5.2.1). Она активно используется в реализации. В большинстве случаев были сохранены стандартные конфигурации классификаторов, однако для некоторых методов проводились дополнительные эксперименты по подбору оптимальных гиперпараметров. Текущие параметры по умолчанию:

KNN: количество соседей $k = 7$.

Decision Tree: узлы расширяются до тех пор, пока все листья не станут чистыми или пока все листья не будут содержать менее 2 сэмплов.

Random Forest: 100 деревьев; параметры отдельных деревьев соответствуют параметрам Decision Tree.

Kernel SVM: используется линейное ядро.

6.3. Реализация метода с использованием ближайших соседей

Реализацию алгоритма обнаружения разладки с использованием графа ближайших соседей было принято решение не подводить под

⁷<https://github.com/scikit-learn/scikit-learn> (Дата обращения: 24.05.2025)

интерфейс, представленный на рис. 5: его реализация строится на основе подхода, описанного в статье [3] и использующего классификатор с метрикой монолитно. Их разделение чревато потерей производительности. Помимо этого, была проведена некоторая оптимизация подсчета статистики, что существенно увеличило его эффективность.

7. Фреймворк для бенчмаркинга

Для автоматизации проведения экспериментального исследования и сравнения реализованных алгоритмов необходимо разработать фреймворк для бенчмаркинга. В данной секции приводится архитектура фреймворка, а затем описываются детали реализации его прототипа.

7.1. Схема фреймворка

В смежном проекте, PySATL-experiment⁸, ранее была спроектирована схема фреймворка (см. рис. 6) для экспериментов в самом общем виде.

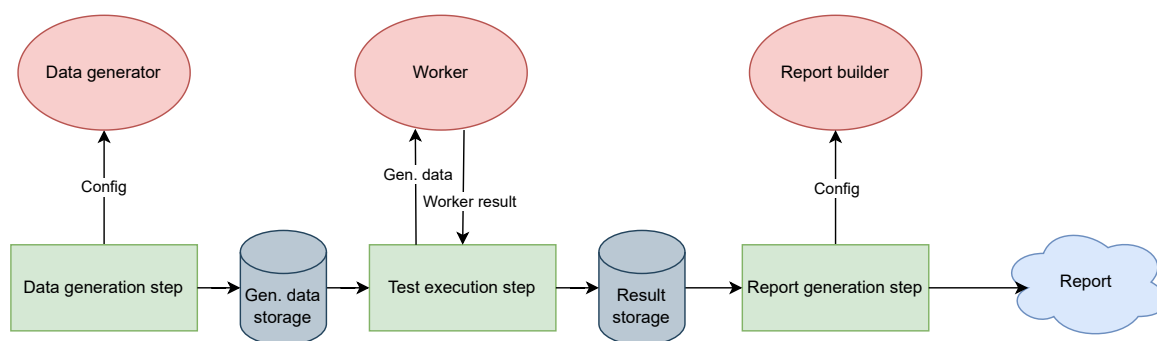


Рис. 6: Устройство пайплайна для экспериментов в общем виде.

Пайплайн состоит из 3 изолированных шагов:

1. Генерация данных по заданным распределениям;
2. Запуск рабочей стадии, в процессе которой данные проходят через алгоритм, сохраняется необходимая статистика;
3. Подсчет метрик на основе сохраненной статистики, генерация отчета.

7.2. Требования к фреймворку

Для бенчмаркинга скрабирующих методов поиска разладки были выдвинуты следующие требования:

⁸PySATL-experiment: <https://github.com/PySATL/pysatl-experiment> (Дата обращения: 24.05.2025)

- Сохранение следующих метрик бенчмаркинга:
 - Общая временная оценка.
 - Время, за которое алгоритм анализирует одно окно скраббера.
 - Общие затраты по памяти.
 - Значения матрицы несоответствий.
- Пайплайн должен соответствовать схеме из секции 7.1.
- Возможность сравнения различных скрабирующих алгоритмов по определённым общим метрикам.
- Возможность сравнивать скрабирующие алгоритмы с онлайн-алгоритмами по общим для обоих алгоритмов метрикам.

7.3. Архитектура системы бенчмаркинга для скрабирующих алгоритмов

Чтобы удовлетворить вышеперечисленным требованиям, были приняты следующие архитектурные решения.

Некоторые метрики бенчмаркинга требуют дополнительной информации от алгоритмов. Решено разработать вторую иерархию алгоритмов, повторяющую уже существующую, с той разницей, что алгоритмы в процессе работы будут дополнительно сохранять необходимую для экспериментов информацию, а также реализовывать интерфейс `BenchmarkingAlgorithm`, имеющий метод, позволяющий получать информацию для бенчмаркинга. Иерархию наследования надлежит сохранить: это позволит сравнивать разные алгоритмы по сходным метрикам (см. приложение А, рис. 18), выбрав общий узел в иерархии. Архитектура приведена на рис. 17 (см. приложение А).

Генератор должен работать в двух режимах: генерировать данные с разладкой и генерировать данные без разладки (см. приложение А, рис. 19).

Архитектура воркера представлена на рис. 20 (см. приложение А). Конкретный **Worker** может быть одним из двух типов: он может либо решать задачу оптимизации, высчитывая оптимальный параметр для алгоритма по, например, заданному уровню значимости, либо проводить бенчмаркинг алгоритма на основе заданных извне параметров. В первом случае воркер (объект **Worker**) работает с гипотезой, что в данных нет разладки, во втором случае — что она есть. В связи с этим нужны гарантии, что генератор не будет создавать данные, не удовлетворяющие предположениям. Это обеспечит паттерн "Абстрактная фабрика", описание которой будет приведено ниже.

Использование абстрактной фабрики приведено на рис. 21 (см. приложение А). Конкретные фабрики реализуют методы, создающие объекты генератора, воркера и отчета, так чтобы обеспечивать согласованность между стадиями бенчмаркинга. Например, воркер, предполагающий, что в данных нет разладки, требует генерацию данных без разладки, это обеспечивает **NoCPGenerator**.

Результаты стадий поступают в хранилища, через которые части системы взаимодействуют между собой.

Отчет должен быть согласован с замерами, которые делает воркер. На основе них при генерации отчета вычисляются необходимые метрики. Для разных воркеров подходят разные метрики; выбор нужных метрик может происходить за счет использования паттерна "Строитель".

7.4. Реализация фреймворка для бенчмаркинга

Прототип фреймворка был реализован в соответствии с представленной выше архитектурой. Были реализованы части **Generator**, на основе генераторов, ранее поддержанных в проекте, **Worker** для решения задачи оптимизации, возникающей при подсчете порогов и для проведения бенчмаркинга с различными метриками (сохранение значений метрик качества, времени работы алгоритма) и **Report**, реализующий паттерн "Строитель", обрабатывающий артефакты воркера. Этих частей было достаточно для проведения экспериментов, представленных

в разделе 8. Код реализованной системы бечмаркинга можно увидеть в ветке `scrubbing-methods-bench`⁹ репозитория проекта `pysatl-cpd`.

⁹Прототип фреймворка для бенчмаркинга: <https://github.com/PySATL/pysatl-cpd/tree/scrubbing-methods-bench> (Дата обращения: 24.05.2025)

8. Экспериментальное исследование реализованных алгоритмов

Цель экспериментального исследования — проведение сравнительного анализа эффективности реализованных алгоритмов на основе синтетических датасетов и реальных данных.

Характеристики системы:

- Аппаратное обеспечение: Intel® Core™ i7-4790 CPU @ 3.60 GHz (4 ядра, 8 потоков), 32GiB RAM.
- Программное обеспечение: Ubuntu 22.04, Python 3.12.3.

Исследуемые алгоритмы:

- **7-NN CM:** KNN с $k = 7$ и комбинаторной функцией (CM) оценивания качества классификации (см. раздел 3.1.1).
- **7-NN F_1 :** KNN с $k = 7$ и функцией оценивания качества классификации F_1 .
- **7-NN MCC:** KNN с $k = 7$ и функцией MCC.
- **DT MCC:** Decision Tree с функцией MCC.
- **RF MCC:** Random Forest с функцией MCC.
- **Lin. SVM MCC:** Kernel SVM с линейным ядром и функцией MCC.
- **RBF SVM MCC:** Kernel SVM с ядром RBF и функцией MCC.

8.1. Исследовательские вопросы

Перед проведением экспериментального исследования были поставлены следующие исследовательские вопросы.

- RQ1. Насколько качественно реализованные алгоритмы справляются с обнаружением разладки на синтетических датасетах с различными одномерными распределениями в офлайн-режиме?
- RQ2. Как быстро реализованные алгоритмы анализируют синтетические датасеты с различным количеством точек разладки в офлайн-режиме?
- RQ3. Насколько качественно реализованные алгоритмы справляются с обнаружением разладки в многомерных данных в сравнении с реализацией из статьи [3]?
- RQ4. Насколько алгоритмы эффективны на реальных данных, рассмотренных в других работах?

8.2. RQ1: сравнительный анализ алгоритмов на основе синтетических данных

Для сравнительного анализа алгоритмов обнаружения разладки был проведен вычислительный эксперимент на синтетических данных с контролируемыми параметрами. Каждый тестовый датасет содержал 10 000 независимых реализаций временных рядов длины 200 с контролируемой точкой разладки в позиции $\tau = 100$. В рамках одного датасета все ряды генерировались по схеме:

$$Y_i \sim \begin{cases} F_0 & \text{при } i < 100, \\ F_1 & \text{при } i \geq 100, \end{cases}$$

где F_0 и F_1 — различные распределения (см. Таблицу 1). Особое внимание уделялось трём ключевым сценариям: изменение стандартного отклонения, сдвиг математического ожидания и изменение носителя распределения. Метрикой бенчмаркинга являлась статистическая мощность (вероятность отвержения нулевой гипотезы H_0 (см. раздел 2)). Подбор порогов осуществлялся на основе выборок из одномерных распределений для фиксированного уровня значимости $\alpha = 5\%$.

Во всех случаях были взяты ранее подобранные гиперпараметры (Секция 6.2). Количество элементов в окне скрабера было 48 элементов. Длина отступа с обеих сторон окна была 12 элементов, следовательно, рабочая область состояла из 24 центральных элементов. Эти гиперпараметры скрабера оказались оптимальными в случае быстрой обработки окна, но для конкретных случаев и для конкретных методов можно подобрать индивидуально более оптимальные.

Обозначение	Тип	Параметры
$\mathcal{N}^1$	normal	$\mu = 0.0, \sigma = 1.0$
$\mathcal{N}^2$	normal	$\mu = 0.0, \sigma = 3.0$
$\mathcal{N}^3$	normal	$\mu = 2.0, \sigma = 3.0$
$\mathcal{N}^4$	normal	$\mu = 2.0, \sigma = 1.0$
$\mathcal{U}^1$	uniform	min = 0.0, max = 1.0
$\mathcal{U}^2$	uniform	min = 0.0, max = 2.0
$\mathcal{U}^3$	uniform	min = -1.0, max = 2.0
$\mathcal{U}^4$	uniform	min = 0.25, max = 1.0
$\mathcal{U}^5$	uniform	min = 1.0, max = 1.25
$\mathcal{B}^1$	beta	$\alpha = 1.0, \beta = 1.0$
$\mathcal{B}^2$	beta	$\alpha = 0.5, \beta = 0.5$
$\mathcal{B}^3$	beta	$\alpha = 2.0, \beta = 2.0$
$\mathcal{W}^1$	weibull	<i>shape</i> = 0.5, <i>scale</i> = 1.0
$\mathcal{W}^2$	weibull	<i>shape</i> = 1.0, <i>scale</i> = 1.0
$\mathcal{W}^3$	weibull	<i>shape</i> = 2.0, <i>scale</i> = 1.0

Таблица 1: Одномерные распределения

8.2.1. Монте-Карло моделирование для подсчета порогов

Прежде чем приступать к замерам, необходимо подобрать оптимальные пороги. Был разработан алгоритм для автоматического подсчёта порога по заданному уровню значимости. Алгоритм поиска разладки запускается один раз на сгенерированных выборках без разладки, сохраняя статистику — каждая точка выборки рассматривается в качестве барьера, и для неё подсчитывается функция оценки качества классификации. Далее решается задача оптимизации: бинарным поиском

подсчитывается порог, приближая уровень значимости к необходимому значению.

Был взят стандартный набор уровней значимости 0.5%, 1% и 5%. Целью был подсчет оптимальных значений порога для указанных выше датасетов; были взяты распределения до достижения разлаженного состояния, на основе них было сгенерировано 10000 выборок без разладки. Порог высчитывался индивидуально для каждой тройки алгоритм-распределение-уровень значимости. Для $\alpha = 5\%$ высчитанные пороги можно увидеть в таблице 2, примеры высчитанных порогов для остальных уровней значимости изображены на рис. 7.

Распред.	CM	F ₁	MCC				
	7-NN	7-NN	7-NN	Lin. SVM	RBF SVM	RF	DT
$\mathcal{N}^1$	3.25	0.859	0.5	0.414	0.453	0.516	0.516
$\mathcal{U}^1$	3.25	0.859	0.5	0.344	0.453	0.516	0.516
$\mathcal{B}^1$	3.25	0.859	0.484	0.344	0.453	0.516	0.516
$\mathcal{W}^1$	3.25	0.859	0.5	0.391	0.406	0.516	0.516
$\mathcal{W}^2$	3.25	0.859	0.5	0.406	0.438	0.516	0.516
$\mathcal{W}^3$	3.25	0.859	0.5	0.375	0.453	0.516	0.516
MN^1	2.813	0.878	0.5	0.531	0.484	0.563	0.57
MN^2	2.625	0.878	0.484	0.52	0.344	0.516	0.578
MN^3	2.625	0.878	0.5	0.486	0.34	0.488	0.578

Таблица 2: Подсчитанные пороги. $\alpha = 5\%$

Сопоставляя порог для метода на основе KNN, высчитанный в рамках текущей работы, с порогами из статьи [3], можно увидеть, что они не сильно отличаются, возможна небольшая погрешность из-за различных уровней значимости и других распределений. Что касается других методов, то стоит отметить, что для них пороги лежат в промежутке от -1 до 1, это свойство метрики MCC. Для F₁ пороги лежат в границах от 0 до 1.

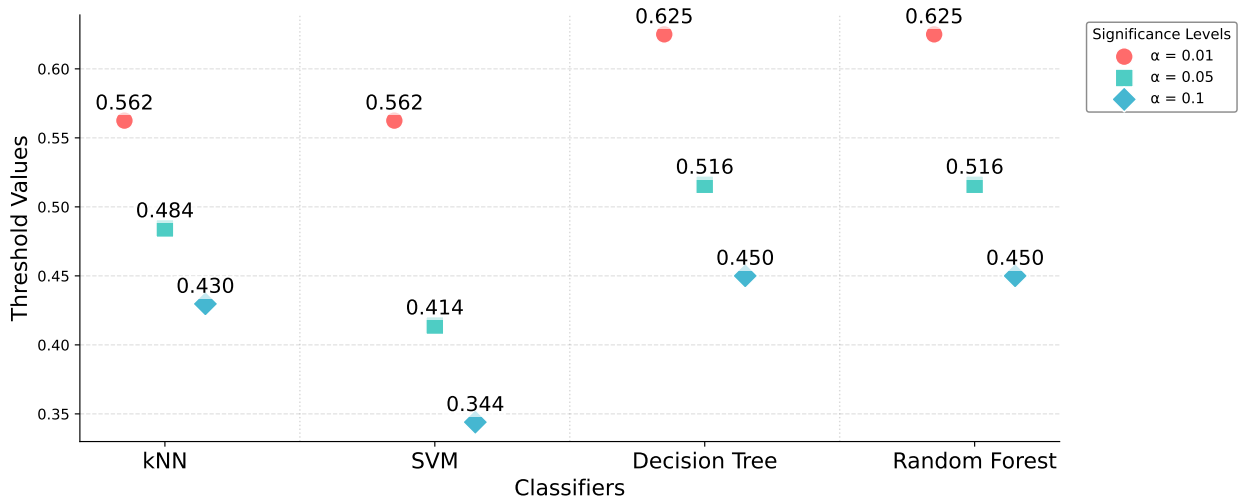


Рис. 7: Подсчитанные пороги для разных α

8.2.2. Монте-Карло моделирование для анализа мощностей

Сравним реализованные методы на основе одномерных датасетов, используя подсчитанные пороги (см. таблицу 2). Для этого подсчитаем статистическую мощность для каждой пары алгоритм-распределение. Результаты представлены в таблице 3, значения отсортированы по столбцу 7-NN CM.

Начнем анализ с худших результатов: разладка в данных с распределением $\mathcal{B}^1 - \mathcal{B}^2$ почти не обнаруживается: это можно объяснить достаточной схожестью бета распределений с взятыми параметрами. Их близость можно проверить при помощи подсчета функций правдоподобия для каждой из точек носителя: в большинстве случаев они будут очень близки.

В рамках распределения $\mathcal{U}^1 - \mathcal{U}^4$ меняется носитель, однако переменная недостаточно сильна, чтобы это увидели алгоритмы.

Что касается распределения Вейбулла, то за основу было взято три конфигурации и их симметрии. Видна одна тенденция: алгоритмы справляются хуже, если распределение с более тяжелым хвостом приходится на часть, идущую до разладки. В случае 7-NN это видно ярче всего: численная разница равна примерно 7%.

С распределения $\mathcal{N}^1 - \mathcal{N}^2$ начинается содержательное сравнение. Стоит отметить два факта: 7-NN CM — единственный алгоритм, ко-

Распред.	CM	F ₁	MCC				
	7-NN	7-NN	7-NN	RF	DT	Lin. SVM	RBF SVM
$\mathcal{B}^1 - \mathcal{B}^2$	0.1681	0.055	0.128	0.1022	0.1017	0.0587	0.1427
$\mathcal{U}^1 - \mathcal{U}^4$	0.2362	0.066	0.153	0.1361	0.1357	0.1479	0.206
$\mathcal{W}^2 - \mathcal{W}^3$	0.2674	0.075	0.22	0.1474	0.1467	0.1053	0.2479
$\mathcal{W}^1 - \mathcal{W}^2$	0.2871	0.051	0.182	0.1502	0.1499	0.1527	0.2036
$\mathcal{W}^3 - \mathcal{W}^2$	0.3426	0.058	0.147	0.1648	0.1644	0.1446	0.2494
$\mathcal{W}^2 - \mathcal{W}^1$	0.3440	0.073	0.209	0.1572	0.1561	0.1322	0.1527
$\mathcal{N}^1 - \mathcal{N}^2$	0.7858	0.081	0.417	0.3749	0.373	0.2945	0.6974
$\mathcal{W}^1 - \mathcal{W}^3$	0.8761	0.131	0.549	0.4281	0.427	0.2607	0.5427
$\mathcal{U}^1 - \mathcal{U}^2$	0.9003	0.11	0.657	0.4253	0.4227	0.922	0.8617
$\mathcal{W}^3 - \mathcal{W}^1$	0.9057	0.105	0.457	0.4986	0.4962	0.3393	0.5212
$\mathcal{N}^1 - \mathcal{N}^3$	0.9518	0.136	0.796	0.515	0.5134	0.8552	0.8675
$\mathcal{U}^1 - \mathcal{U}^3$	0.963	0.124	0.574	0.6572	0.6549	0.55	0.9503
$\mathcal{N}^1 - \mathcal{N}^4$	0.9976	0.452	0.94	0.7529	0.7516	0.9898	0.9667
$\mathcal{U}^1 - \mathcal{U}^5$	1.0	0.98	1.0	1.0	1.0	1.0	1.0

Таблица 3: Мощность. Одномерные распределения

торый качественно справился с задачей обнаружения, однако видно, что смена ядра в SVM с Linear на RBF значительно увеличила вероятность обнаружения. Легко объяснимо, почему DT отработал хуже Linear SVM: в одномерном случае DT способен подрезать носитель на несколько мелких отрезков, отнеся каждый к одному из двух классов, Linear SVM делит лишь на два отрезка, в связи с чем его точность хуже.

В случае с $\mathcal{U}^1 - \mathcal{U}^2$ и $\mathcal{U}^1 - \mathcal{U}^3$ происходит значительная смена носителя. В случае с 7-NN CM видно, что чем сильнее изменяется носитель, тем больше вероятность обнаружить разладку. Но так как расширение в случае второго распределения происходит в обе стороны, то алгоритму с Linear SVM становится сложнее найти центральную точку, и мощность уменьшается. Гибкость RBF в таком случае сильнейшим образом исправляет ситуацию, увеличивая мощность на 50%. Также в результате визуального анализа сэмплов было отмечено, что вокруг точки разладки в случае изменения лишь амплитуды колебаний (то есть изменения стандартного отклонения) сильной разницы может не

быть, в таком случае можно попробовать увеличить размер окна.

Как и ожидалось, Random Forest отработал лучше Decision Tree во всех случаях. Однако столь незначительное улучшение вряд ли стоит затраченных ресурсов (раздел 8.3).

8.3. RQ2: исследование относительной производительности реализованных алгоритмов

Для ограничения области применимости методов необходим анализ затрачиваемых ресурсов. В текущем разделе представлены результаты замеров времени работы алгоритмов на одном и том же датасете. Были сгенерированы три реализации временных рядов размером 1000 с разной частотой разладки, в которые вошли случайные сочетания распределений из таблицы 1. Каждый из алгоритмов был запущен 100 раз на каждом из сэмплов. Результаты представлены в таблице 4.

Alg.\CP	5	10	20
DT	0.7	0.7	0.7
L. SVM	0.8	0.8	0.8
7-NN CM	42	42	42
RF	94	94	94

Таблица 4: Среднее время работы алгоритмов в секундах

Корреляцию между частотой разладки и затраченным временем установить не удаётся.

Для данного эксперимента абсолютные величины не столь важны, поскольку реализация не оптимизирована для скорости вычислений; ключевыми являются лишь их соотношения. При сравнении алгоритмов на основе Decision Tree и Random Forest наблюдается увеличение времени работы примерно в 140 раз. Поэтому в одномерном случае использование Random Forest практически неоправданно — получается лишь незначительное улучшение качества обнаружения. Метод на основе KNN с комбинаторной метрикой оказался в 56 раз медленнее, чем Linear SVM, и в 63 раза медленнее, чем Decision Tree. Однако качество

обнаружения при этом также значительно повысилось по сравнению с другими двумя классификаторами, как видно из таблицы 3. Linear SVM работал в 1,1 раза медленнее, чем Decision Tree.

8.4. RQ3: сравнение с оригинальной реализацией метода обнаружения разладки на основе графа ближайших соседей на многомерных данных

Для оценки качества работы алгоритмов с многомерными данными и сравнения с оригинальной [3] реализацией метода обнаружения разладки на основе графа ближайших соседей был проведен вычислительный эксперимент на основе синтетических многомерных данных. Описание многомерных распределений можно найти в таблице 5. Распределение до точки разладки — многомерное стандартное нормальное, после — происходит сдвиг мат. ожидания на значение, указанное в таблице. Подсчитанные пороги для многомерных распределений можно увидеть в таблице 2. Выбор указанных значений сдвига и размерности обусловлен использованием этих же значений в работе [3], с которой происходило сравнение.

Обозначение	Сдвиг	Размерность
MN, MN^1	0.7	10
MN, MN^2	1.8	100
MN, MN^3	2.7	1000

Таблица 5: Многомерные нормальные распределения с единичной матрицей ковариаций

Вычисленные мощности можно увидеть в таблице 6. Результаты алгоритмов `pysatl-cpd` оказались гораздо лучше результатов реализации алгоритма обнаружения разладки на основе ближайших соседей из статьи. Оказалось, что для приближения к абсолютно точному обнаружению разладки достаточно взять $k = 7$. В статье для достижения мощности 0.8 в гораздо более очевидных ситуациях были взяты значения $k > 10$.

Distribution	CM	F ₁	MCC				
	7-NN	7-NN	7-NN	RF	DT	Lin. SVM	RBF SVM
$MN - MN^1$	0.994	0.225	0.841	0.799	0.349	0.757	0.976
$MN - MN^2$	1.0	1.0	1.0	1.0	0.986	1.0	1.0
$MN - MN^3$	1.0	1.0	1.0	1.0	0.999	1.0	1.0

Таблица 6: Мощность. Многомерные распределения

Если горизонтально просмотреть таблицу, можно увидеть, что KNN с комбинаторной метрикой имеет наилучшие результаты. Random Forest значительно улучшил результаты Decision Tree, RBF SVM улучшил результаты Linear SVM.

8.5. RQ4: валидация реализованных алгоритмов на реальных данных

Для валидации реализованных алгоритмов были использованы следующие реальные датасеты: данные геофизического каротажа (Well-Log), временные ряды индекса Dow Jones, статистика аварий на угольных шахтах. Эти наборы данных обладают множеством ключевых преимуществ: широкая воспроизводимость (использованы в большом количестве публикаций, включая [1, 7, 18]), разнообразие типов разладки, наличие эталонных результатов для сравнения. Таким образом, возникает возможность провести прямое сравнение с классическими методами (Байесовский подход, CUSUM) и оценить устойчивость к реальным шумам и артефактам.

8.5.1. Описание данных

Coal Mine Disasters — количество несчастных случаев, происходящих ежегодно в период с 1851 по 1962 год (Рис. 8). С 1887 года, после подписания закона о регуляции угольных шахт, несчастных случаев становится значительно меньше: происходит изменение распределения случайной величины, значением которой является

количество несчастных случаев, происшедших за указанный год. Случайная величина принимает всего 7 целочисленных значений.

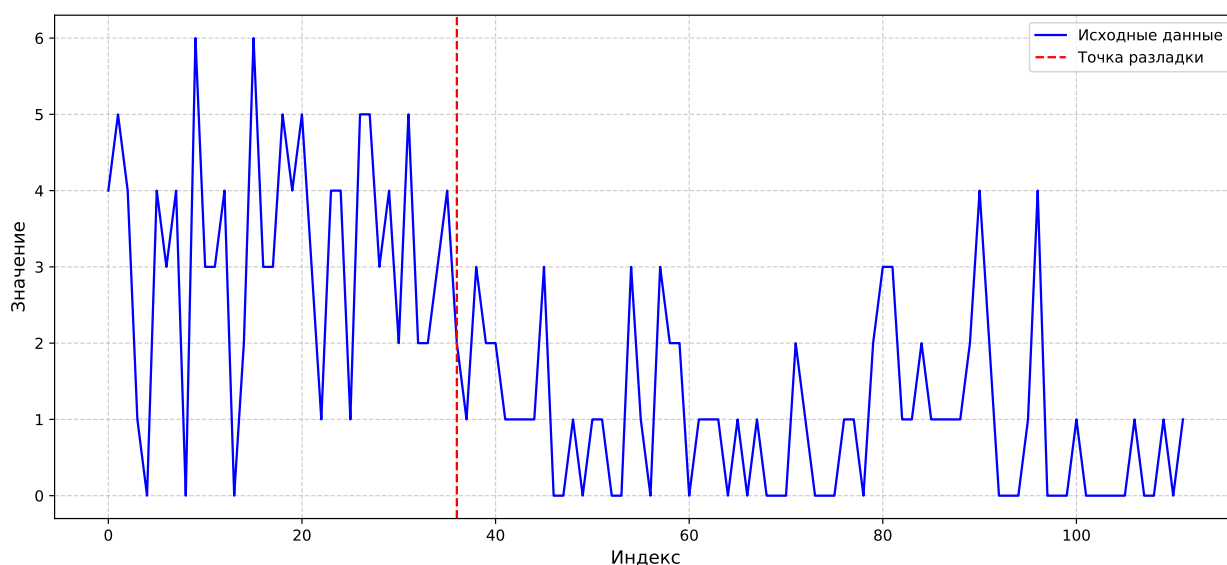


Рис. 8: Данные Coal Mine Disasters, красной линией отмечено время подписания закона о регуляции шахт

Dow Jones Industrial Average Daily Returns — ежедневная доходность индекса Dow Jones Industrial Average с 3 июля 1972 года по 30 июня 1975 года (Рис. 9). В этот период произошли 3 значительных политических события, в результате чего показатели претерпевали заметные изменения. Эти данные довольно трудны для нахождения разладки: в силу того, что они являются следствием социальных процессов, они содержат много шума. Помимо этого, в результате визуального анализа был сделан вывод, что изменения происходят скорее в отклонении случайной величины, нежели в её математическом ожидании, что, например, делает менее эффективным SVM.

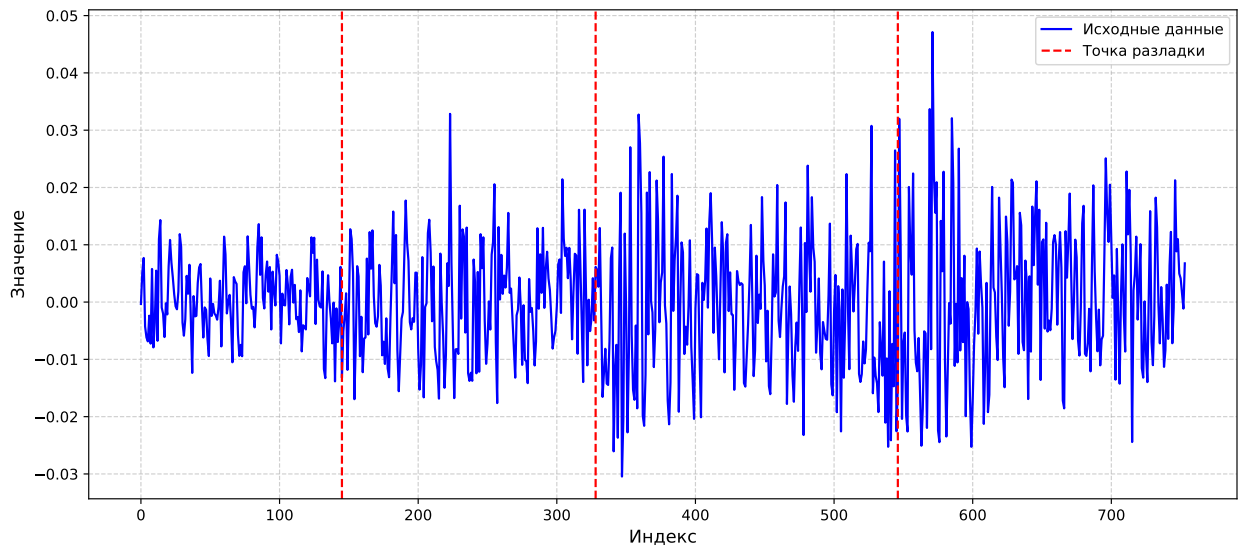


Рис. 9: Данные DJI, красными линиями отмечены наблюдения, соответствующие событиям, повлекшими за собой значительные изменения в поведении модели

Well-Log Data представляет собой измерения ядерно-магнитного отклика, полученные в процессе бурения скважины (Рис. 10). Этот датасет, несмотря на отсутствие точной разметки моментов разладки и привязки к конкретным технологическим событиям, обладает выраженными визуально идентифицируемыми скачками математического ожидания, что делает его ценным тестовым случаем для проверки чувствительности алгоритмов к резким изменениям среднего уровня сигнала. Характерные инструментальные шумы в данных позволяют оценить устойчивость алгоритмов обнаружения разладки к подобного рода помехам.

8.5.2. Результаты классификационных алгоритмов

Все реальные данные не являются размеченными, фундаментальные изменения, происходящие в них, — это реакции на события, произошедшие в определенной сфере. Реакция может быть запоздалой, вследствие чего в математической модели смена распределения может произойти не сразу.

Результаты работы алгоритмов, основанных на классификаторах 7-NN с комбинаторной метрикой и RBF SVM с MCC, на данных **Well Log**

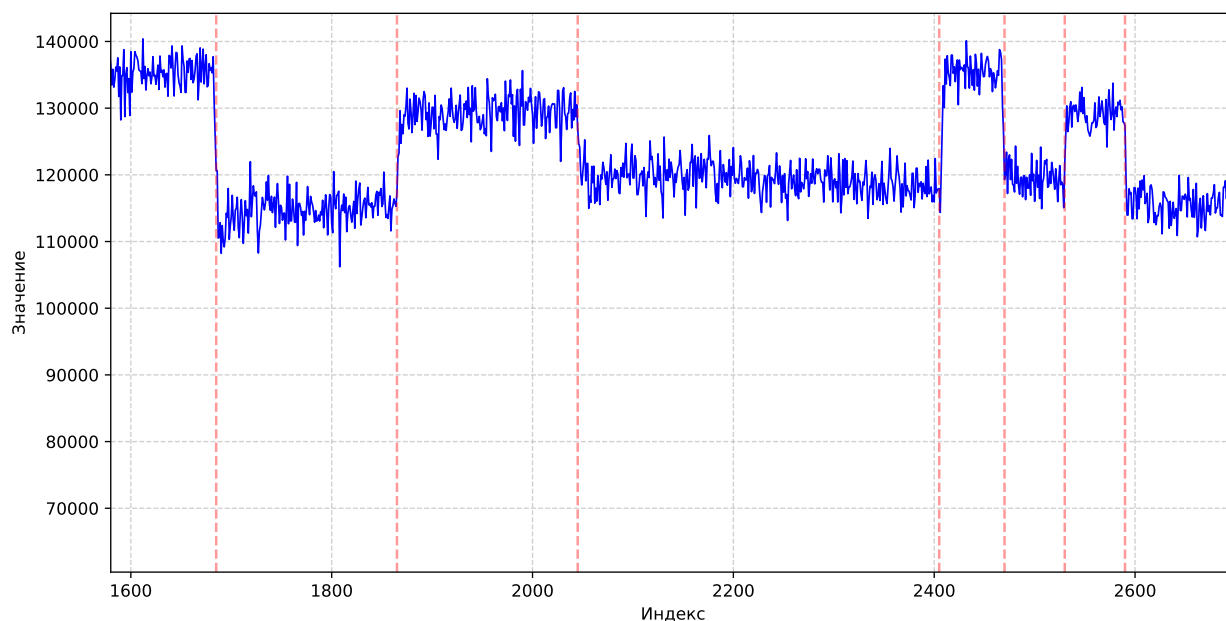


Рис. 10: Данные Well Log, красными линиями отмечены изменения среднего уровня сигнала

изображены на рис. 11 и 12 соответственно.

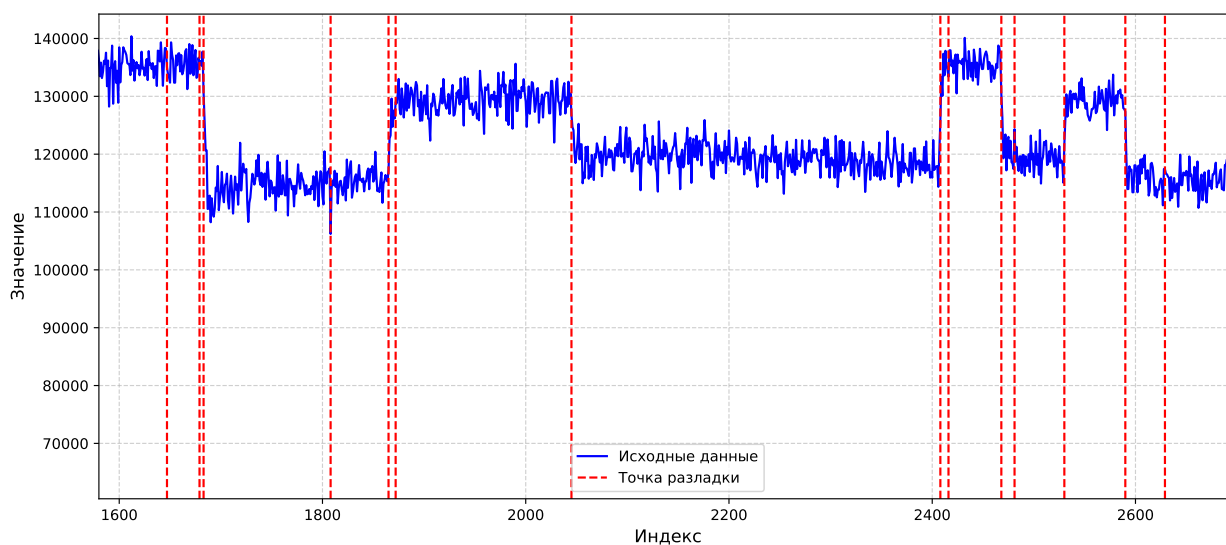


Рис. 11: Результаты алгоритма 7-NN CM на данных Well Log

В обоих случаях алгоритмы успешно справились с нахождением разладки с некоторой погрешностью (в данном случае решалась задача обнаружения разладки в рамках окна размером 64, следовательно, погрешность равна примерно половине размера окна). Однако почти каждое изменение распределения в данных сопровождалось нахождением двух точек разладки в результате работы алгоритмов. Это связано с

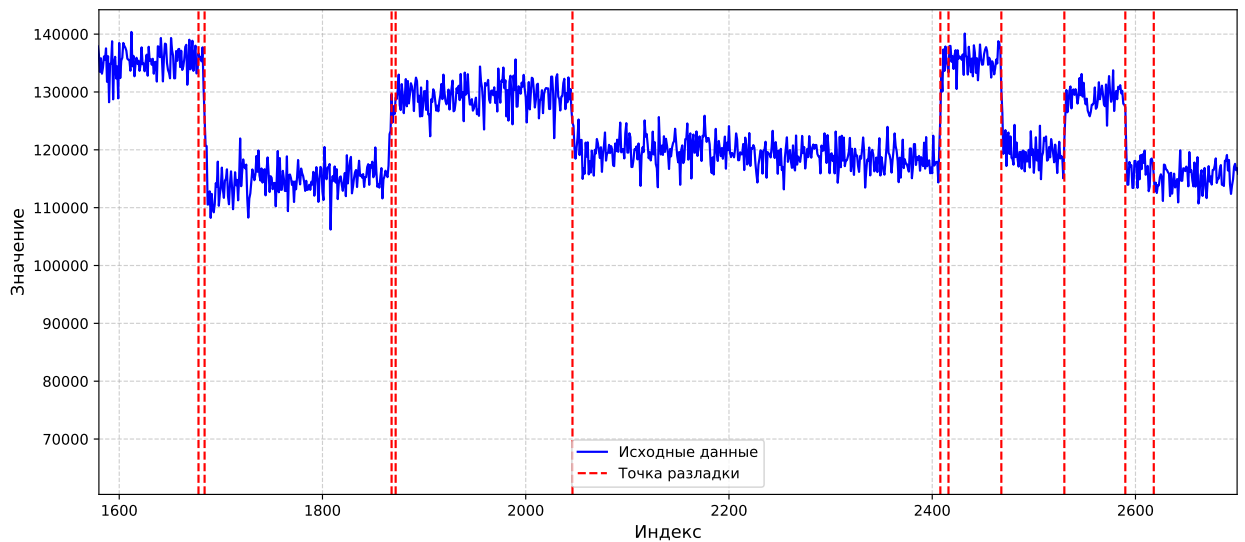


Рис. 12: Результаты алгоритма RBF SVM MCC на данных Well Log

тем, что два подряд идущих окна могут пересекаться в определённом количестве крайних точек, которых в некоторых случаях достаточно для того, чтобы алгоритм выдал положительный ответ о разладке. Из графиков видно, что RBF SVM более точно обнаружил большие сдвиги в математическом ожидании, чем это сделал 7-NN.

Работу алгоритмов на данных **DJI** (Рис. 13 и 14) более трудно анализировать, момент смены распределения на глаз точно определить невозможно. Всё, что известно — это время происшествия важных политических событий, которые повлекли за собой изменения показателей.

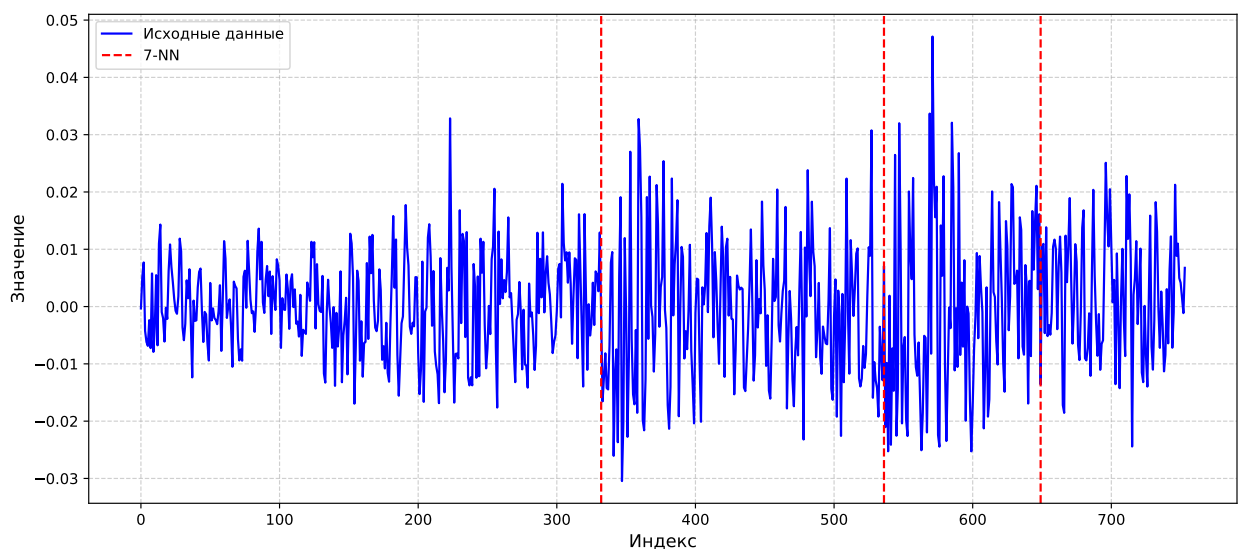


Рис. 13: Результаты алгоритма 7-NN CM на данных DJI

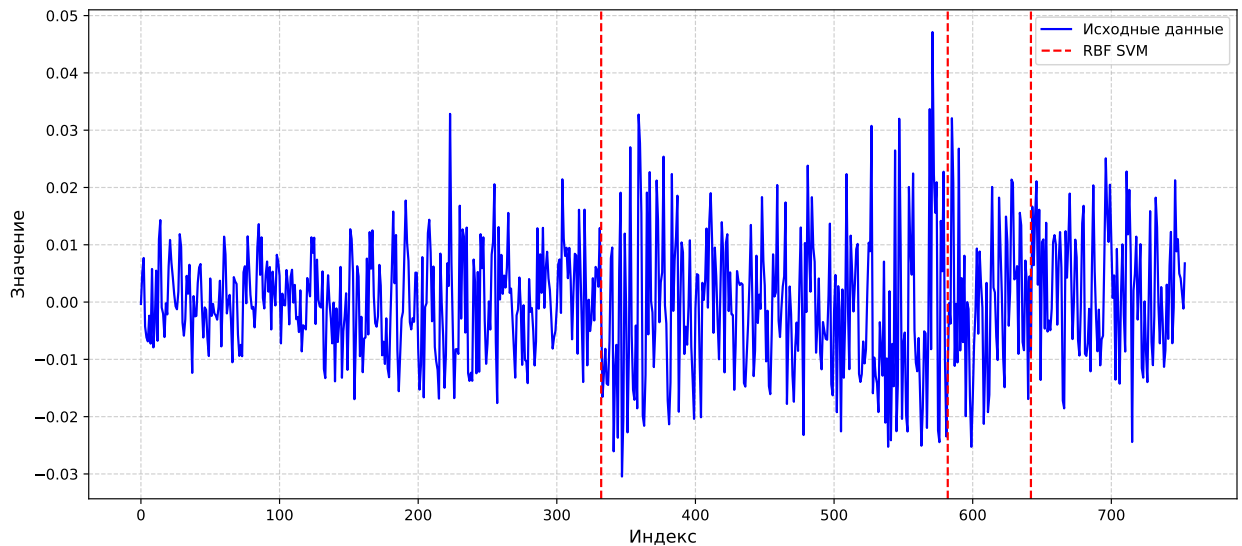


Рис. 14: Результаты алгоритма RBF SVM MCC на данных DJI

Алгоритмы не справились с обнаружением изменений, которые повлекло за собой первое событие, однако очень точно обнаружили второе. Что касается третьего, его обнаружили оба алгоритма, однако 7-NN сделал это с меньшей погрешностью. Оба алгоритма обнаружили еще одну разладку, что вполне обосновано: в промежутке от 600 до 700 заметно изменение отклонения, что является признаком разладки.

Обнаружить разладку на данных **Coal Mine** не составило труда для 7-NN и RBF SVM (Рис. 15).

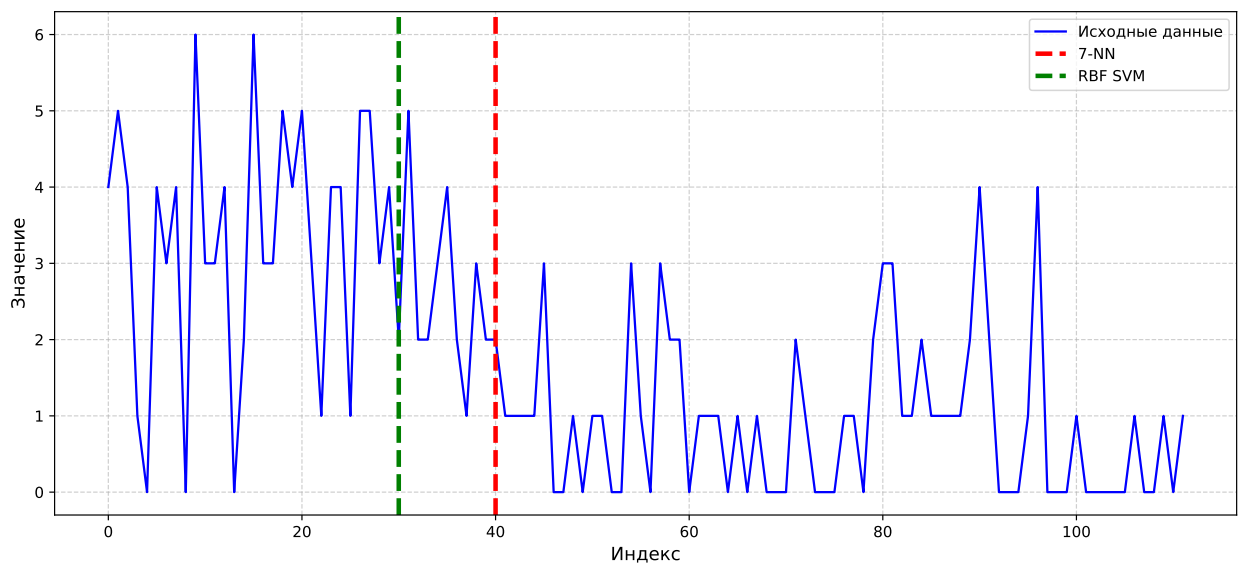


Рис. 15: Результаты алгоритмов 7-NN CM и RBF SVM MCC на данных Coal Mine

8.6. Замечание о скорости обнаружения разладки

Весь предыдущий анализ был посвящён решению задачи обнаружения разладки. В `pyratl-cpd` методы также могут решать проблему локализации точки разладки. В такой постановке задачи можно изучать множество других свойств методов, например, скорость обнаружения разладки. Определим это свойство следующим образом: скорость обнаружения разладки — это время между появлением точки разладки в датасете и её обнаружением. В онлайн случае у каждого наблюдения есть физическое время его появления во временном ряду. Интересны 4 значения времени: время t_{cp} появления точки разладки; время t_{fin} заполнения окна с точкой разладкой, то есть появление последней точки в этом окне; время $t_{startAlg}$ начала работы алгоритма с окном с точкой разладки; время t_{detect} обнаружения алгоритмом точки разладки.

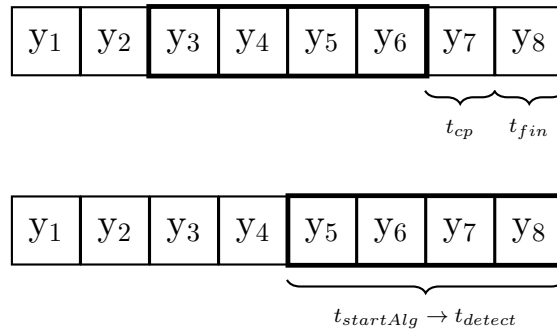


Рис. 16: Работа скрабера в онлайн режиме. Каждое наблюдение имеет физическое время появления во временном ряду

Функции для исследования:

$$f(cp) = t_{detect} - t_{cp} = (t_{detect} - t_{startAlg}) + (t_{startAlg} - t_{cp})$$

$$g(cp) = f(cp) - (t_{fin} - t_{cp})$$

Функция f показывает для определённой точки разладки время, прошедшее с момента её появления до момента её обнаружения. Функция g убирает время на накопление точек в окне. Она учитывает только время на работу алгоритма с текущим окном, и, что важно, время на завершение работы алгоритма с предыдущим окном, то есть время $t_{fin} \rightarrow t_{startAlg}$. С каждым новым окном эта задержка будет накапли-

ваться, откладывая анализ новых наблюдений. Соответственно, выбор стратегии скрабирования (параметры сдвиг, размер окна) должен зависеть от этой задержки. Если скорость поступающих данных изменится, скраберу необходимо будет подстроиться под неё. В рамках данной работы не предполагалось получение эмпирических результатов касательно скорости обнаружения разладки в онлайн-режиме. Исследование реализованных алгоритмов в онлайн-режиме является одним из направлений будущих работ.

8.7. Выводы по результатам экспериментального исследования

В представленном экспериментальном исследовании был проведен статистический бенчмаркинг реализованных алгоритмов на основе одномерных и многомерных синтетических датасетов и реальных данных, а также сравнение с существующими алгоритмами. Результаты экспериментов позволяют сделать следующие выводы.

7-NN CM с наибольшей вероятностью обнаруживал разладку на одномерных синтетических датасетах. RBF SVM MCC и Lin. SVM MCC имеют близкие к 7-NN CM показатели статистической мощности, однако в некоторых случаях сильно уступают. В основном, их проигрыш наблюдается на датасетах, где ключевым изменением является смена дисперсии или незначительная смена носителя. То есть ситуациях, когда возникает трудность в выборе центральной точки для разделения на классы. Однако отмечены случаи, в которых RBF SVM MCC имел лучшие показатели. Например, в одной из ситуаций с значительным расширением носителя равномерного распределения мощность RBF SVM MCC составляла примерно 0.92, в то время как мощность 7-NN CM была примерно 0.9.

На одномерных распределениях Random Forest и Decision Tree показали низкие результаты, справившись успешно лишь в самых очевидных ситуациях. В результате чего можно сделать вывод, что в зафиксированной в данной работе конфигурации они оказались малочувстви-

тельными.

Алгоритмы 7-NN CM и SVM RBF MCC почти идеально справились с анализом многомерных распределений: их показатели были в пределах от 0.994 до 1.0 и от 0.976 до 1.0 соответственно. Остальные алгоритмы продемонстрировали отличные результаты при сдвиге 1.8 и 2.7, однако гораздо хуже справились со сдвигом 0.7.

Итого, на синтетических датасетах 7-NN CM и RBF SVM MCC имели лучшие результаты. При сравнении этих двух алгоритмов важным фактом является то, что время работы 7-NN CM в 56 раз больше. При выборе алгоритма для конкретной ситуации, стоит учитывать баланс между эффективностью алгоритма и его производительностью. Например, в офлайн-случае время работы не имеет большого значения, однако в онлайн-случае время работы является одним из важнейших факторов.

Однако результаты на синтетических данных не гарантируют точно такого же успеха на реальных. RBF SVM MCC показал лучшие результаты в сравнении с 7-NN CM на Well Log, где требовалось обнаружить сильный сдвиг в математическом ожидании: при одинаково качественной локализации сдвигов RBF SVM MCC был более устойчив к шумам, чем 7-NN CM. На DJI и Coal Mine 7-NN и RBF SVM справились почти одинаково (7-NN в некоторых случаях обнаруживал разладку несколько раньше, чем это делал RBF SVM). Однако каковы бы ни были относительные показатели, оба алгоритма успешно прошли валидацию на указанных реальных данных, отреагировав почти на все изменения, связанные со значительными событиями.

Исследование различных метрик для классификатора 7-NN указывает на то, что аналогичный подход стоит применить и к другим методам. Выбор подходящей метрики может значительно улучшить результаты: например, в многих случаях SVM с MCC-метрикой превосходит 7-NN с MCC (Таблицы 3 и 6). Таким образом, SVM с ядром в сочетании с удачной метрикой качества может стать достойной альтернативой 7-NN с комбинаторной метрикой.

9. Заключение

В рамках представленной выпускной квалификационной работы было проведено исследование классификационных методов обнаружения разладки, основанных на барьерной технике. Основной целью являлась разработка и анализ обобщенного подхода, позволяющего гибко использовать различные классификаторы и метрики качества для решения этой задачи.

Ключевыми результатами работы стали:

- Разработка обобщенного классификационного барьерного метода обнаружения разладки.
- Реализация на его основе семейства алгоритмов с использованием классификаторов KNN, Kernel SVM (с линейным ядром и RBF), Decision Tree и Random Forest и метрик F_1 , MCC.
- Реализация метода обнаружения разладки с использованием KNN-графа [3].
- Создание и внедрение специализированного фреймворка для бенчмаркинга, позволившего провести всестороннее экспериментальное исследование реализованных алгоритмов.

Проведенное экспериментальное исследование позволило ответить на поставленные исследовательские вопросы:

- На синтетических одномерных данных (RQ1) наилучшую статистическую мощность продемонстрировали алгоритмы 7-NN с комбинаторной метрикой (7-NN CM) и SVM с ядром RBF и метрикой MCC (RBF SVM MCC).
- Анализ производительности (RQ2) показал, что алгоритмы на основе Decision Tree и Linear SVM являются наиболее быстрыми, в то время как 7-NN CM и Random Forest требуют значительно больших вычислительных затрат.

- При работе с многомерными данными (RQ3) реализованные алгоритмы, в частности 7-NN CM и RBF SVM MCC, показали высокую эффективность. Реализация метода на основе KNN-графа в рамках проекта `pysatl-cpd` продемонстрировала существенно лучшие результаты по сравнению с оригинальной реализацией из статьи [3].
- Валидация на реальных данных (RQ4) – Well-Log, Dow Jones Index и статистика аварий на угольных шахтах – подтвердила эффективность разработанных подходов. Алгоритмы RBF SVM MCC и 7-NN CM успешно локализовали большинство предполагаемых точек разладки, при этом RBF SVM MCC часто показывал лучшую устойчивость к шумам, а 7-NN CM – более точную локализацию в некоторых случаях. Было отмечено, что Kernel SVM, обладая сопоставимой эффективностью с методом на основе графа ближайших соседей из [3] в большинстве сценариев, работает значительно быстрее (до 60 раз).

Таким образом, представленная работа демонстрирует потенциал классификационных барьерных методов для задачи обнаружения разладки. Разработанный подход и реализованные алгоритмы предлагают гибкие и эффективные инструменты, применимые как в офлайн, так и (с использованием техники скрабирования) в онлайн сценариях. Выбор конкретного алгоритма и его параметров должен основываться на балансе между требуемой точностью обнаружения и доступными вычислительными ресурсами.

Код с реализациями алгоритмов находится в ветке `main` репозитория `pysatl-cpd`¹⁰. Код с реализацией прототипа фреймворка для бенчмаркинга находится в ветке `scrubbing-methods-bench`¹¹. Имя пользователя: `artemiipatov`.

Результаты данной работы были представлены на конференции

¹⁰`pysatl-cpd`: <https://github.com/PySATL/pysatl-cpd> (Дата обращения: 24.05.2025)

¹¹Фреймворк для бенчмаркинга: <https://github.com/PySATL/pysatl-cpd/tree/scrubbing-methods-bench> (Дата обращения: 24.05.2025)

FRUCT 2025¹² [15].

В работе были выполнены все поставленные задачи, однако остается еще множество открытых направлений исследования:

- Экспериментальное исследование реализованных алгоритмов на основе онлайн-данных в том числе при помощи методологии, сформулированной в разделе 8.6.
- Комбинирование различных классификационных моделей, применение предобработки.

¹²FRUCT 2025: <https://fruct.org/publications/volume-37/fruct37/> (Дата обращения: 24.05.2025)

Список литературы

- [1] Adams Ryan Prescott, MacKay David J.C. Bayesian Online Change-point Detection // [arXiv preprint](#). — 2007. — 0710.3742.
- [2] Aminikhanghahi Samaneh, Cook Diane J. A Survey of Methods for Time Series Change Point Detection // [Knowledge and Information Systems](#). — 2017. — Vol. 51.
- [3] Chen Hao. Sequential change-point detection based on nearest neighbors // [The Annals of Statistics](#). — 2019. — Vol. 47, no. 3. — P. 1381–1407.
- [4] Chen Hao, Zhang Nancy. Graph-based change-point detection // [The Annals of Statistics](#). — 2015. — Vol. 43, no. 1. — P. 139–176.
- [5] Chicco Davide. Ten quick tips for machine learning in computational biology // [BioData Mining](#). — 2017. — Vol. 10.
- [6] Chicco Davide, Jurman Giuseppe. The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification // [BioData Mining](#). — 2023. — Vol. 16.
- [7] Fearnhead Paul, Clifford Peter. On-Line Inference for Hidden Markov Models via Particle Filters // [Journal of the Royal Statistical Society: Series B](#). — 2003. — Vol. 65, no. 4. — P. 887–899.
- [8] Iwata Takuma et al. [Accelerating Online Change-Point Detection Algorithm Using 10 GbE FPGA NIC](#) // Euro-Par 2018: Parallel Processing Workshops. — 2019. — P. 506–517.
- [9] Jiang Xiaodong et al. Kats: Time Series Analysis Library. — GitHub repository. — 2022. — URL: <https://github.com/facebookresearch/Kats>.
- [10] Lawhern Vernon, Kerick Scott, Robbins Kay. Detecting alpha spindle events in EEG time series using adaptive autoregressive models // [BMC Neuroscience](#). — 2013. — Vol. 14. — P. 101.

- [11] Li Yuancheng, Qiu Rixuan, Jing Sitong. Intrusion detection system using Online Sequence Extreme Learning Machine (OS-ELM) in advanced metering infrastructure of smart grid // [PLOS ONE](#). — 2018. — Vol. 13, no. 2. — P. e0192216.
- [12] Londschien Malte, Bühlmann Peter, Kovács Solt. Random Forests for Change Point Detection // [arXiv preprint](#). — 2022.
- [13] Madrid Padilla Oscar Hernan et al. Optimal Nonparametric Multivariate Change Point Detection and Localization // [IEEE Transactions on Information Theory](#). — 2019. — Vol. 68. — P. 1922–1944.
- [14] Mondal Pronoy K., Biswas Munmun, Ghosh Anil K. On high dimensional two-sample tests based on nearest neighbors // [Journal of Multivariate Analysis](#). — 2015. — Vol. 141. — P. 168–178.
- [15] Patov Artemii, Gorikhovskii Viacheslav, Kutuev Vladimir. Classification-Based Barrier Change Point Detection Methods // [Proceedings of the 37th Conference of FRUCT Association](#). — 2025.
- [16] Pepelyshev Andrey, Polunchenko Aleksey S. Real-time financial surveillance via quickest change-point detection methods. — 2015. — 1509.01570.
- [17] Rauhameri Anton et al. A comparison of online methods for change point detection in ion-mobility spectrometry data // [Array](#). — 2022. — Vol. 14. — P. 100151.
- [18] Riedel Kurt. Numerical Bayesian Methods Applied to Signal Processing // [Technometrics](#). — 2012. — Vol. 39.
- [19] Schilling Mark F. Multivariate Two-Sample Tests Based on Nearest Neighbors // [Journal of the American Statistical Association](#). — 1986. — Vol. 81, no. 395. — P. 799–806.
- [20] Sokolova Marina, Japkowicz Nathalie, Szpakowicz Stan. [Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Per-](#)

- formance Evaluation // AI 2006: Advances in Artificial Intelligence. — Vol. 4304. — 2006. — P. 1015–1021.
- [21] Takayasu Hideki. [Basic methods of change-point detection of financial fluctuations](#) // IEEE International Conference on Noise and Fluctuations (ICNF). — 2015. — P. 1–3.
 - [22] Takeuchi J., Yamanishi K. A unifying framework for detecting outliers and change points from time series // [IEEE Transactions on Knowledge and Data Engineering](#). — 2006. — Vol. 18, no. 4. — P. 482–492.
 - [23] Tharwat Alaa. Classification Assessment Methods // [Applied Computing and Informatics](#). — 2018.
 - [24] Truong Charles, Oudre Laurent, Vayatis Nicolas. Selective review of offline change point detection methods // [Signal Processing](#). — 2019. — Vol. 167. — P. 107299.
 - [25] Wadinger Marek, Kvasnica Michal, Kawahara Yoshinobu. Change-Point Detection in Industrial Data Streams based on Online Dynamic Mode Decomposition with Control // [arXiv preprint](#). — 2024.
 - [26] Yang Wan, Lipsitch Marc, Shaman Jeffrey. Inference of seasonal and pandemic influenza transmission dynamics // [Proceedings of the National Academy of Sciences](#). — 2015. — Vol. 112, no. 9. — P. 2723–2728.
 - [27] Zhang Hao et al. Online Dynamic Mode Decomposition for Time-Varying Systems // [SIAM Journal on Applied Dynamical Systems](#). — 2019. — Vol. 18, no. 3. — P. 1586–1609.
 - [28] Zhang Yuxuan, Chen Hao. Graph-based multiple change-point detection. — 2021. — URL: <https://api.semanticscholar.org/CorpusID:238259132>.

А. Приложение: Архитектура фреймворка для бенчмаркинга

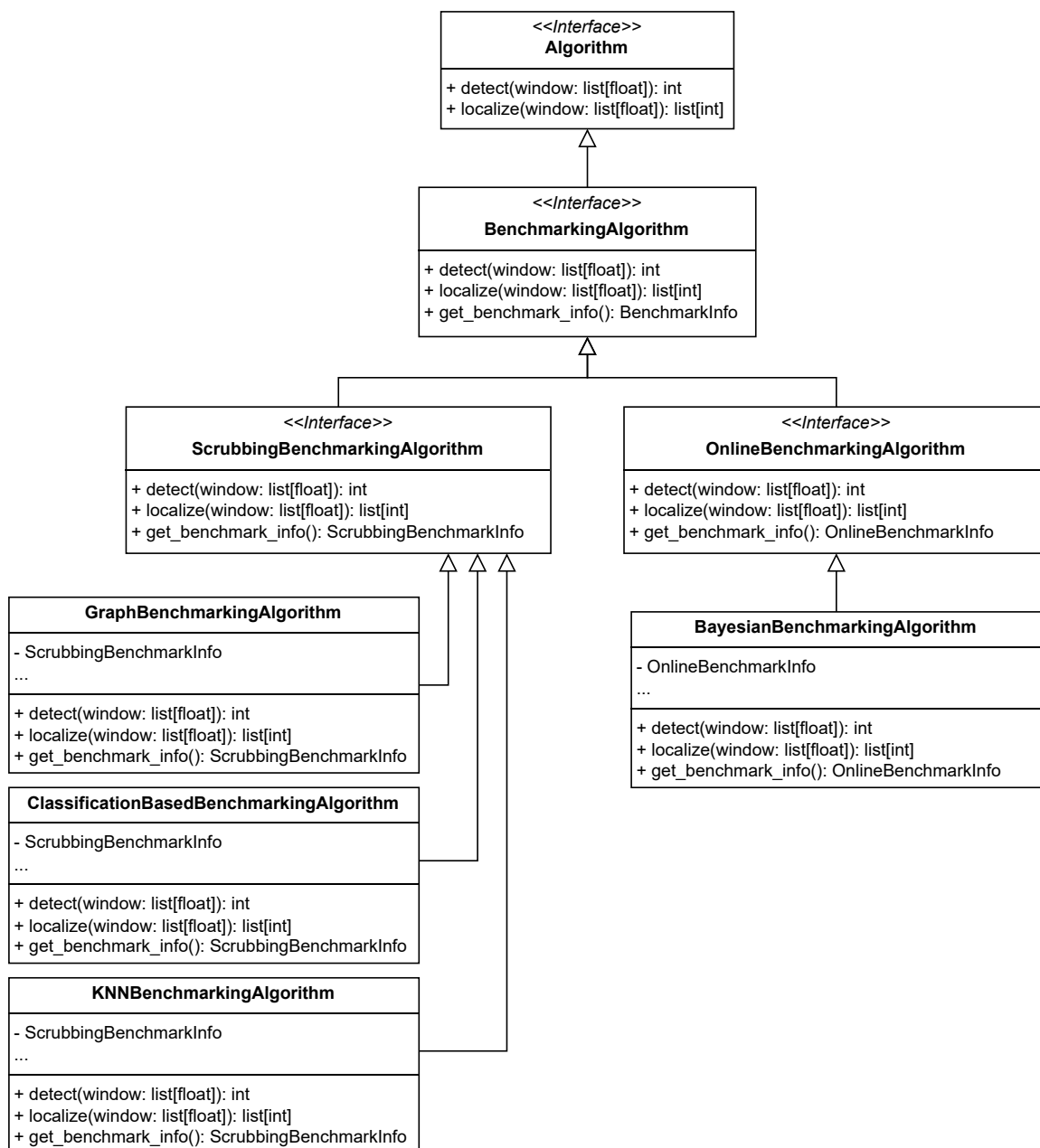


Рис. 17: Архитектура алгоритмов для бенчмаркинга

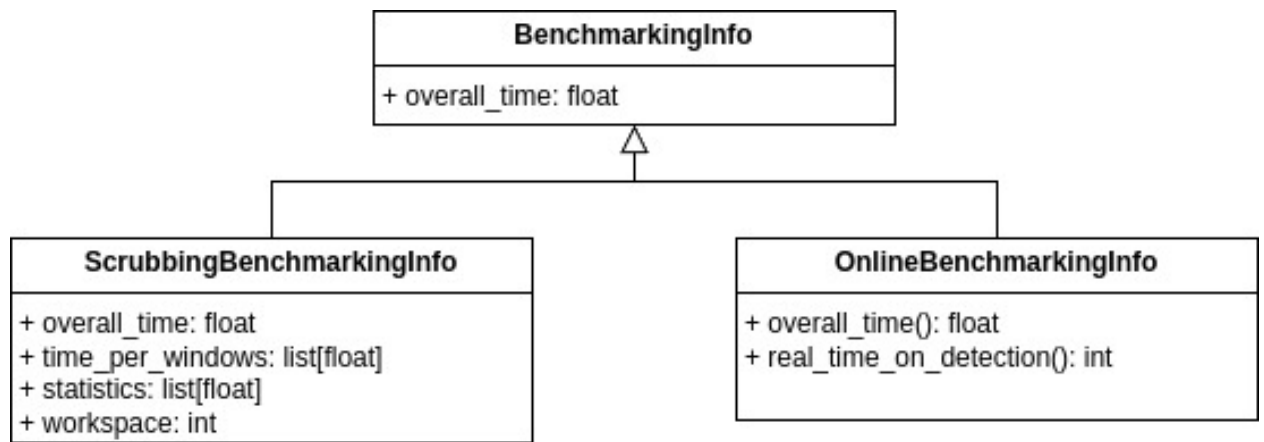


Рис. 18: Архитектура артефактов бенчмаркинга

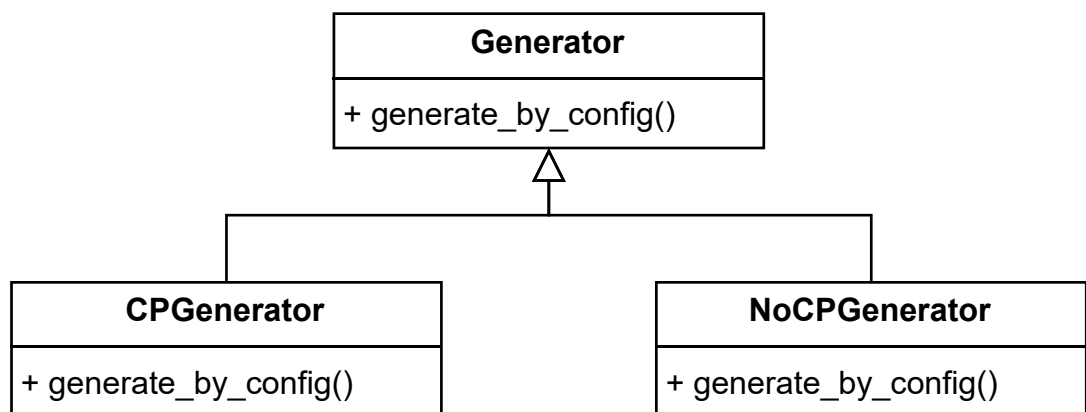


Рис. 19: Генератор данных по заданному распределению

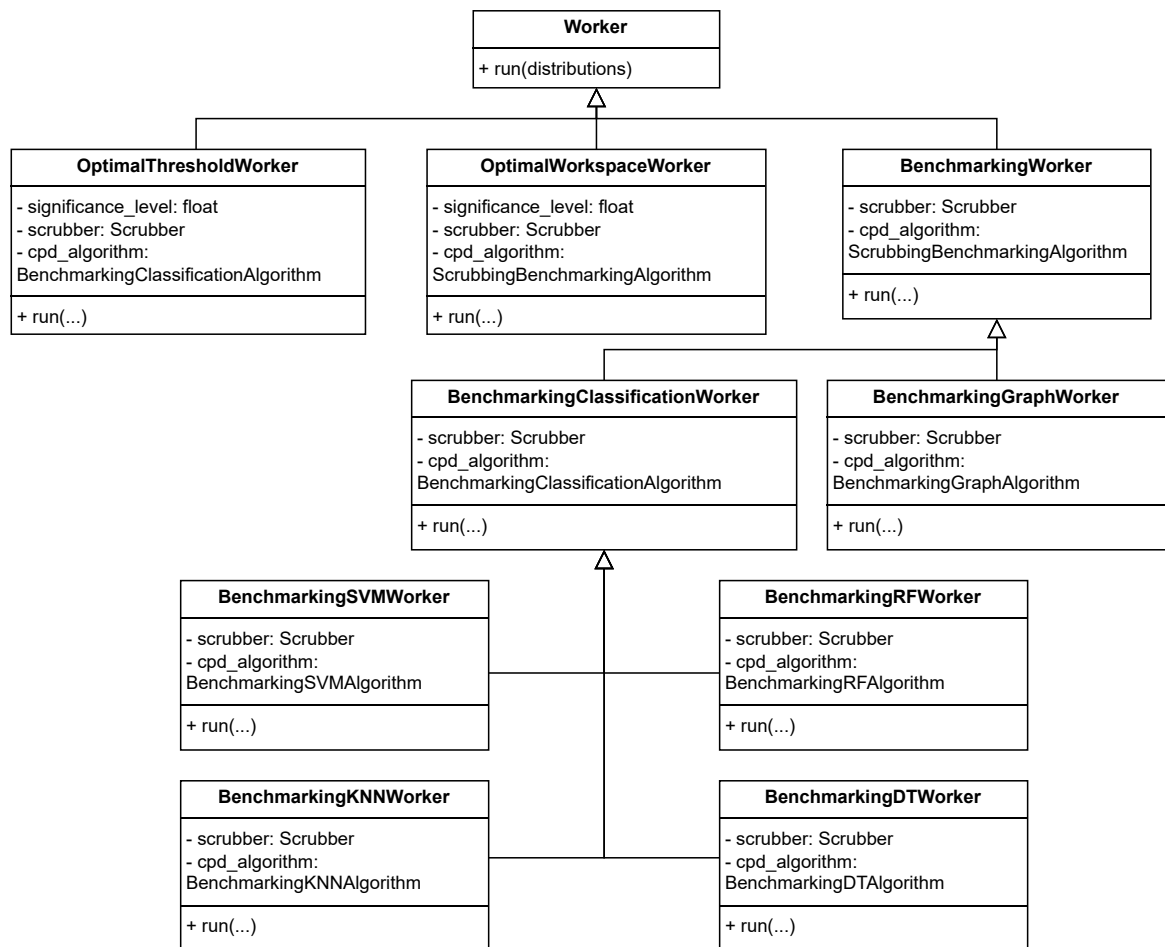


Рис. 20: Архитектура воркера

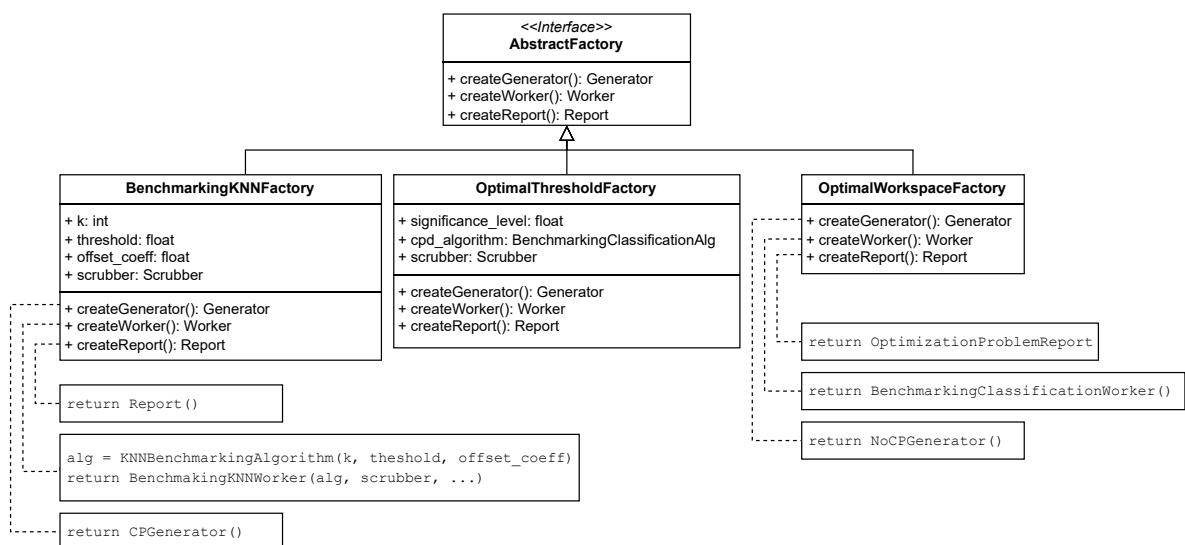


Рис. 21: Абстрактная фабрика