Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Адаптация и исследование байесовского подхода к обнаружению разладки для PySATL-CPD-Module

ТАТЬЯНЕНКО Алексей Дмитриевич

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2024

Оглавление

Введение

Задача обнаружения разладки (англ. «change point detection») заключается в определении точки временного ряда, в которой одно распределение сменилось другим. Это бывает необходимо для определения смены состояний наблюдаемой системы [20, 22], для разделения данных на однородные сегменты [9, 11] и других целей [2]. Однако на данный момент не существует оптимальных алгоритмов, поскольку они могут накладывать различные ограничения на данные, требовать обучения на размеченных данных, не позволять определять разладку на ходу, иметь ограниченную точность и т.д. [2] Другая проблема заключается в том, что существующие решения зачастую не подготовлены к индустриальному использованию и являются скорее исследовательскими прототипами, а те немногие инструменты, в которых собрано несколько алгоритмов, охватывают лишь малую их часть, а также порой написаны на специфичных языках (например, на популярном среди статистиков R) [15], что препятствует более широкому их использованию.

Для решения этой проблемы был создан PySATL-CPD-Module. Это Pyhton-библиотека, целью которой является реализация и объединение под общим интерфейсом разнообразных алгоритмов обнаружения разладки, ранее описанных в научных статьях, исследование их характеристик на синтетических и реальных данных, их развитие и автоматизация экспериментального пайплайна. Важно заметить, что на данный момент основное внимание уделяется исследованию теоретических свойств, а не оптимизации быстродействия. В рамках пакета вводится разделение на задачи детекции (определения наличия) и локализации (определения местоположения) разладки, а также на оффлайн- (алгоритму доступно лишь некоторое окно данных, внутри которого и проводится анализ) и оффлайн-режимы (окно двигается по данным и мы оцениваем результат работы алгоритма в совокупности).

Одним из важных методов обнаружения разладки является байесовский [1], основанный на анализе распределений длин пробега (т.е. времён, прошедших с последней разладки). Предложенный в оригинальной ста-

тье подход активно развивается в различных работах, а алгоритм имеет много реализаций, однако одним из камней преткновения является отсутствие формального критерия определения разладки. На эту тему не было найдено исследований, а в реализациях разладка выбирается различными способами (например, по порогу, по некоторому окну или по наибольшей вероятности). Для конечного пользователя это является проблемой, потому что без понимания устройства алгоритма принятие хорошего решения окажется затруднительным. Именно интеграции адаптированного байесовского алгоритма, а также исследованию базового порогового критерия и посвящена эта работа.

Особенностью байесовского алгоритма является его сильная зависимость от выбора предсказательной модели и априорных значений параметров. В данной работе рассматривается предсказательная модель для одномерного нормального распределения с обучением априорных параметров. Такая модель гарантирует оптимальные результаты для гауссовского распределения, тогда как её применимость для остальных остаётся под вопросом. Поэтому вводятся понятия штатного и нештатного режимов. Штатный заключается в переходе с нормального распределения на любое другое, а нештатному соответствуют все остальные случаи. Также появляется вопрос поведения алгоритма при переключении с некоторого распределения на нормальное, поскольку это позволит говорить о его применимости в условиях, когда переходы происходят поочерёдно между нормальным и некоторым другим распределениями.

По результатам исследований были выбраны значения порогов в соответствии с уровнями значимости и проведено вычисление мощности для них. Мощность детекции разладки оказалась близкой к 1.0, что является крайне хорошим результатом, а мощность локализации варьируется в зависимости от распределений: нормальное, равномерное и бета- показывают себя хорошо, тогда как экспоненциальное и Вейбулла — хуже. Тем не менее, алгоритм хорошо работает в штатном режиме и при выборе низких значений порога показывает приемлемые результаты для нештатного режима. Анализ плохих случаев указал на предпочтительность учёта в детекторах разладки толерантности к небольшим

падениям длины пробега и корректной обработки падений её значения за 2 этапа. Это даёт основания для предложения более точных критериев в ходе дальнейших исследований.

1. Обзор

В данном разделе приводятся постановка задачи об обнаружении разладки и контекст её появления на практике. Далее кратко описываются различные алгоритмы её решения, существующие классификации. Затем внимание сосредотачивается на байесовском алгоритме обнаружения разладки как основном предмете работы. Рассматриваются существующие реализации алгоритма и их недостатки. В конце проводится краткий экскурс в проект PySATL-CPD-Module.

1.1. Задача обнаружения разладки

Задача обнаружения разладки заключается в поиске точки временного ряда, где одна выборка сменяется другой, отличной он неё (в частности, это подразумевает изменение распределения).

$$
\begin{aligned}
&\{y_i\}, i = 1, \ldots, n \\
&H_0 \colon y_i \sim F_0, i = 1, \ldots, n \\
&H_1 \colon \exists \tau : 1 \leq \tau < n, y_i \sim \begin{cases} F_1, & i > \tau, \\ F_0 & \text{иначе.} \end{cases}
\end{aligned}
$$

Тогда τ — точка разладки. Задача обнаружения разладки встречается в различных областях: финансы, робототехника и др. [2] При этом контекст её возникновения можно условно разделить на две группы случаев:

- анализ накопленных данных с целью получения некоторой новой информации (т.н. «data mining»);
- детектирование изменений состояния в процессе работы (например, анализ температуры и частоты работы процессора при бенчмаркинге).

Также есть смысл рассматривать обнаружение разладки как два различные задачи: определения её наличия (детектирование разладки)

и определение её местоположения (локализация разладки). В силу статистической природы задачи во втором случае может потребоваться рассмотрение не точного соответствия, а некоторого допустимого отклонения от истинной точки при локализации.

1.2. Алгоритмы обнаружения разладки

Существуют различные методы и алгоритмы обнаружения разладки, классифицируемые в зависимости от интересующего нас свойства. Например, онлайн-алгоритмы позволяют засекать разладку (обычно с некоторой задержкой) по мере поступления новых данных, а оффлайн-алгоритмы способны анализировать лишь уже набранные данные; работа параметрических алгоритмов зависит от выбранных значений параметров, в отличие от непараметрических. [2] Обе классификации можно рассматривать не как бинарные, а как спектр: в первом случае оценивается задержка по количеству шагов, после которой возможно обнаружить разладку, а во втором — мощность или другие количественные характеристики для оценки устойчивости алгоритма.

В частности, графовый алгоритм представляют измерения как граф, у которого анализируются пересечения рёбер и прямой, отмечающей момент времени. Алгоритмы, основанные на отношении плотностей (uLSIF, RuLSIF, KLIEP), рассматривают две выборки, разделённые точкой, и анализируют отношения их плотностей. Алгоритмы, использующие классическое машинное обучение, строят классификаторы на основе метода k ближайших соседей, решающих деревьев и прочего. Для вероятностных алгоритмов важными оказываются идеи, заложенные байесовским подходом, о котором пойдёт речь ниже.

1.3. Байесовский онлайн-метод

Байесовский онлайн-метод обнаружения разладки [1] основан на анализе распределений так называемых длин пробега (длина пробега — время, прошедшее с предполагаемой последней разладки) с учётом априорных параметров и получаемых наблюдений. Он требует задания:

- функции выживаемости, вычисляющей вероятность разладки с учётом всех предыдущих наблюдений;

- функции правдоподобия, вычисляющей вероятность получения текущего наблюдения с учётом прошлых наблюдений и распределения длин пробега.

Помимо блока вычисления распределений длин пробега требуется блок их анализа на предмет наличия разладки и, возможно, определения её положения. В статье [1], описывающей этот алгоритм, явные критерии не приводятся, поэтому их выбор остаётся предметом для исследования.

Байесовский онлайн-метод задаёт общий подход к задаче обнаружения разладки, оставляя простор для различных модификаций и оптимизаций. В рамках этого подхода продолжают появляться и развиваться новые алгоритмы, использующие похожие идеи, но отличающиеся в тех или иных аспектах, например:

- Расширенный байесовский алгоритм, уменьшающий вычислительную сложность, снимающий часть ограничений на входные данные и предлагающий некоторые аппроксимации [11].

- Адаптивный байесовский алгоритм, использующий гауссовские процессы в качестве предсказательной модели вместо функций правдоподобия с сопряжёнными априорными распределениями [18].

- Байесовский алгоритм, адаптирующийся под дисперсию и тренды [21].

Байесовский подход имеет ряд выделяющих его в лучшую сторону свойств: квадратичная (или линейная при некоторых оптимизациях и аппроксимация) сложность вычисления, онлайн-природа, позволяющая в «жадном» режиме обрабатывать каждое новое значение и сразу же пробовать искать разладку, отсутствие необходимости в обучении с наблюдателем. [2] К его минусам можно отнести сильную зависимость

базовой версии от выбираемой параметризации (функции выживаемости и функции правдоподобия, задающей предсказательную модель, априорные параметры) и предположение о том, что исходные данные являются выборкой на отрезках без разладки. На борьбу с этими недостатками и направлены различные модификации базового алгоритма.

В представленном виде метод предполагает, что мы сразу указываем априорные гиперпараметры функции правдоподобия перед началом работы исходя из некоторых представлений об области. Однако для общности инструмента предпочтительно иметь возможность оценить их из некоторой обучающей выборки и уже с учётом этих данных начинать строить распределения длин пробега. Это затруднительно в случае итеративного поступления данных, поскольку до окончания этапа обучения мы не сможем рассматривать распределения длин пробега. Если же данные поступают для обработки сразу некоторым окном, то мы можем выделить полноценный этап обучения, причём проводить его не только в начале, но и после обнаруженной разладки. При этом предполагается, что данные до и после разладки друг от друга не зависят, а слишком близкие разладки разделить будет в любом случае затруднительно в силу нестабильности алгоритма ближе к концам отрезка, особенно в начале.

Байесовский подход к задаче обнаружения разладки применяется в различных областях: байесовский вывод в физике [20], анализ туристического потока [22], сегментация мозговой активности при эпилепсии [11] и т.д. Общий характер постановки задачи и общая формулировка подхода к её решению способствует широкому спектру возможных применений байесовских алгоритмов.

1.4. Существующие инструменты

Существуют различные инструменты для обнаружения разладок, в основном на языках Python и R (второй более популярен для реализации статистических алгоритмов, нежели в других областях). Недостатком R является его нишевость, из-за чего встроенное применение

обычно будет требовать интеграции кода на R в проекты на других языках программирования. Это особенно существенно в случаях, когда есть необходимость не просто анализировать накопленные данные, но ещё и использовать полученные результаты на ходу (например, переключение режима работы после разладки). Также предпочтительной является как можно более высокая модульность реализации, позволяющая настраивать пользователю алгоритм самостоятельно.

Реализации байесовского метода, описанного выше, не удалось найти в составе крупных инструментов с набором разнообразных алгоритмов и общим интерфейсом. Тем не менее, существуют инструменты, основанные на байесовском подходе в целом:

- **PyMC** позволяет обнаруживать разладку лишь с одного указанного распределения на другое, а потому сильно зависит от наших исходных знаний о предметной области, без которых применение пакета становится затруднительным;
- **RBeast** предоставляет широкую функциональность для анализа получаемых данных, но не нацелен непосредственно на автоматическое обнаружение разладок, делая вместо этого упор визуальный анализ различных статистик.

Ряд существующих реализаций алгоритма [3–8, 16, 23], большинство из которых были рассмотрены в прошлой работе, обладает следующими недостатками:

- отсутствие разрешительной лицензии;
- низкая конфигурируемость (например, функция правдоподобия вычисляется прямо в коде основного алгоритма);
- отсутствие явного критерия определения положения разладки;
- наличие лишь одного алгоритма без анализа качества его работы и специфики применимости, что снижает универсальность такого инструмента.

Отдельного упоминания заслуживает пакет для байесовского метода обнаружения разладки на Rust с Python-обёрткой [24]. В нём реализованы базовый алгоритм и его модификация (адаптивный байесовский алгоритм) с предсказательной моделью, основанной на гауссовских процессах.

Все перечисленные выше реализации объединяет общий недостаток: в них отсутствует явный критерий обнаружения разладки, вместо чего людям предлагается либо самостоятельно выбирать некоторые пороги и численно анализировать результаты работы алгоритма, либо оценивать вычисленные статистики визуально, строя графики. Это связано с тем, что чёткий критерий отсутствует и в исходной статье [1] с описанием байесовского метода обнаружения разладки.

2. Постановка задачи

Целью работы является адаптация и интеграция байесовского онлайн-алгоритма обнаружения разладки в проект PySATL-CPD-Module, а также его численное исследование. Для выполнения этой цели были поставлены следующие задачи:

1. Адаптация алгоритма для использования в оффлайн-режиме.
2. Интеграция адаптированного алгоритма в PySATL-CPD-Module.
3. Проведение численного исследования алгоритма для получения порогов детектора и оценки мощности для разных уровней значимости.
4. Анализ полученных результатов экспериментов в штатном и нештатном режимах для оценки применимости в онлайн-форме и оптимизации детектирования.

3. Метод

Центральным для байесовского метода обнаружения разладки является понятие длины пробега, под которой подразумевается время (количество шагов), прошедшее с последней разладки. Оценивая её распределение с учётом поступающих данных, мы можем оценивать, сколько шагов назад была прошлая разладка. То есть, подход позволяет нам свести анализ измерений к анализу длин пробега, сообщающих нам о разладках.

Алгоритм при вычислениях декомпозируется на 3 ключевые составляющие:

- функция выживаемости;
- предсказательная модель (в базовом виде — некоторая функция правдоподобия);
- распределение длин пробега с измерениями в определённый момент времени.

3.1. Функция выживаемости

Функция выживаемости требуется для *априорной* оценки вероятности разладки в некоторый момент времени, то есть оценки, получаемой без знания самих измерений. Наиболее простым является случай постоянной функции выживаемости, соответствующий экспоненциальному распределению.

Функция задаётся в явном виде при конфигурации алгоритма. Постоянная функция выживаемости соответствует фиксированной априорной вероятности наличия разладки в любой момент времени. Для постоянной функции выживаемости вероятности будут выглядеть так:

$$P(r_t \mid r_{t-1}) = \begin{cases} 1/\lambda, & \text{если } r_t = 0, \\ 1 - 1/\lambda, & \text{если } r_t = r_{t-1} + 1, \\ 0 & \text{иначе,} \end{cases} \tag{1}$$

где λ — параметр выживаемости (hazard rate).

3.2. Предсказательная модель (функция правдоподобия)

В базовом байесовском алгоритме роль предсказательной модели выполняет функция правдоподобия, позволяющая нам при наличии сопряжённого априорного/апостериорного распределения получать апостериорные значения гиперпараметров из априорных после каждого наблюдения и с их помощью рассчитывать предсказательные вероятности.

Спецификой байесовского подхода является то, что априорные значения гиперпараметров могут выбираться произвольно, отображая наши представления о предметной области. Но в случае, когда мы создаём простой в использовании инструмент широкого назначения, встаёт вопрос об автоматизации оценки исходных априорных гиперпараметров. Важно заметить, что, помимо обновляемых при байесовском выводе гиперпараметров, в фнукции правдоподобия могут присутствовать и фиксируемые в самом начале работы другие параметры (например, некоторая зафиксированная дисперсия). Ошибки в выборе значений таких параметров могут сильно влиять на результат и не могут быть скорректированы. Поэтому для лучшей адаптивности предпочтительно иметь функцию правдоподобия, самостоятельно оценивающую все необходимые параметры с как можно менее жёстким ограничениями на моделируемое распределение.

В качестве базового варианта была выбрана нормальная функция правдоподобия с моделированием как среднего, так и дисперсии. [12] Ей соответствует нормальное-обратное гамма распределение в качестве сопряжённого априорного/апостериорного распределения. Пусть x_t^{obs} — наблюдение в момент t (на этапе после обучения априорных гиперпара-

метров), тогда:

$$\mu_{t+1} = \frac{k_t \mu_t + x_t^{obs}}{k_t + 1} \tag{2}$$

$$k_{t+1} = k_t + 1 \tag{3}$$

$$\alpha_{t+1} = \alpha_t + 1/2 \tag{4}$$

$$\beta_{t+1} = \beta_t + \frac{k_t}{k_t + 1} \frac{(x_t^{obs} - \mu_t)^2}{2} \tag{5}$$

Нормальному-обратному гамма распределению соответствует предсказательная вероятность, вычисляемая по плотности распределения Стьюдента с 2α степенями свободы:

$$P(x_{t+1}^{obs} \mid \boldsymbol{x}_{1:t}, \Theta) = t_{2\alpha_{t+1}}(x_{t+1}^{obs} \mid \mu_{t+1}, \frac{\beta_{t+1}(k_{t+1} + 1)}{k_{t+1}\alpha_{t+1}}) \tag{6}$$

Такое сопряжённое распределение накладывает ограничение на данные в виде возможности менять порядок случайных величин без изменения совместного распределения. Однако, поскольку в байесовском алгоритме параметры обновляются пошагово после каждого наблюдения, выборка состоит из одного элемента и это ограничение не имеет значения. Также из искомой формулы для параметра β единичный размер выборки убирает сумму квадратов отклонений от выборочного среднего, поскольку среднее совпадает с единственным элементом.

Исходная оценка априорных значений гиперпараметров проводится в соответствии со следующей их интерпретацией: среднее оценивается из выборки размера k_0 со средневыборочным μ_0, дисперсия оценивается из выборки размера $2\alpha_0$ со средневыборочным μ_0 и суммой квадратов отклонений $2\beta_0$. Поскольку оценка параметров проводится из одной и той же обучающей выборки, среднее у них совпадёт и будет равным μ_0. С точки зрения вычислений нам необходимо оценить лишь параметры

k_0, μ_0, α_0 и β_0. Пусть $\{x^{learn}\}_n$ — обучающая выборка размера n, тогда:

$$\bar{x}^{learn} = \frac{1}{n}\sum_{i=1}^{n} x_i^{learn} \tag{7}$$

$$\mu_0 = \bar{x}^{learn} \tag{8}$$

$$k_0 = n \tag{9}$$

$$\alpha_0 = n/2 \tag{10}$$

$$\beta_0 = \sum_{i=1}^{n} \frac{(x_i^{learn} - \bar{x}^{learn})^2}{2} \tag{11}$$

Стоит отметить, что такая функция правдоподобия даёт теоретические гарантии только для нормального (гауссовского) распределения. Тем не менее, с той или иной степенью достоверности её можно использовать и в других случаях как неоптимальную модель.

3.3. Распределение длин пробега

Распределение длин пробега с учётом предыдущих наблюдений с момента последней разладки можно рассчитать через совместное распределение длин пробега и наблюдений, а также обоснованности (evidence): $P(r_t \,|\, \boldsymbol{x}_{1:t}) = P(r_t, \boldsymbol{x}_{1:t})/P(\boldsymbol{x}_{1:t})$

Тогда распределение длин пробега на каждом шаге может быть вычислено при помощи рекуррентного произведения значений функции выживаемости, функции правдоподобия (как частного случая предсказательной модели) с учётом последнего наблюдения и распределения длин пробега на предыдущем шаге:

$$\begin{aligned} P(r_t, \boldsymbol{x}_{1:t}) &= \sum_{r_{t-1}} P(r_t, r_{t-1}, \boldsymbol{x}_{1:t}) \\ &= \sum_{r_{t-1}} P(r_t, x_t \,|\, r_{t-1}, \boldsymbol{x}_{1:t-1}) P(r_{t-1}, \boldsymbol{x}_{1:t-1}) \\ &= \sum_{r_{t-1}} P(r_t \,|\, r_{t-1}) P(x_t \,|\, r_{t-1}, \boldsymbol{x}_t^{(r)}) P(r_{t-1}, \boldsymbol{x}_{1:t-1}). \end{aligned}$$

Важно заметить, что сумма в любой момент времени будет иметь

конечное количество слагаемых, поскольку длина пробега ограничена длиной временного отрезка.

3.4. Алгоритм

В прошлой работе мной был подробно описан шаг байесовского алгоритма, за который обновляются распределение длин пробега и гиперпараметры функции правдоподобия. Важной особенностью проекта PySATL-CPD-Module является доступ алгоритмов к целому окну заранее известных данных. Ещё одной особенностью является то, что необходимость обучения исходных априорных гиперпараметров с возвращением на момент последней разладки нарушает линейность обхода данных.

В алгоритме (рис. 1) выделены этапы обучения функции правдоподобия, последовательного байесовского обновления (алг. 1), обнаружения (детектирования) разладки и определения её местоположения (локализации). Для частного случая вычисления константной функции выживаемости (форм. 1), оценивания (форм. 9, 8, 10, 11) исходных гиперпараметров нормальной функции правдоподобия, их обновления (форм. 3, 2, 4, 5) и вычисления предсказательных вероятностей (форм. 6) используются указанные выше формулы.

Для простоты в описанном алгоритме рассматривается только задача локализации разладки, однако её решение легко сводится к решению для задачи детектирования разладки. Также указаны базовый детектор (алг. 2) и локалайзер (алг. 3), основанные на преодолении порога вероятности максимальной длины пробега и на выборе наиболее вероятной не максимальной длины пробега соответственно. После выбора фиксированного порога детектора их сигнатура оказывается наиболее общей. Возможны и другие детекторы, локалайзеры, имеющие такую же сигнатуру, вопрос качества их работы является предметом для дальнейшего исследования.

Algorithm 1 Bayesian update step

```
1: function BayesianUpdate(array[float] GrowthProbs, float observation, int gapSize)
2:     PredictiveProbs ← LikelihoodPredict(observation)
3:     HazardVal ← Hazard(Arange(gapSize))
4:
5:     ChangePointProb ← GrowthProbs[0 : gapSize] * PredictiveProbs * HazardVal
6:     GrowthProbs[1 : gapSize + 1] ← GrowthProbs[0 : gapSize] * PredictiveProbs * (1.0 − HazardVal)
7:     GrowthProbs[0] ← ChangePointProb
8:
9:     Evidence ← Sum(GrowthProbs)
10:    GrowthProbs[0 : gapSize + 1] ← GrowthProbs[0 : gapSize + 1]/Evidence
11:
12:    LikelihoodUpdate(observation)
```

Algorithm 2 Simple detector

```
1: function SimpleDetect(array[float] GrowthProbs, float threshold)
2:     return GrowthProbs[−1] < threshold
```

Algorithm 3 SimpleLocalizer

```
1: function SimpleLocalize(array[float] GrowthProbs)
2:     return Argmax(GrowthProbs)
```

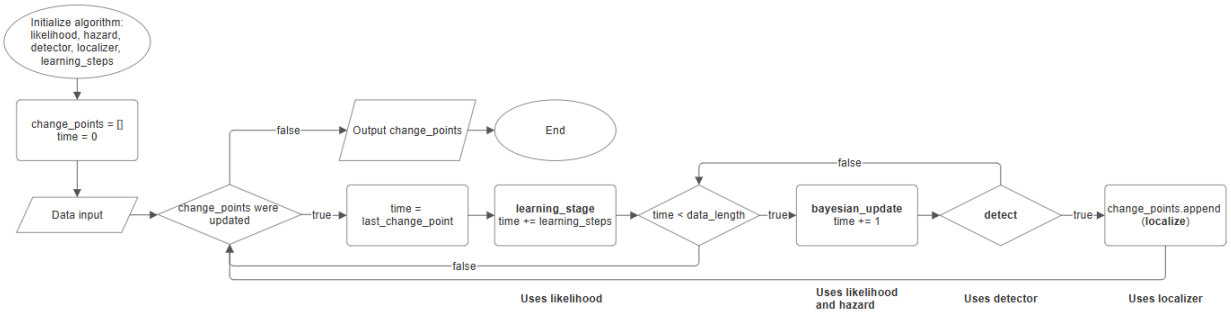

Рис. 1: Блок-схема адаптированного под архитектуру байесовского алгоритма

4. Реализация

Данный раздел посвящён реализации алгоритма. Он начинается с перечисления используемых библиотек и инструментов. Затем описываются абстракции, определяющие ключевые интерфейсы для работы алгоритма в составе PySATL-CPD-Module. Далее следует описание первых результатов, полученных при запуске алгоритма на сгенерированных размеченных экспериментальных данных: обсуждаются ключевые статистики, их предполагаемое и наблюдаемое поведение, а также то, какие на основе этого могут быть сделаны предположния об устройстве детекторов.

4.1. Проект PySATL-CPD-Module

Совместно со студентами Темерланом Ахметовым [1], Алексеем Ивановым [2], Артёмом Романюком [3] и Александрой Листковой [4] на летней школе по программированию была разработана архитектура модуля PySATL-CPD (рис. 2), в частности, описывающая интерфейс для работы с алгоритмом. Одной из особенностей инструмента является использование сркаббера (от англ. «to scrub» — «скрести») для того, чтобы из общего потока данных пошагово выбирать некоторые сегменты и передавать их на обработку алгоритмам в оффлайн-режиме. Проходя

[1] https://github.com/Lesh79
[2] https://github.com/Sor4nex
[3] https://github.com/wrdxwrdxwrdx
[4] https://github.com/alistkova

скраббером по всем данным мы получаем уже онлайн-алгоритм. Скраббер также может поддерживать ленивое добавлене новых данных.

На текущий момент модуль позволяет генерировать размеченные наборы данных с задаваемой конфигурацией распределений и передавать их в скраббер, представляющий из себя движущееся с наложениями окно. Алгоритм при этом может возвращать количество найденных разладок (детектировать, `detect` в коде программы) или их список (локализовывать, `localize` в коде программы). Поскольку алгоритм работает лишь с данными, вычлененными скраббером, моменты найденных разладок должны также пересчитываться скраббером относительно общей временной шкалы. Работу модуля определяют следующие абстракции:

- Абстрактный класс `Algorithm` с методами `detect` и `localize`, задающий интерфейс для реализации конкретных алгоритмов.
- Абстрактный класс `Scrubber`, задающий метод скраббинга данных для перевода алгоритма из оффлайн-режима в онлайн-режим.
- Класс `CPDShell`, обеспечивающий весь пайплайн работы с данными; пользователь передаёт алгоритм, скраббер, рабочий сценарий (обнаружение разладки, определение её местоположения, количество разладок для поиска) и данные.

Проект реализуется на Python. Часть ситуаций, в которых появляется задача об обнаружении разладки, требует высокой производительности, для чего были бы предпочтительнее другие языки программирования. Однако на данный момент проект сфокусирован на оценке теоретических свойств алгоритмов, поэтому основное внимание уделяется общему интерфейсу, возможностям для воспроизводимого бенчмаркинга (в первую очередь, мощности статистических критериев) на различных данных и оценке оптимальных параметров по умолчанию, проектированию инженерной составляющей. Pyhton же предоставляет множество инструментов для работы с данными и широко используется в этой области. Кроме того, использование специализированных библиотек позволяет существенно ускорить работу с данными по сравнению

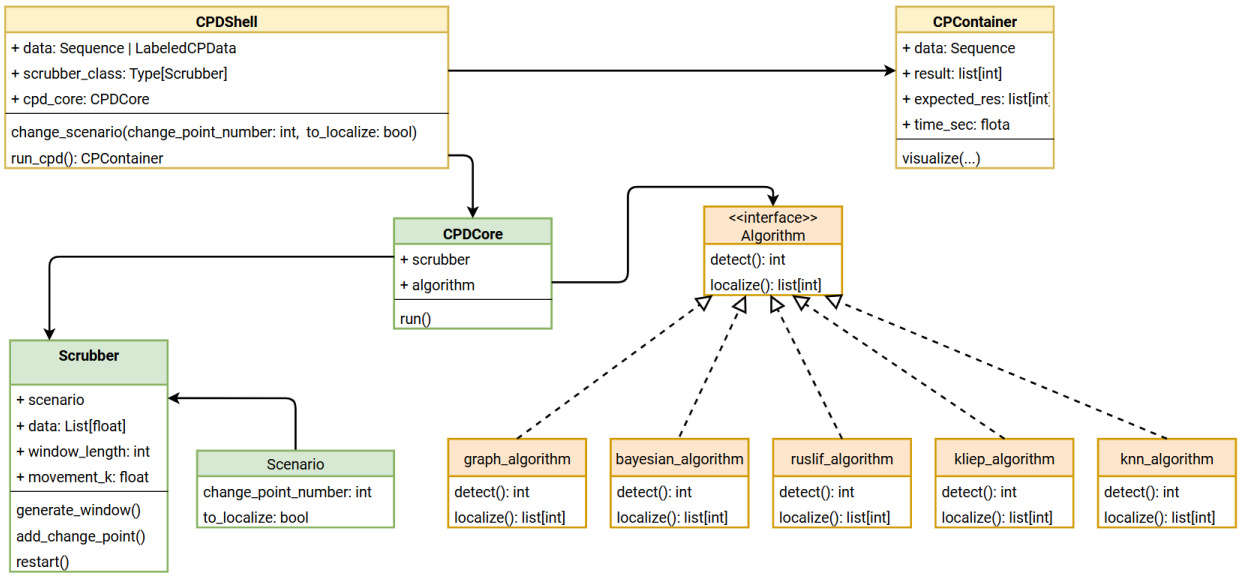

Рис. 2: UML-диаграмма ядра проекта PySATL-CPD-Module

со встроенными по умолчанию в язык структурами и операциями.

4.2. Интеграция алгоритма в PySATL-CPD-Module

Байесовский онлайн-алгоритм обнаружения разладки был реализован (рис. 3) и интегрирован в пакет для обнаружения разладок PySATL-CPD-Module. При этом были учтены особенности архитектуры модуля, а используемые для конфигурирования алгоритма сущности были выделены в отдельные абстракции.

В проекте используются линтеры black [10] и ruff [17], проверка аннотации типов осуществляется Муру [13]. При реализации байесовского алгоритма активно использовала библиотека NumPy [14], предоставляющая широкую функциональность для работы с вычислениями (в первую очередь, существенными были арифметические операции над массивами вероятностей). Для вычисления предсказательных вероятностей использовались распределения, реализованные в SciPy [19].

Основная функциональность находится в классе `BayesianAlgorithm`, обеспечивающем подсчёт всех необходимых статистик и их анализ. Для лучшей конфигурируемости в него возможно передавать объекты классов, реализующих интерфейсы из следующих абстрактных классов:

- `ILikelihood`, который определяет интерфейс для работы с функциями правдоподобия, использующимися в качестве предсказательных моделей. Позволяет обучать исходные априорные параметры, рассчитывать из априорных параметров апостериорные с учётом наблюдения, получать предсказательные вероятности по наблюдению и очищать текущее состояние. На данный момент реализована функция правдоподобия для нормального распределения в классе `GaussianUnknownMeanAndVariance`, использующая нормальное-обратное гамма сопряжённое распределение.

- `IHazard`, который определяет интерфейс для работы с функциями выживаемости. Позволяет по распределению длин пробега получать априорные вероятности разладки в текущий момент. На данный момент реализована постоянная функция выживаемости, соответствующая экспоненциальному распределению, в классе `ConstantHazard`, которая параметризуется константой в знаменателе искомой вероятности.

- `IDetector`, который определяет интерфейс для детектирования (определения наличия) разладки в текущий момент по распределению длин пробега. Позволяет по переданному распределению длин пробега определить наличие разладки и очистить текущее состояние. На данный момент реализованы детекторы, сравнивающие вероятность максимальной длины пробега с порогом (класс `SimpleDetector`) и падение вероятности максимальной длины пробега за последний шаг (класс `DropDetector`). Существуют и другие варианты рассматриваемых статистик. Качество работы этих детекторов, как и других предполагаемых, является предметом для дальнейшего исследования.

- `ILocalizer`, который определяет интерфейс для локализации (определения местоположения) разладки в текущий момент по распределению длин пробега. Выбрав новую длину пробега, алгоритм устанавливает новую разладку соответствующее количество ша-

гов назад. На данный момент реализован локалайзер, выбирающий наиболее вероятную длину пробега за исключением максимальной (класс `SimpleLocalizer`). Аналогично детекторам, требуют расширения списка реализаций и дальнейшего исследования.

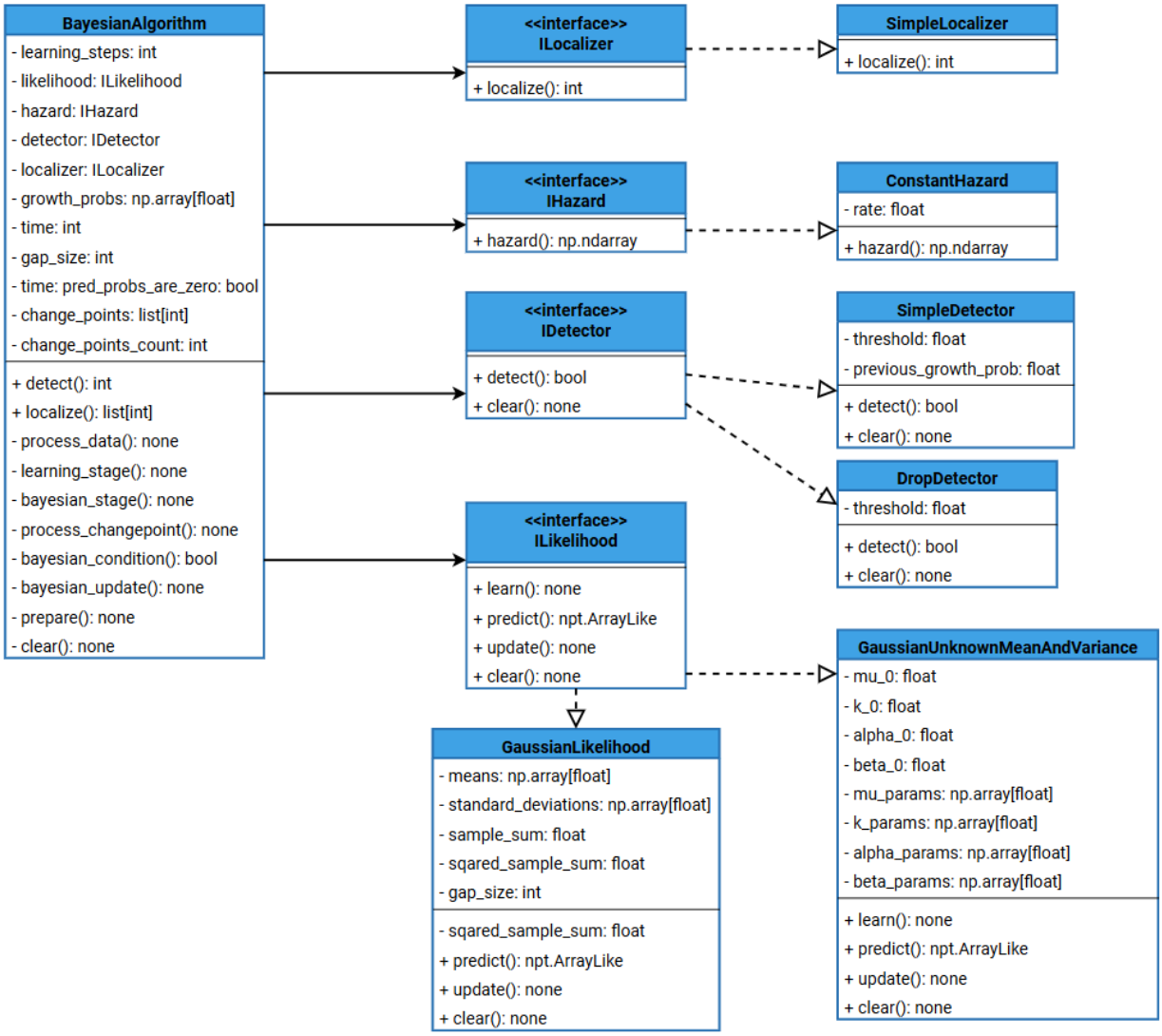

Рис. 3: UML-диаграмма байесовского алгоритма

Помимо этого была осуществлена попытка реализовать функцию правдоподобия для нормального распределения в классе `GaussianLikelihood` на основе обновления параметров средневыборочного и несмещённой выборочной дисперсии, а также нормального предсказательного распределения. Однако полученные результаты, хоть и продемонстрировали некоторый общий шаблон поведения при наличии

разладки (не наблюдаемый при её отсутствии), не соответствуют теоретической интерпретации распределения длин пробега.

В случае отсутствия разладки вероятность максимальной длины постепенно падала, что не противоречит статье. Однако в случае её наличия наблюдалось краткосрочное проседание значений, дальнейший их рост к вероятности 1.0 и удержание на этом уровне. Для нормального распределения это происходило резким скачком. То есть появление разладки безвозвратно укрепляло уверенность в том, что её не было, в противовес естественному спаду этой уверенности при отсутствии разладки.

Реализация `GaussianUnknownMeanAndVariance`, в отличие от упомянутой выше, дала на синтетических данных результаты, эмпирически согласующиеся с теоретической интерпретацией распределения длин пробега. Далее будут более подробно описаны как синтетические исходные данные, так и полученные на них результаты.

Репозиторий проекта [5] [6] [7] [8]: https://github.com/Lesh79/PySATL-CPD-Module

[5] PR №1 (nickname: alexdtat) — https://github.com/Lesh79/PySATL-CPD-Module/pull/15 — дата обращения 23.10.2024

[6] PR №2 (nickname: alexdtat) — https://github.com/Lesh79/PySATL-CPD-Module/pull/24 — дата обращения 23.10.202423.10.2024

[7] PR №3 (nickname: alexdtat) — https://github.com/Lesh79/PySATL-CPD-Module/pull/25 — дата обращения 23.10.2024

[8] PR №4 (nickname: alexdtat) — https://github.com/Lesh79/PySATL-CPD-Module/pull/27 — дата обращения 23.10.2024

5. Численное исследование

Архитектура PySATL-CPD-Module предполагает получение алгоритмом некоторого сегмента данных, на которых он детектирует или локализует разладку. С точки зрения алгоритма это означает, что он работает в оффлайн-режиме. Однако, поскольку скраббер постепенно обрабатывает более новые наблюдения и передаёт их алгоритму, модуль в целом позволяет работать в онлайн-режиме. Тем не менее, это приводит к вопросу, как правильно осуществить переход от оффлайн-режима к онлайн-режиму, если для анализа доступно лишь поведение алгоритма на передаваемых ему сегментах данных?

В данном разделе приводится описание численного исследования алгоритма, на основе которого можно предложить и реализовать детектор разладки, оценить его работу и получить некоторые рекомендации по использованию. Согласно плану, сначала выбираются различные статистики для анализа в оффлайн-режиме, на их основе предлагаются различные детекторы и значения параметров. Полученные детекторы исследуются в оффлайн-режиме, что должно позволить получить оптимизированный под оффлайн-режим детектор. Наконец, детектор исследуется в онлайн-режиме.

В следующей секции описывается генерация экспериментальных данных. Первые два этапа исследования требуют одних данных, а третий — других, на данный момент в работе затрагиваются лишь данные для первых. Описываются используемые инструменты, рассматриваемые распределения, размер данных, положение в них разладки.

В завершение приводится описание проведения эксперимента для первого этапа исследования. Указываются детали проведённого эксперимента, анализируются полученные результаты.

5.1. План численного исследования

Поскольку в алгоритме используется функция правдоподобия для нормального распределения, разумно говорить о штатном режиме работы, в котором мы рассматриваем переход с некоторого нормального

распределения на любое другое распределение, и нештатного режима работы, в котором мы рассматриваем переход с любого распределения, не являющегося нормальным, на любое другое. В случае без разладки штатный режим — работа на гауссовских данных, а нештатный — на любых других.

Для начала алгоритм требует валидации для работы в штатном режиме. Валидация и подготовка к первому этапу эксперимента состоят из следующих этапов:

1. Генерация гауссовских данных без разладки со случайными параметрами (решётка параметров по 5 средним и 5 дисперсиям, 100 распределений для каждой комбинации).

2. Оценка порогов, при которых уровень значимости (вероятность найти разладку при её заведомом отсутствии в данных) составит не более 10%, 5%, 1% и 0.5%.

3. Использование этих порогов для оценки мощностей детекции и локализации в штатном режиме (здесь предлагается использовать соответствующие экспериментальные данные из 1 этапа численного исследования).

Таким образом, появляется RQ1: какова мощность метода в при работе в штатном режиме? Важно заметить, что мощность может сильно различаться в зависимости от формулировки гипотез. Поэтому из соображений соответствия возможным сценариям использования алгоритмов в PySATl-CPD-Module и специфики алгоритма было решено выделить следующие пары гипотез:

- Гипотезы детекции: в данных либо нет разладки (H_0), либо она есть (H_1); алгоритм сообщает, что либо обнаружил некоторую разладку, либо не обнаружил разладок вообще.

- Гипотезы локализации: в данных либо нет разладки (H_0), либо она есть в пределах некоторого допустимого интервала (249, допустимым отклонением эмпирически было выбрано 25; H_1); алгоритм

возвращает первую найденную разладку, поскольку после неё он независимо от прошлых данных будет пересчитывать все распределения длин пробега, и для этой разладки проверяется, попала ли она в допустимый интервал.

После валидации алгоритма в штатном режиме появляется RQ2: возможно ли применение метода для нештатного режима? При этом используются вычисленные ранее пороги вероятностей. Ответ на него можно получить в ходе запланированного численного исследования алгоритма путём проведения нескольких этапов эксперимента:

1. На данном этапе предполагается рассмотреть работу алгоритма в пределах окна скраббера, содержащем не более 1 разладки. Предварительно проводится исследование базового критерия (порог вероятности максимальной длины пробега). Этот критерий чаще всего встречался в реализациях и необходимо понять, насколько он хорош. Оценивается его мощность для наборов гипотез детекции и локализации. Далее выбираются и более подробно исследуются некоторые статистики. На основе полученных данных предполагается выдвинуть гипотезы о принципе работы возможных детекторов (что оценивается, какие параметры при этом берутся). Для начала выбираются неварьируемые параметры и фиксируются их значения. Генерируются данные для оффлайн-режима. Постоянная функция выживаемости с таким параметром, что вероятность наличия хотя бы одной разладки на окне составляет 1/2. Вычисляем в каждый момент распределение длин пробега без обработки разладок. Рассматриваем выбранные статистики, по результатам этого этапа выдвигаем гипотезы об устройстве детекторов и о значениях эмпирических параметров. В дальнейшем это позволит сформулировать на их основе статистический критерий.

2. На данном этапе предполагается оценка качества работы предложенных ранее детекторов. Формулируются гипотезы для детекции и локализации, проверяются статистические мощности всех

предложенных критериев. Данные используются такие же, как и на предыдущем этапе. По результатам этого этапа мы имеем оптимизированные в пределах окна детекторы (оффлайн-режим).

3. На данном этапе предполагается оценка качества работы алгоритма в онлайн-режиме с учётом оптимизации детекторов для оффлайн-режима. Предполагаются данные большего размера (порядка 1500 измерений) с большим количеством разладок (около 7-8) между разнообразными распределениями, в том числе и в пределах одного окне. По результатам этого этапа будет получена оценка качества работы алгоритма в онлайн-режиме и получен статистический критерий (детектор с зафиксированными значениями параметров).

5.2. Генерация данных для первого этапа

Первые два этапа эксперимента требуют одинаковых данных, поскольку оба направлены на исследование поведения алгоритма в оффлайн-режиме. Для создания синтетических данных было решено использовать встроенный в PySATL-CPD-Module генератор случайных данных. Поддерживаемые им распределения на данный момент наиболее значительны с учётом специфики проекта, поэтому другие распределения пока не рассматривались. Эмпирически был зафиксирован размер данных 499, поскольку байесовский алгоритм требует времени как на изначальное обучение и подкрепление уверенности (корректировку), так и на укрепление уверенности в том, что разладка действительно произошла. Разладка, в случае её присутствия, эмпирически расположена посередине, в точке 249, из аналогичных соображений. Важно заметить, что, хоть в онлайн-режиме разладка и не обязательно будет оказываться в центре, настройкой работы скраббера можно постараться избежать однократного появления разладки на одном из краёв окна в алгоритме. Теперь приведём описание генерируемых данных.

Параметры данных до разладки (если она есть) или на всём окне (если её нет):

- нормальное (гауссово) распределение — $\mu = 1.0$, $\sigma = 1.0$;
- экспоненциальное распределение — $\lambda = 1.0$;
- равномерное распределение — $a = 0.0$, $b = 1.0$;
- распределение Вейбулла — $k = 1.0$, $\lambda = 0.5$;
- бета-распределение — $\alpha = 0.5$, $\beta = 0.5$.

Параметры данных после разладки:

- нормальное (гауссово) распределение — $\mu = 10.0$; $\sigma = 5.0$
- экспоненциальное распределение — $\lambda = 5.0$;
- равномерное распределение — $a = 1.0$, $b = 4.0$;
- распределение Вейбулла — $k = 1.0$, $\lambda = 5.0$;
- бета-распределение — $\alpha = 5.0$, $\beta = 5.0$.

Для получения данных с разладкой были сгенерированы всевозможные комбинации указанных распределений до и после разладки (далее будем называть их конфигурацией данных).

5.3. Исследование работы алгоритма в штатном режиме

Для начала была проведена валидация базового критерия обнаружения разладки, сравнивающего вероятность максимальной длины пробега с порогом. Он соответствует классу `SimpleDetector` в коде.

Были сгенерированы 5 значений среднего и 5 значений дисперсии, по которым была составлена решётка параметров. Значения параметров оказались следующими:

- Среднее: 3.74540119, 9.50714306, 7.31993942, 5.98658484, 1.5601864.
- Дисперсия: 0.88316058, 0.5389045, 2.08107682, 1.73365944, 1.88158521.

Для каждой комбинации параметров было сгенерировано по 100 выборок размера 500. Далее на этих данных были проанализированы перебором с шагом 0.01 пороги детектора с целью достижения необходимых уровней значимости. В результате было получено следующее соответствие:

- уровень значимости $\alpha = 10\%$ достигается при пороге 0.27;
- уровень значимости $\alpha = 5\%$ достигается при пороге 0.18;
- уровень значимости $\alpha = 1\%$ достигается при пороге 0.07;
- уровень значимости $\alpha = 0.5\%$ достигается при пороге 0.04.

Для валидации обнаружения разладки был выбран штатный режим работы алгоритма, в котором рассматривается переход с гауссовских данных на любые другие. При этом использовались ранее полученные в соответствии с уровнями мощностей пороги.

По результатам анализа мощностей базового критерия было выяснено, что для гипотезы детекции мощность критерия близка к 1.0 и падает несущественно при уменьшении порога (табл. **??**), что является хорошим результатом для задачи. Для гипотезы локализации же мощности оказались меньшими (что логично, поскольку это частный случай детекции), но растущими по мере увеличения порога (табл. 1). Такое поведение вызвано особенностью сформулированных гипотез: уменьшая порог, мы исключаем более ранние ложные разладки, что перевешивает эффект от возможного пропуска настоящей разладки. При этом мощности всё ещё остаются достаточно большими и предсказуемо наилучшие для случая двух нормальных распределений. Это позволяет в ответ на RQ1 констатировать высокий уровень мощности алгоритма при работе в штатном режиме.

Таблица 1: Мощности проверки гипотезы локализации в зависимости от зафиксированного уровня значимости для перехода с нормальных данных на любые другие

Конфигурация распределений	$\alpha = 10\%$	$\alpha = 5\%$	$\alpha = 1\%$	$\alpha = 0.5\%$
normal-beta	0.794	0.824	0.864	0.878
normal-exponential	0.779	0.805	0.84	0.863
normal-normal	0.968	0.975	0.981	0.981
normal-uniform	0.87	0.893	0.917	0.929
normal-weibull	0.783	0.811	0.853	0.863

5.4. Исследование работы алгоритма в нештатном режиме

После валидации и исследования работы алгоритма в штатном режиме стоит рассмотреть его работу в нештатном. Допустим, мы имеем зафиксированные по результатам исследования штатного режима пороги. Важно понимать, что для для других распределений эти оценки могут оказаться не такими хорошими.

Некоторые представления о качестве работы этих порогов мы можем получить, рассмотрев зависимость уровня значимости от порога для различных распределений. Здесь хорошо заметно, что нормальное (рис. 7), равномерное (рис. 8) и бета-распределения (рис. 6) ведут себя крайне похожим образом. Вероятно, это связано с симметрией (для бета-распределения брались симметричные случаи) и ограниченным носителем равномерного и бета-распределений, из-за чего они лучше моделируются выбранной функцией правдоподобия. Тогда как экспоненциальное распределение (рис. 9) и распределение Вейбулла (рис. 10) асимметричны и чаще дают сильно отклоняющиеся значения, что сильно ухудшает их моделирование.

Используя зафиксированные нами пороги, рассмотрим мощности для наборов гипотез. Гипотеза детекции в целом всё так же имеет хорошую мощность, однако для конфигураций экспоненциальное-бета, Вейбулл-бета и Вейбулл-равномерное мощность оказывается не 1.0 и постепенно, хоть и не сильно, падает.

Для гипотезы локализации же имеет смысл разделить конфигурации ещё на 2 группы: переход с любых данных на гауссовские (табл. 2) и случаи, не имеющие гауссовских данных (табл. 3). В первом случае мы наблюдаем, что для нормального, равномерного и бета-распределений мощность оказывается крайне высокой и растущей, тогда как для экспоненциального распределения и распределения Вейбулла изначально мощности оказываются низкими, но растущими до более приемлемых (ошибка менее, чем в половине случаев). Это согласуется с описанной при анализе зависимости уровня значимости от порога группировкой распределений. Во втором случае мы наблюдаем похожее поведение в зависимости от того, к какой группе относилось распределение до разладке. Если они «похожи» на нормальное, то мощность высокая, если «не похожи», то низкая. В прочем, во всех случаях мощность локализации растёт по мере увеличения порога.

С учётом указанной выше разбивкой конфигураций, можно расширить применение штатного режима до случая, где у нас по мере поступления данных происходят переключения между гауссовскими данными и некоторыми другими в обе стороны. При этом переход с экспоненциального распределения или распределения Вейбулла, с учётом мощности гипотезы детекции, всё-таки будет фиксироваться, но помимо него будет заметное количество ложных преждевременных срабатываний. Исходя из роста мощностей локализации при снижении порога, есть смысл рассматривать и более низкие его значения. Так, произвольно выбранный порог 0.00001 дал мощности локализации в худшем случае $2/3$, а во всех остальных не менее $3/4$. Однако более детальное исследование других порогов не проводилось. Таким образом, отвечая на RQ2, можно постановить, что метод можно применять и на других рассмотренных данных, принимая во внимание снижение качества локализации и, возможно, понижая порог.

Таблица 2: Мощности проверки гипотезы локализации в зависимости от зафиксированного уровня значимости для перехода с любых данных на нормальные

Конфигурация распределений	$\alpha = 10\%$	$\alpha = 5\%$	$\alpha = 1\%$	$\alpha = 0.5\%$
beta-normal	0.997	0.998	1.0	1.0
exponential-normal	0.512	0.574	0.681	0.724
normal-normal	0.968	0.975	0.981	0.981
uniform-normal	0.998	1.0	1.0	1.0
weibull-normal	0.47	0.523	0.623	0.665

Таблица 3: Мощности проверки гипотезы локализации в зависимости от зафиксированного уровня значимости для нештатного режима

$\beta \backslash$Порог	0.1	0.05	0.01	0.005
$\beta_{\mathcal{B}-\mathcal{B}}$	0.945	0.96	0.976	0.983
$\beta_{\mathcal{B}-Exp}$	0.818	0.839	0.869	0.881
$\beta_{\mathcal{B}-U}$	0.993	0.996	0.998	0.998
$\beta_{\mathcal{B}-W}$	0.813	0.832	0.862	0.876
$\beta_{Exp-\mathcal{B}}$	0.309	0.353	0.439	0.477
$\beta_{Exp-Exp}$	0.337	0.393	0.471	0.508
β_{Exp-U}	0.406	0.465	0.543	0.594
β_{Exp-W}	0.374	0.425	0.518	0.544
$\beta_{U-\mathcal{B}}$	0.892	0.917	0.946	0.955
β_{U-Exp}	0.829	0.851	0.875	0.888
β_{U-U}	0.987	0.991	0.996	0.997
β_{U-W}	0.851	0.877	0.904	0.914
$\beta_{W-\mathcal{B}}$	0.324	0.37	0.456	0.498
β_{W-Exp}	0.373	0.421	0.516	0.561
β_{W-U}	0.374	0.423	0.504	0.538
β_{W-W}	0.351	0.416	0.508	0.553

5.5. Рассматриваемые статистики

Была рассмотрена работа алгоритма без обнаружения разладки на сгенерированных данных. Из 1000 образцов данных для каждой конфигурации был выбран один образец для визуального рассмотрения статистик на нём. Рассматриваются следующие статистики:

- Распределение длин пробега в каждый момент. Эта статистика даёт информацию о состоянии Байесовской модели в целом, поз-

воляет сопоставлять изменения в ней с данными.

- Вероятность максимальной возможной длины пробега в каждый момент. Поскольку именно она соответствует отсутствию новых разладок, заметное (а особенно резкое) её падение может указывать на произошедшую разладку.

- Проверка того, превосходит ли по вероятности максимальная длина пробега нулевую. Это может указывать на резкую смену распределения и предлагается в одной из статей, исследующих байесовский метод, в качестве эвристики. [9]

- Наиболее вероятная длина пробега в каждый момент. Вероятность максимальной длины пробега может оказываться очень небольшой, но если другие вероятности ещё меньше, то эта статистика позволяет оценивать нам состояние модели в условиях повышенной неопределённости. При этом достаточно стабильные изменения этой статистики могут указывать на новую длину пробега и, следовательно, на новую разладку.

5.6. Обсуждение неудач локализации при работе алгоритма в штатном режиме

Для предложения возможного устройства альтернативных детекторов важным частным случаем являются ошибки локализации при работе алгоритма в штатном режиме (переключение с гауссовских данных на любые другие). Во всех таких случаях момент *детекции* разладки был позже настоящей разладки, а *локализована* она была раньше допустимого интервала. Таким образом, здесь уместно говорить именно об ошибке локалайзера, что не влияет на возможности алгоритма при детекции. Тем не менее, в ситуации, когда мы слишком рано сообщаем о разладке, не имея данных для уверенной её локализации, ответственность за это лежит на детекторе. Это указывает на возможную предпочтительность разделения детекторов для режима детекции (которые

будут замечать разладку как можно быстрее) и локализации (которые будут выжидать некоторое время для более надёжной локализации). При этом не было ни одного случая, чтобы разладка не была найдена вообще.

Из всех случаев для каждой конфигурации было выбрано по 2 случая с достаточно большим отклонением от настоящий разладки, поскольку среди малых отклонений сложнее определить, является ли это неизбежной статистической неточностью или фундаментальным изъяном критерия. На этих примерах можно заметить 2 существенные особенности, сочетание которых в разных пропорциях так искажает результат:

1. Падение наиболее вероятной длины пробега происходит достаточно быстро и состоит из двух этапов, и преждевременная реакция на первый приводит к недооценке падения длины пробега (рис. 4).

2. Падение наиболее вероятной длины пробега происходит достаточно медленно и состоит из двух этапов. Первое падение оказывается слишком маленьким, что ведёт к принятию «размытой» вероятностной массы исходной длины пробега за длину пробега, соответствующую новой разладке, тогда как более точная оценка случается позже (рис. 4).

Стоит отметить, что сравнение по вероятности максимальной длины пробега с нулевой оказывается наиболее устойчивым критерием, но требует больше времени. Помимо этого, полезными при рассмотрении детекторов могут быть толерантность к небольшим падениям наиболее вероятной длины пробега (из предположения, что мы вряд ли засечём две настолько близкие разладки, т.е. это искажение) и к краткосрочным выбросам, недостаточным падениям. Однако остаются промежуточные случаи с заметным падением и долгим удержанием новой ложной длины пробега, обработка которых представляется затруднительной. Анекдотические наблюдения за работой алгоритма во внештатном режиме также указывают на схожие проблемы.

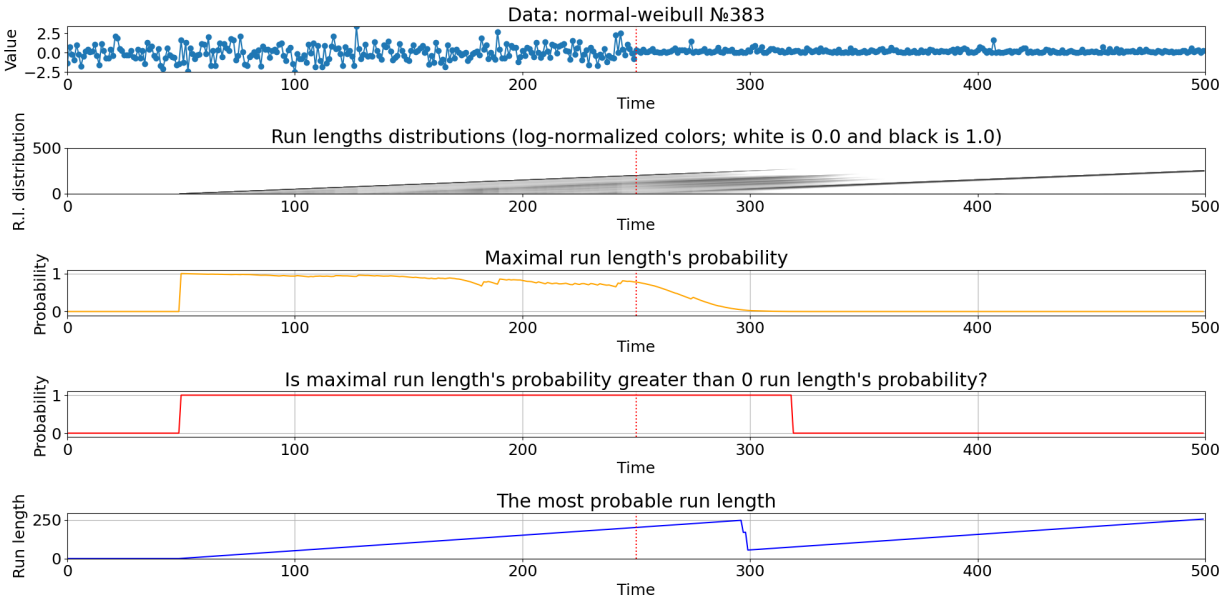

Рис. 4: Пример падения наиболее вероятной длины пробега в два близких этапа.

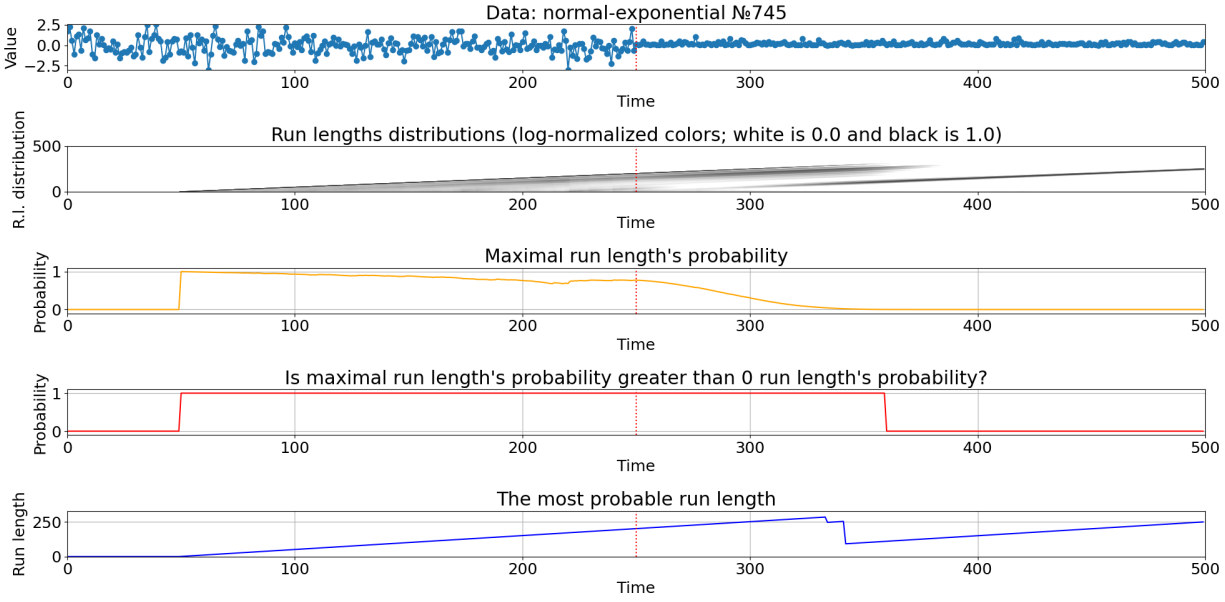

Рис. 5: Пример небольшого падения наиболее вероятной длины пробега перед большой корректировкой.

Ещё одной характерной особенностью алгоритма можно назвать то, что после первой разладки ближе к концу окна вероятностная масса плотно концентрируется вокруг настоящей длины пробега. Впрочем к этому стоит относиться с осторожностью, поскольку на таких временных промежутках возможно появление новых разладок, а для реаль-

ных, более зашумлённых данных такая уверенность может оказаться недостижимой.

Заключение

По результатам работы байесовский онлайн-алгоритм обнаружения разладки был адаптирован и интегрирован в проект PySATL-CPD-Module, а также было проведено его численное исследование. По ходу работы были выполнены следующий задачи:

1. Получен алгоритм для оффлайн-задачи обнаружения разладки
2. Код с реализацией алгоритма находится в репозитории (nick: alexdtat): https://github.com/Lesh79/PySATL-CPD-Module
3. Проведено численное исследование алгоритма:
    - Вычислены пороговые значения для гипотезы обнаружения разладки
    - Численное моделирование показало мощность, близкую к 1, для задачи обнаружения разладки в штатном и нештатном режимах
    - Алгоритм показал хорошие результаты для гипотез корректного обнаружения разладки в штатном режиме и обнаружения возвращения из разлаженного состояния
    - Для нештатного режима алгоритм показывает высокую мощность для равномерного и бета-распределений и слабую для Вейбулла и экспоненциального
4. Выявлены 2 паттерна некорректной локализации разладки для дальнейшего улучшения детекции, также в дальнейшем предлагается использовать непороговый подход для локализации

В будущем планируется продолжение работы над проектом. На основе анализа полученных результатов могут быть предложены новые виды детекторов, исследование которых также входит в разработанный план экспериментов. В рамках других исследований конфигурируемость алгоритма позволит реализовать новые функции правдоподо-

бия и другие предсказательные модели, более сложные функции выживаемости. В перспективе на этапе обучения в алгоритм может быть добавлен выбор предсказательной модели после проверки обучающей выборки критериями согласия.

Список литературы

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

[2] Aminikhanghahi Samaneh, Cook Diane Joyce. A survey of methods for time series change point detection // Knowledge and Information Systems. — 2016. — Vol. 51. — P. 339 – 367. — URL: https://api.semanticscholar.org/CorpusID:15595198.

[3] BOCD. — URL: https://github.com/naspli/BOCD/tree/master (дата обращения: 2024-11-14).

[4] BOCD. — https://github.com/AyeshaUlde/BOCD.

[5] Bayesian Changepoint Detection. — URL: https://github.com/hildensia/bayesian_changepoint_detection/tree/master (дата обращения: 2024-11-14).

[6] Bayesian Changepoint Detection. — URL: https://github.com/jeremy9959/bayesian_changepoint_detection (дата обращения: 2024-11-14).

[7] Bayesian Online Changepoint Detection. — URL: https://github.com/dtolpin/bocd/tree/master (дата обращения: 2024-11-14).

[8] Bayesian Online Changepoint Detection. — URL: https://github.com/gwgundersen/bocd/tree/master (дата обращения: 2024-11-14).

[9] Cakmak Ayse, Reinertsen Erik, Nemati Shamim, Clifford Gari D. Benchmarking changepoint detection algorithms on cardiac time series. — 2024. — 2404.12408.

[10] Black: The Uncompromising Code Formatter. — https://github.com/psf/black.

[11] Malladi Rakesh, Kalamangalam Giridhar P, Aazhang Behnaam. Online Bayesian change point detection algorithms for segmentation of epileptic activity // 2013 Asilomar Conference on Signals, Systems and Computers. — 2013. — P. 1833–1837.

[12] Murphy Kevin P. Conjugate Bayesian analysis of the Gaussian distribution. — 2007. — URL: https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf.

[13] Mypy. — URL: https://mypy-lang.org/ (дата обращения: 2024-10-06).

[14] NumPy. — URL: https://numpy.org/ (дата обращения: 2024-10-06).

[15] Rbeast: A Python package for Bayesian changepoint detection and time series decomposition. — https://github.com/zhaokg/Rbeast/tree/master/Python.

[16] Robust and Scalable Bayesian Online Changepoint Detection. — URL: https://github.com/maltamiranomontero/DSM-bocd/tree/main (дата обращения: 2024-11-14).

[17] Ruff. — URL: https://docs.astral.sh/ruff/ (дата обращения: 2024-10-06).

[18] Saatçi Yunus, Turner Ryan D, Rasmussen Carl E. Gaussian process change point models // Proceedings of the 27th International Conference on Machine Learning (ICML-10). — 2010. — P. 927–934.

[19] SciPy. — URL: https://scipy.org/ (дата обращения: 2024-10-06).

[20] Von Toussaint Udo. Bayesian inference in physics // Reviews of Modern Physics. — 2011. — Vol. 83, no. 3. — P. 943.

[21] Wu Haoxuan, Schafer Toryn L. J., Matteson David S. Trend and Variance Adaptive Bayesian Changepoint Analysis & Local Outlier Scoring. — 2024. — 2011.09437.

[22] Zhao Zhan, Koutsopoulos Haris N, Zhao Jinhua. Detecting pattern changes in individual travel behavior: A Bayesian approach // Transportation research part B: methodological. — 2018. — Vol. 112. — P. 73–88.

[23] bocd. — https://github.com/y-bar/bocd/tree/master.

[24] changepoint - Change point detection for Rust. — https://github.com/promised-ai/changepoint.

Appendix

β\Порог	0.27	0.18	0.07	0.04
$\beta_{\mathcal{B}-\mathcal{B}}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{B}-Exp}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{B}-\mathcal{N}}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{B}-W}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{B}-U}$	1.0	1.0	1.0	1.0
$\beta_{Exp-\mathcal{B}}$	0.999	0.998	0.998	0.998
$\beta_{Exp-Exp}$	1.0	1.0	1.0	1.0
$\beta_{Exp-\mathcal{N}}$	1.0	1.0	1.0	1.0
β_{Exp-U}	1.0	1.0	1.0	1.0
β_{Exp-W}	1.0	1.0	1.0	1.0
$\beta_{\mathcal{N}-\mathcal{B}}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{N}-Exp}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{N}-\mathcal{N}}$	1.0	1.0	1.0	1.0
β\Порог	0.27	0.18	0.07	0.04
$\beta_{\mathcal{N}-U}$	1.0	1.0	1.0	1.0
$\beta_{\mathcal{N}-W}$	1.0	1.0	1.0	1.0
$\beta_{U-\mathcal{B}}$	1.0	1.0	1.0	1.0
β_{U-Exp}	1.0	1.0	1.0	1.0
$\beta_{U-\mathcal{N}}$	1.0	1.0	1.0	1.0
β_{U-U}	1.0	1.0	1.0	1.0
β_{U-W}	1.0	1.0	1.0	1.0
$\beta_{W-\mathcal{B}}$	0.995	0.994	0.993	0.992
β_{W-Exp}	1.0	1.0	1.0	1.0
$\beta_{W-\mathcal{N}}$	1.0	1.0	1.0	1.0
β_{W-U}	0.986	0.984	0.978	0.978
β_{W-W}	1.0	1.0	1.0	1.0

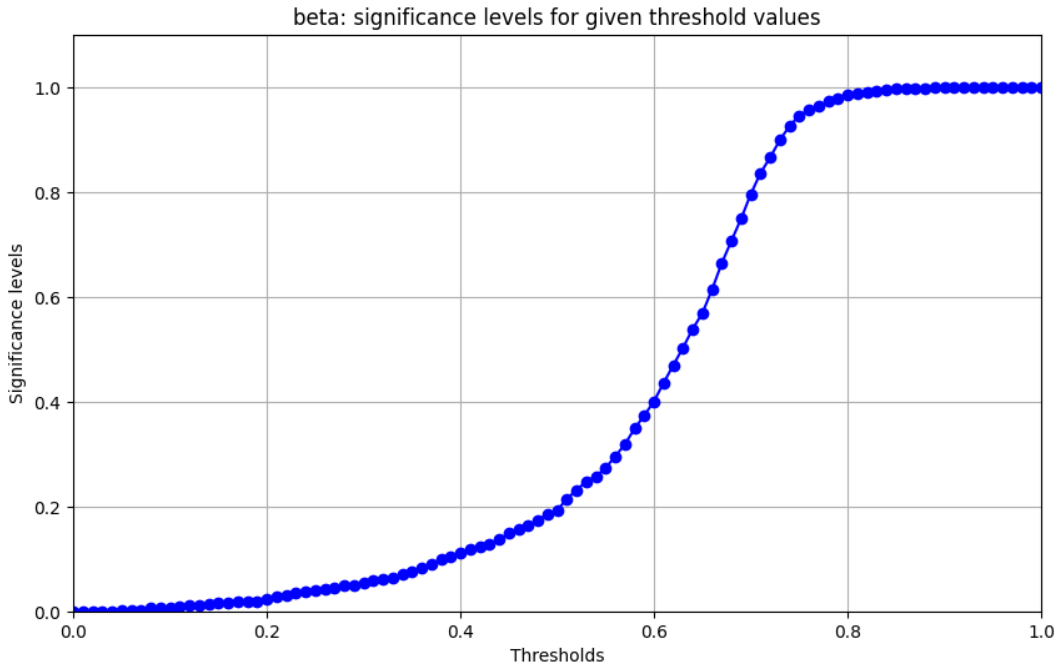

Рис. 6: Зависимость уровня значимости от значения порога для бета-распределения.

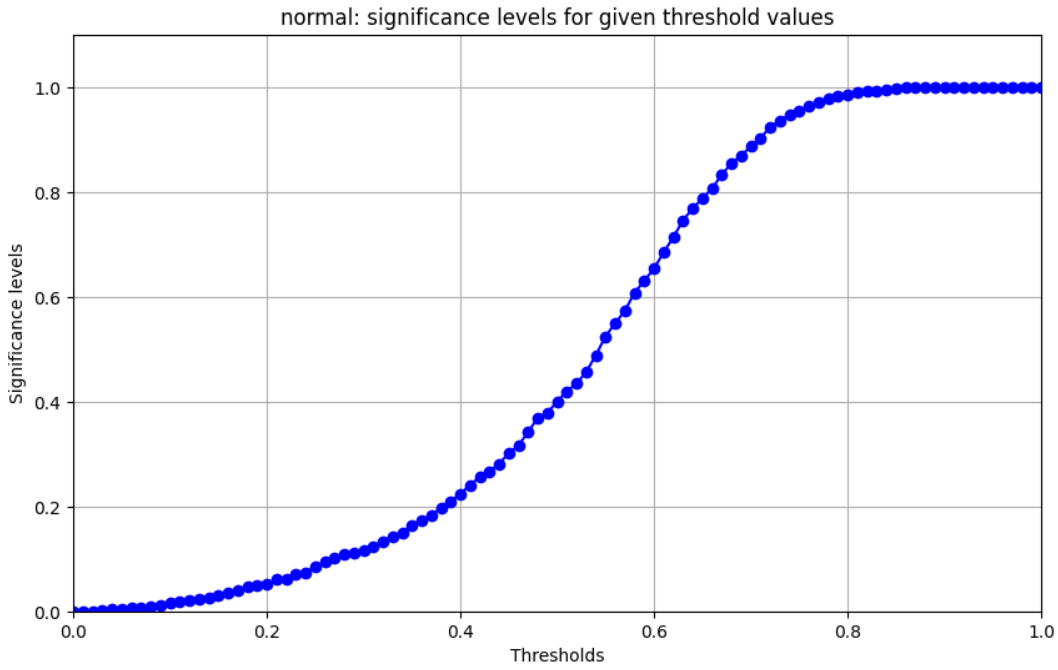

Рис. 7: Зависимость уровня значимости от значения порога для нормального распределения.

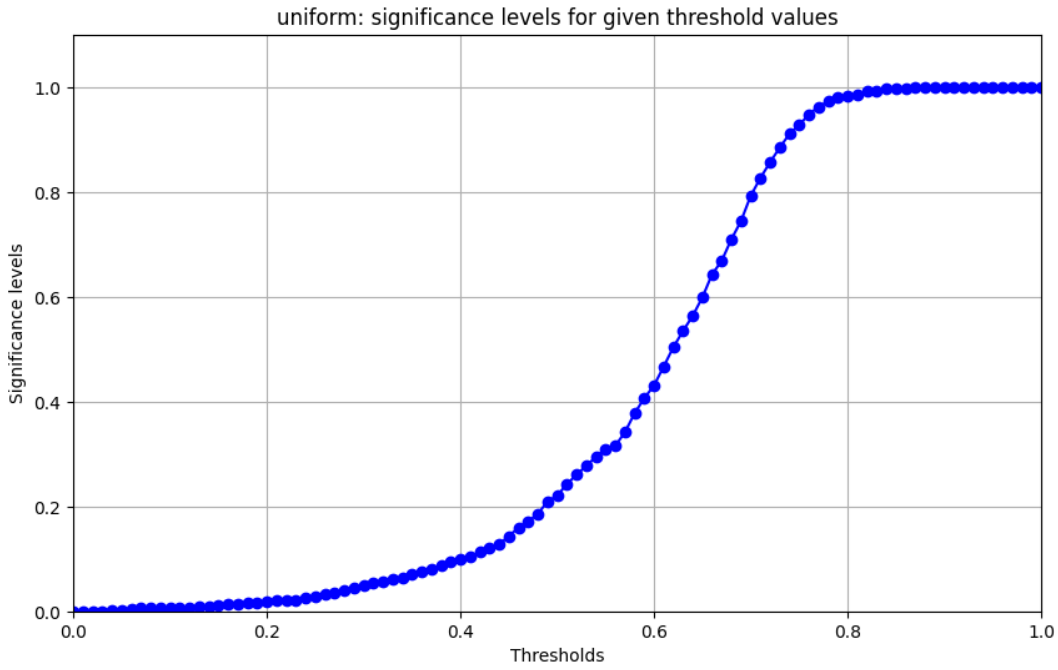

Рис. 8: Зависимость уровня значимости от значения порога для равномерного распределения.

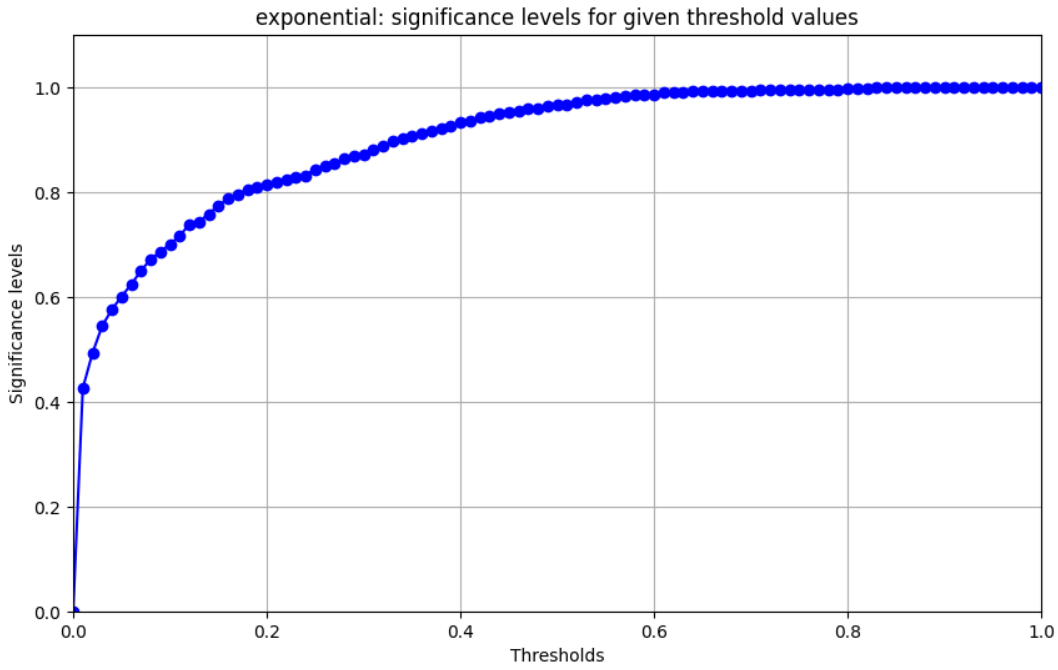

Рис. 9: Зависимость уровня значимости от значения порога для экспоненциального распределения.

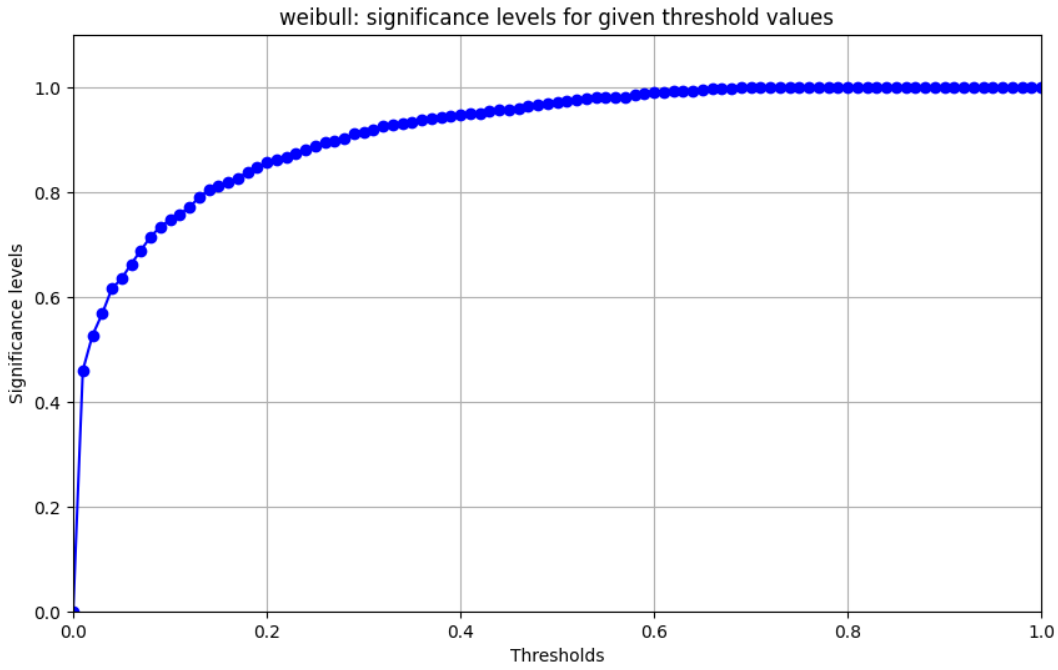

Рис. 10: Зависимость уровня значимости от значения порога для распределения Вейбулла.