МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Директор Ассоциации «Искусственный интеллект в промышленности»

_____ Т. М. Супатаев
___ _____ 2024 г.

УТВЕРЖДАЮ
Научный руководитель ИЦ СИИП Университета ИТМО

_____ А. В. Бухановский
___ _____ 2024 г.

БИБЛИОТЕКА АЛГОРИТМОВ СИЛЬНОГО ИИ ДЛЯ ЗАДАЧ ПРОМЫШЛЕННОСТИ

КОМПОНЕНТ ПРЕДОБРАБОТКИ, АВТОНОМНОГО ОЦЕНИВАНИЯ И ПРОГНОЗИРОВАНИЯ СОСТОЯНИЯ СЛОЖНЫХ ОБЪЕКТОВ И ПРОЦЕССОВ НА ОСНОВЕ ИНТЕЛЛЕКТУАЛЬНОЙ ОБРАБОТКИ СОБЫТИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ И НЕДОСТОВЕРНОСТИ ДАННЫХ

РУКОВОДСТВО ПРОГРАММИСТА

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.00853-02 33 21-ЛУ

Представители
Организации-разработчика

Руководитель разработки

_____ И.В. Котенко
_____ 2024 г.

Нормоконтролер

_____ А.В. Киреева
_____ 2024 г.

2024

Подп. и дата | Инв.№ дубл. | Взам.инв.№ | Подп. и дата | Инв.№ подл.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН
RU.СНАБ.00853-02 33 12-ЛУ

БИБЛИОТЕКА АЛГОРИТМОВ СИЛЬНОГО ИИ ДЛЯ ЗАДАЧ ПРОМЫШЛЕННОСТИ

КОМПОНЕНТ ПРЕДОБРАБОТКИ, АВТОНОМНОГО ОЦЕНИВАНИЯ И ПРОГНОЗИРОВАНИЯ СОСТОЯНИЯ СЛОЖНЫХ ОБЪЕКТОВ И ПРОЦЕССОВ НА ОСНОВЕ ИНТЕЛЛЕКТУАЛЬНОЙ ОБРАБОТКИ СОБЫТИЙ В УСЛОВИЯХ НЕОПРЕДЕЛЕННОСТИ И НЕДОСТОВЕРНОСТИ ДАННЫХ

РУКОВОДСТВО ПРОГРАММИСТА

RU.СНАБ.00853-02 33 21

ЛИСТОВ 81

Инв.№ подл.	Подп. и дата	Взам.инв.№	Инв.№ дубл.	Подп. и дата

2024

АННОТАЦИЯ

Документ содержит указания по настройке и применению компонента библиотеки алгоритмов сильного ИИ в части алгоритмов предобработки, автономного оценивания и прогнозирования состояния сложных объектов и процессов на основе интеллектуальной обработки событий в условиях неопределенности и недостоверности данных RU.СНАБ.00853-02 33 21.

В документе приведены следующие сведения:

а) назначение компонента, а также область и условия его применения (функциональные и технические);

б) прикладные задачи, решаемые с помощью компонента;

в) основные характеристики и особенности компонента;

г) способы программного взаимодействия с компонентом (обращения, входные и выходные данные, генерируемые сообщения);

д) способы программной проверки работоспособности компонента;

е) особенности применения сильного искусственного интеллекта.

Документ позволит полноценно использовать функциональные возможности расширенной версии ранее построенного компонента для разработки специализированных программных решений для широкого спектра задач, требующих оценивание и прогнозирование состояния сложных объектов и процессов с применением сильного искусственного интеллекта.

Разработанное программное обеспечение предназначено для предобработки, автономного оценивания и прогнозирования состояния сложных объектов и процессов входит в состав инструментального ПО, разрабатываемого в рамках плана Исследовательского центра в сфере искусственного интеллекта «Сильный ИИ в промышленности» (ИЦ ИИ) в соответствии с соглашением с АНО «Аналитический центр при Правительстве Российской Федерации» (ИГК 000000D730321P5Q0002), № 70–2021–00141, с целью осуществления предобработки, автономного оценивания и прогнозирования состояния сложных объектов и процессов на основе интеллектуальной обработки событий в условиях неопределенности и недостоверности данных.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

Компонент предобработки, автономного[1] оценивания и прогнозирования состояния сложных объектов и процессов на основе интеллектуальной обработки событий в условиях неопределенности и недостоверности данных (сокр. ПАОПС) библиотеки алгоритмов сильного ИИ RU.СНАБ.00853-02 13 12 разработан в соответствии с мероприятием «Разработка и испытание экспериментального образца библиотеки алгоритмов сильного ИИ в части алгоритмов автономного оценивания и прогнозирования состояния сложных объектов и процессов на основе интеллектуальной обработки событий в условиях неопределенности и недостоверности данных» (п. 1.1.2, 1.1.4 Плана деятельности СИИП).

Компонент предназначен для определения значений характеристик сложных технических объектов и процессов (сокр. СлОП) в текущий, прошедшие и будущие моменты времени с требуемыми точностью и достоверностью.

Компонент разработан на языке Python (версия не ниже 3.11) с использованием следующих библиотек: TensorFlow, Keras, Matplotlib, NumPy, Scikit-Learn, Pandas, pickle, data_classes, gzip.

Компонент размещен в репозитории по адресу https://gitlab.actcognitive.org/itmo-sai-code/foressment_lib.

Для использования необходим интерпретатор Python (версия не ниже 3.11).

2. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ ПРОГРАММ

2.1 Назначение программного компонента

Назначением программного компонента является предоставление базового функционала для создания и программной реализации перспективных методов ИИ, направленных на решение задач, связанных с оцениванием и прогнозированием состояния СлОП. При этом в части сильного ИИ, назначением программного комплекса является следующее (реализованное в рамках отдельных алгоритмов):

— улучшение качества предоставляемых данных, в том числе исходных и нечетких данных[2], что может быть задействовано на подготовительной стадии оценивания и прогнозирования состояния СлОП (алгоритм ПОИНД);

[1] Термин автономный используется для обозначения характеристики процесса, выполняемого с максимальным отсутствием вмешательства человека.

[2] Дискретная оптимизация и моделирование в условиях неопределенности данных. Перепелица В. А., Тебуева Ф. Б. Издательство: Академия Естествознания. 2007. ISBN: 978-5-91327-013-9.

— построение модели СлОП, основанной на знаниях, что обеспечивается путем извлечения множества знаний, описывающих связи между понятиями и действиями в предметной области оценивания и прогнозирования состояния СлОП, а также извлечением множества действий (выводов), вытекающих из этих знаний (алгоритм ИЗСНД);

— автономное[3] оценивание состояний СлОП, что обеспечивается путем автоматического извлечения высокоуровневых представлений исходных данных и взаимосвязей между ними; при этом, наличие большого числа гиперпараметров и настраиваемых весов, свойственных для глубоких НС, позволяет с достаточно высокой точностью выявлять закономерности между признаками обрабатываемого объекта и оцениваемой меткой его класса (алгоритм АОССОП);

— автономное прогнозирование состояний СлОП; что обеспечивается наличием большого числа гиперпараметров и настраиваемых весов, свойственных для глубоких НС, что позволяет с достаточно высоким качеством выявлять закономерности между признаками состояния системы и прогнозировать последующие состояния за заданный отрезок времени (алгоритм АПССОП).

Таким образом, разработанные алгоритмы, а также их совместное применение, реализуют Национальную стратегию развития искусственного интеллекта в части разработки перспективных методов искусственного интеллекта, а именно методов, направленных на автономное решение задач оценки и прогнозирования состояний сложных объектов и процессов.

2.2 Область применения

Модули разработанного компонента могут быть применены для таких объектов, как вычислительные сети и системы большой размерности (включая Интернет вещей, киберфизические системы), автономные робототехнические комплексы (беспилотные летательные аппараты, беспилотные автомобили и др.) и т. п., а также к протекающим в них процессам управления и функционирования.

К направлениям применения интеллектуальных средств оценки и прогнозирования сложных технических объектов и процессов можно отнести различные программы модернизации инфраструктуры предприятия, совершенствования промышленных установок, повышения срока их службы и снижения вероятности возникновения инцидентов на них.

[3]Термин автономный используется для обозначения характеристики процесса, выполняемого с максимальным отсутствием вмешательства человека.

Представленные интеллектуальные автономные алгоритмы позволят на основе имеющихся исторических и статистических данных бизнес-процессов выявлять их некорректные состояния и переходы, связанные в том числе со следующими событиями:

— неправильная настройка оборудования;

— износ отдельных деталей;

— возможные ошибки и неправильное использование различного технического оборудования персоналом;

— злонамеренными воздействиями со стороны внешних или внутренних нарушителей информационной безопасности.

Кроме того, использованием таких алгоритмов будет способствовать определению основных закономерностей и тенденций в дальнейшем ходе индустриальных процессов и прогнозирования дальнейших состояний технических объектов и особенностей, а также характеристик проистекающих в них процессов.

2.3 Функциональные условия применения

Экспериментально обоснованные функциональные ограничения на применение алгоритмов компонента сильного ИИ являются следующими:

1) для алгоритма ПОИНД устанавливаются:

— порог уникальности выбирается алгоритмом автоматически или может быть задан пользователем (значение по умолчанию: 0.70, пользователю рекомендуется изменять данное значение в соответствии с решаемой задачей и используемым набором данных, рекомендуемый диапазон [0.65; 0.95]);

— порог количества уникальных значений признака для отнесения к категориальному типу данных (значение по умолчанию: 10, пользователю рекомендуется изменять данное значение в соответствии с решаемой задачей и используемым набором данных, в том числе в процентном соотношении к количеству строк в наборе данных);

— максимальное количество кластеров, на которые предполагается разбивать данные (значение по умолчанию: 10, пользователю рекомендуется изменять данное значение в соответствии с решаемой задачей и используемым набором данных);

— максимальное количество итераций процесса кластеризации (значение по умолчанию: 10, пользователю рекомендуется изменять данное значение в соответствии с решаемой задачей и используемым набором данных);

— максимальное количество узлов, реализующих параллельную обработку кластеров (значение по умолчанию: 2, пользователю рекомендуется изменять данное

значение в соответствии с параметрами вычислительной техники, на которой используется алгоритм);

— минимальное/максимальное значение порога информативности признаков (значение по умолчанию: 0.70, пользователю рекомендуется изменять данное значение в соответствии с решаемой задачей и используемым набором данных, рекомендуемый диапазон [0.65; 0.95]);

2) для алгоритма ИЗСНД устанавливаются:

— максимальный размер / объем обучающих данных, характеризующих СлОП, заданных множеством описаний (ограничен объемом оперативной и долговременной памяти машины, алгоритмических ограничений нет);

— максимальное количество агрегатов, которые объединяют отдельные значения каждого признака (ограничено значением M * L * 2, где M – это количество признаков в наборе данных, L – количество различных меток класса в наборе данных);

— максимальное количество ассоциативных правил (ограничено максимальным количеством агрегатов);

— минимальное/максимальное пороговое значение метрики информативности (должно быть в интервале [0; 1] для стандартизованных метрик);

3) для алгоритма АОТССОП вводятся ограничения:

— набор данных должен представлять собой набор записей, каждая из которых описывает состояние исследуемого объекта в определенный момент времени;

— каждая запись из набора данных должна представлять собой последовательность числовых признаков фиксированной размерности, при этом величина этой размерности должна быть постоянной для всех записей;

— каждой записи из обучающего набора данных должна быть присвоена метка класса состояния;

— для достижения наилучшего результата с моделями глубокого обучения, набору данных рекомендуется иметь достаточное количество образцов (> 1000) и не менее 10 признаков;

— для достижения наилучшего результата рекомендуется воспользоваться хорошо известными рекомендациями по избавлению от таких проблем формирования выборок, как отсутствие данных, недостаточное количество данных, разбалансировка, ложные зависимости, ограниченный набор источников, изменение генеральной совокупности во времени и др.).

4) для алгоритма АПССОП вводятся ограничения:

— прогнозируются только числовые параметры состояний СлОП; работа с категориальными характеристиками не поддерживается, если они не были предварительно закодированы в числовые значения;

— для обучения модели используется фиксированный вектор характеристик для каждого состояния СлОП, длина и последовательность характеристик должна быть неизменной для состояния СЛоП в каждый момент времени (ограничения на длину вектора не накладываются);

— прогнозирование с использованием обученной модели осуществляется только для фиксированного вектора характеристик, заложенного в обученную модель;

— сумма значений длины исторической последовательности для обучения и длины горизонта прогнозирования модели не может быть больше длины обучающей выборки (ограничения на длину обучающей выборки не накладываются);

— прогнозирование с использованием обученной модели осуществляется только на основе текущей или смоделированной последовательности состояний СлОП за промежуток времени, равный длине исторической последовательности, заложенной в обученную модель;

— для формирования глубокой модели прогнозирования доступны только блоки простой рекуррентной сети (SimpleRNN), долгой краткосрочной памяти (Long Short-Term Memory, LSTM) и управляемые рекуррентные блоки (Gated Recurrent Units, GRU)..

2.4 Технические условия применения

Техническими средствами являются электронно-вычислительные машины и устройства, которые используются при работе программы, должны иметь минимально необходимые характеристики, представленные в Табл. 2.2.1. В качестве возможных операционных систем подходят любые из семейства Linux и Windows, совместимые с Python версии 3.11 и выше.

Таблица 2.2.1 – Минимально необходимые характеристики электронно-вычислительных машин и устройств для выполнения программы

Тип компьютера	Кол-во CPU x кол-во ядер	Тактовая частота CPU, ГГц	Кол-во GPU x кол-во ядер	Тактовая частота GPU, МГц	Оперативная память, Гб	Дисковая память, Гб
Рабочая станция	1 x 8	3.8	1 x 3584	1480	32	2000

3. ОПИСАНИЕ ПРИКЛАДНЫХ ЗАДАЧ

3.1 Классы решаемых задач

Компонент позволяет решить следующие классы задач:

1) Предобработка исходных и нечетких данных о СлОП (алгоритм ПОИНД – коррекция типов, устранение неполноты и мультиколлинеарности входных данных).

2) Извлечения фрагментов знаний, имеющихся в данных (алгоритмы ИЗСНД для представления знаний в виде ассоциативных правил вида «ЕСЛИ <посылка>, ТО <следствие>»)).

3) Автономное оценивание текущего состояния СлОП (алгоритм АОССОП (определение наличия или отсутствия в текущий момент времени определенного вида функциональных неисправностей, дефектов и атакующих воздействий, свойственных целевой системе)).

4) Автономное прогнозирование состояний СлОП (алгоритм АПССОП (обучение модели прогнозирования состояний СлОП на основе исторических данных в виде временного ряда; прогнозирование состояний, описываемых вектором характеристик, за заданный отрезок времени)).

Отметим, что перечень конкретных решаемых задач показывает, что каждого класса задач в отдельности уже может быть полезен разработчикам, однако для решения производственных задач, таких как, например, (1) диагностика неисправностей, (2) анализ процессов систем очистки воды, (3) контроль работы мостового крана, (4) анализ процессов электрических трансформаторов и др., может быть выгодно совместное их использование.

Приведем несколько примеров для различных отраслей промышленности:

— нефтегазовая отрасль: оценивание и прогнозирование состояния оборудования для разведки запасов углеводородов, их добычи, очистки и переработки, логистики и транспортировки нефтепродуктов с использованием трубопроводов и др.;

— энергетический сектор: оценивание показателей энергогенерации, стоимостных расходов на эксплуатацию, аварийности и пр.;

— производство: оценка и прогнозирование сбоев оборудования.

При этом в каждом из этих примеров присутствует ряд задач, связанных с предварительной обработкой данных, имеющих свойства неполноты и неоднородности, а также извлечения знаний о СлОП на их основе.

Все эти задачи решаются для предоставления как оценки текущей ситуации, так и прогнозирования ее развития с учетом имеющихся данных и заданных сценариев.

3.2 Примеры решения задач

Примеры применения конкретных задач с помощью компонента являются следующие.

3.2.1 Задача № 1

В рамках данного примера решается задача *оценивания* текущего состояния на основе данных, описывающих функционирование мостового крана. Целью эксперимента является оценивание состояния крана с точки зрения текущего цикла. Под состоянием системы при этом понимается класс цикла движения крана. Эффективность оценивания состояния системы вычисляется на основе показателей точности и полноты определения класса текущего цикла, а также среднего гармонического значения этих показателей(F-меры). Код данного пример также приведен в репозитории библиотеки[4].

Исходные данные:

Экспериментальным набором данных является Smart Crane Data[5]. Этот набор данных содержит информацию, полученную от промышленного мостового крана, подключенного к серверу OPC UA, во время его работы по L-образной траектории. Кран приводился в движение с различными нагрузками (0 кг, 120 кг, 500 кг и 1000 кг). Каждый цикл движения состоит из пятикратного повторения процесса подъема груза, движения из точки А в точку Б по траектории, опускания груза, подъема груза, возвращения в точку А и опускания груза. Всего в наборе данных имеется 8 циклов движения (Таблица 3.2.1). Количество признаков равно 12

Таблица 3.2.1 – Параметры циклов движения умного крана

№ цикла	Нагрузка (кг)	Контроль раскачивания
0	0	Включен
1	0	Выключен
2	120	Включен
3	120	Выключен
4	500	Включен
5	500	Выключен
6	1000	Включен
7	1000	Выключен

Решение задачи:

Предобработка данных включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [-1, 1] (StandardScaler). Разделение набора

[4] https://github.com/aimclub/foressment_lib/blob/master/examples/small-cranes-sample.ipynb
[5] https://ieee-dataport.org/documents/driving-smart-crane-various-loads

данных на тренировочную и тестовую выборки осуществляется соотношением 7:3. В качестве моделей оценивая использовались следующие:

- *DeepCNN* с параметрами *blocks*=2, *units*=64;
- *Hybrid_CNN_GRU* с параметрами *units*=64;
- *hybrid_variation* с параметрами *block*="*residual*", *units*=64, *loop_number*=2;
- *hybrid_variation* с параметрами *block*="*Xception*", *units*=64, *loop_number*=2.

Модели обучаются с использованием 100 эпох и параметром *batch_size* равным 32. В качестве показателей качества для оценки модели используются точность (precision, P), полнота (recall, R) и F-мера (F1) для определения цикла движения крана.

Результаты качества оценивания для перечисленных моделей представлены в Таблице 3.2.1. Показатели качества представлены как для каждого цикла отдельно, так и для всего тестового набора данных. Можно отметить, что модель *residual hybrid_variation* дает ненамного лучшие результаты, но в то же время все модели демонстрируют сравнительно высокие результаты между собой. Среди отдельных циклов более высокие оценки получены для циклов 6 и 7. Также достигается поставленное требование с точности не менее 0,7 на малых выборках данных (получены для отдельных циклов) и 0,9 на больших выборках данных (получены для всего набора данных).

Таблица 3.2.2. Результаты оценивания состояния на наборе Smart Crane Data

	DeepCNN			***Hybrid_CNN_GRU***			***residual hybrid_variation***			***Xception hybrid_variation***		
Цикл	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
0	0,88	0,88	0,88	0,87	0,82	0,84	0,93	0,87	0,89	0,86	0,89	0,87
1	0,91	0,92	0,91	0,87	0,92	0,90	0,88	0,87	0,92	0,92	0,90	0,91
2	0,91	0,86	0,88	0,93	0,83	0,88	0,89	0,93	0,90	0,91	0,85	0,88
3	0,89	0,92	0,91	0,88	0,94	0,91	0,93	0,88	0,92	0,88	0,93	0,91
4	0,90	0,83	0,86	0,87	0,88	0,87	0,90	0,87	0,88	0,88	0,84	0,86
5	0,91	0,95	0,93	0,93	0,92	0,93	0,92	0,93	0,93	0,91	0,93	0,92
6	0,93	0,89	0,91	0,94	0,88	0,91	0,94	0,94	0,92	0,91	0,93	0,92
7	0,92	0,95	0,93	0,91	0,95	0,93	0,92	0,91	0,94	0,95	0,93	0,94
Все циклы	0,90	0,90	0,90	0,90	0,89	0,89	**0,91**	**0,91**	**0,91**	0,90	0,90	0,90

3.2.2 Задача № 2

В рамках данного примера решается задача *оценивания* текущего состояния на основе данных устройств Интернета вещей. В данном случае под состоянием понимается метка класса состояния безопасности (норма или класс атаки). Целью эксперимента является оценивание безопасности состояния устройств Интернета вещей. Таким образом модель оценивания выступает в роли модели обнаружения вторжений.

Исходные данные:

Экспериментальным набором данных является Edge-IIoT[6]. Набор данных включает данные более чем 10 типов устройств Интернета вещей и анализирует 14 атак, связанных с протоколами подключения Интернета вещей: Backdoor, DDoS_HTTP, DDoS_ICMP, DDoS_TCP, DDoS_UDP, Fingerprinting, MITM, Password, Port_Scanning, Ransomware, SQL_injection, Uploading, Vulnerability_scanner, XSS . Количество признаков, описывающих параметры передачи данных между устройствами, равно 63.

Решение задачи:

Предобработка данных включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [-1, 1] (*StandardScaler*). Также используется выбор признаков с использованием методов PCA и ElasticNet. После отбора в наборе данных был выбран 41 признак. Разделение набора данных на тренировочную и тестовую выборки осуществляется соотношением 67:33. В качестве моделей оценивая использовались следующие:

– *DeepCNN* с параметрами *blocks*=3, *units*=128;

– *Hybrid_CNN_GRU* с параметрами *units*=128;

– *hybrid_variation* с параметрами *block*="*residual*", *units*=128, *loop_number*=2;

– *hybrid_variation* с параметрами *block*="*Xception*", *units*=128, *loop_number*=2.

Модели обучаются с использованием 20 эпох и параметром *batch_size* равным 32. В качестве показателей качества для оценки модели используются точность (precision, P), полнота (recall, R) и F-мера (F1). Результаты качества оценивания состояний безопасности системы, в т. ч. обнаружения атак, для перечисленных моделей представлены в Таблице 3.2.3. Точность определения нормального состояния демонстрирует возможность модели избегать ложных срабатываний. Точность оценивания состояния как атакующее позволяет вовремя обнаруживать потенциальные угрозы. Можно отметить, что модели с высокой точностью определяют нормальное поведение системы, а также показатели обнаружения

[6] https://ieee-dataport.org/documents/edge-iiotset-new-comprehensive-realistic-cyber-security-dataset-iot-and-iiot-applications

атак близки к 100%. Среди отдельных циклов более высокие оценки получены для циклов 6 и 7. Также достигается поставленное требование с точности не менее 0,7 на малых выборках данных (получены для отдельных классов состояний) и 0,9 на больших выборках данных (получены для всего набора данных).

Таблица 3.2.2. Результаты оценивания состояния на наборе Edge-IIoT

	DeepCNN			***Hybrid_CNN_GRU***			***residual hybrid_variation***			***Xception hybrid_variation***		
Состояние	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Норма	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
Атака Backdoor	1,00	0,96	0,98	1,00	0,97	0,98	1,00	0,95	0,98	0,89	0,97	0,93
АтакаDDoS_ HTTP	0,99	0,99	0,99	0,98	0,99	0,98	0,99	0,98	0,98	0,96	1,00	0,98
Атака DDoS_ICMP	1,00	0,99	1,00	1,00	0,99	1,00	1,00	0,99	1,00	1,00	0,99	1,00
Атака DDoS_TCP	0,98	1,00	0,99	0,99	0,99	0,99	0,98	0,99	0,99	0,99	0,99	0,99
Атака DDoS_UDP	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
Атака Fingerprinting	0,97	0,76	0,85	0,95	0,78	0,85	0,96	0,74	0,84	0,96	0,76	0,85
Атака MITM	0,99	1,00	0,99	0,99	1,00	0,99	1,00	1,00	1,00	1,00	1,00	1,00
Атака Password	0,99	0,95	0,97	0,98	0,96	0,97	1,00	0,94	0,96	0,94	0,96	0,96
Атака Port_Scanning	0,97	0,93	0,95	0,95	0,94	0,95	0,93	0,95	0,94	0,90	0,96	0,93
Атака Ransomware	0,88	1,00	0,93	0,90	0,98	0,94	0,89	0,99	0,99	0,99	0,88	0,93
Атака SQL_injection	0,98	1,00	0,99	0,97	1,00	0,99	0,98	0,99	0,99	0,96	1,00	0,98
Атака Uploading	0,97	0,99	0,98	0,98	0,98	0,98	0,95	1,00	0,98	1,00	0,94	0,97
Атака Vulnerability_ scanner	0,99	0,98	0,99	0,99	0,97	0,98	1,00	0,97	0,98	0,93	0,97	0,95
Атака XSS	0,99	1,00	0,99	0,99	1,00	0,99	0,97	1,00	0,98	1,00	0,92	0,96
Все состояния	**0,98**	**0,97**	**0,97**	**0,98**	**0,97**	**0,97**	**0,98**	**0,97**	**0,97**	0,97	0,96	0,96

3.2.3 Задача № 3

В рамках данного примера решается задача *прогнозирования* состояний системы очистки воды. Целью эксперимента является определение будущих характеристик процесса очистки в штатном режиме работы. Подобный подход часто используется при обнаружении аномалий, основанном на вычислении ошибок прогнозирования.

Исходные данные:

Экспериментальным набором данных является SWaT[7]. Набор данных собран на испытательном стенде, имитирующем реальную промышленную установку очистки воды. Нормальный режим работы такой системы описывается данными, регистрируемыми 25 датчиками и 26 исполнительными механизмами, когда на систему не осуществляются атаки. Количество признаков, таким образом, равно 51.

Решение задачи:

Предобработка данных включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [0, 1] (*MinMaxScaler*). Разделение набора данных на тренировочную, валидационную и тестовую выборки осуществляется соотношением 7:1:2, соответственно. Тренировочная выборка используется для обучения моделей (70% от общего объема данных), валидационная – для проверки моделей при оптимизации гиперпараметров (10% от общего объема данных), тестовая – для оценки качества моделей (20% от общего объема данных).

В данном эксперименте длина исторической последовательности равно 120 секундам (две минуты), а горизонт прогнозирования – 30 секундам. Число скрытых слоев модели прогнозирования равно 1 (базовая рекуррентная модель). Для оптимизации гиперпараметров глубокой модели прогнозирования устанавливаться следующие множества значений:

– *block_type* = {'*LSTM*', '*GRU*'},

– *units*_0 = {512, 256, 128},

– *dropout* = {0.0, 0.01, 0.1},

– *hidden_activation* = {'*tanh*', '*relu*'},

– *output_activation* = {'*linear*', '*sigmoid*'}.

В качестве алгоритма поиска используется Байесовская оптимизация[8], так как этот метод на практике показал лучшие результаты с меньшими вычислениями по сравнению с

[7] https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/
[8] Garnett R. Bayesian optimization. – Cambridge University Press, 2023.

поиском по решётке и случайным поиском[9]. В качестве показателя качества во время оптимизации гиперпараметров прогнозирования используется средняя квадратичная ошибка (MSE). В результате определены три лучшие конфигурации моделей прогнозирования:

1) GRU-1 – *block_type: GRU, units*_0: 128, *hidden_activation: tanh, dropout*: 0.1, *output_activation: sigmoid* (MSE=0.009);
2) GRU-2 – *block_type: GRU, units*_0: 256, *hidden_activation: relu, dropout*: 0.1, *output_activation: sigmoid* (MSE=0.0092);
3) GRU-3 – *block_type: GRU, units*_0: 128, *hidden_activation: relu, dropout*: 0.01, *output_activation: linear* (MSE=0.0111).

Каждая из этих моделей далее обучалась в течение 30 эпох с параметром *batch_size* равным 64. В качестве показателей качества для оценки модели используются средняя квадратичная ошибка (MSE), корень из MSE (RMSE), средняя абсолютная ошибка (MAE) и коэффициент детерминации (R^2). Минимальное значение для MSE, RMSE, MAE является нулем, что свидетельствует об идеальном совпадении прогнозов с реальными данными. Также значения для этих метрик может быть сколько угодно высоким, так как разница значений может быть сколь угодно большой. Максимальным значением R^2 является 100, а минимальное не ограничено, и может быть отрицательным. Результаты качества прогнозирования для перечисленных моделей представлены в Таблице 3.2.3 как для всего набора данных, так и для отдельных признаков. Можно отметить, что модель GRU-2 показала себя ненамного лучше двух других моделей, так как демонстрирует наиболее низкие значения ошибок. Наиболее хорошо прогнозируемыми признаками в наборе данных являются FIT101, LIT101, MV101, P101, AIT202, FIT201, P203, DPIT301, FIT301.

На рисунке 3.2.1 показана часть прогнозируемого временного ряда для каждой модели. Синим цветом обозначены входные данные для первого временного окна (Inputs), зеленым – реальные значения после первого временного окна (True). Для каждой модели дается обозначение, связанное с типом блока, количеством юнитов и размером дропаута без плавающей запятой.

Низкие оценки для других признаков связаны или с хаотичностью изменения значений без явных закономерностей или с недостатком данных. Хаотично меняющиеся признаки в Таблице выделены голубым цветом, с недостаточными данными – фиолетовым.

[9] Yu T., Zhu H. Hyper-parameter optimization: A review of algorithms and applications //arXiv preprint arXiv:2003.05689. – 2020.

Таблица 3.2.3. Результаты оценки качества прогнозирования состояний на наборе данных SWaT

Модель	**GRU-1**				**GRU-2**				**GRU-3**			
Показатель	**MSE**	**RMSE**	**MAE**	**R2**	**MSE**	**RMSE**	**MAE**	**R2**	**MSE**	**RMSE**	**MAE**	**R2**
Все признаки	0,016	0,109	0,087	**-1,381**	**0,011**	**0,075**	**0,055**	-1,904	0,016	0,093	0,066	-2,214
FIT101	0,022	0,149	0,124	0,882	0,005	0,074	0,037	0,971	0,010	0,102	0,053	0,945
LIT101	0,015	0,124	0,096	0,765	0,014	0,119	0,094	0,782	0,006	0,080	0,062	0,902
MV101	0,013	0,114	0,093	0,766	0,006	0,076	0,046	0,896	0,010	0,101	0,061	0,817
P101	0,031	0,175	0,130	0,847	0,005	0,070	0,017	0,975	0,009	0,096	0,028	0,954
P102	0,002	0,045	0,035	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
AIT201	0,080	0,283	0,274	-65,74	0,127	0,357	0,355	-104,80	0,134	0,366	0,364	-110,38
AIT202	0,011	0,106	0,085	0,875	0,004	0,059	0,038	0,961	0,003	0,051	0,032	0,971
AIT203	0,027	0,165	0,131	0,174	0,015	0,123	0,099	0,542	0,028	0,167	0,129	0,153
FIT201	0,028	0,169	0,125	0,852	0,003	0,054	0,015	0,985	0,005	0,073	0,029	0,972
MV201	0,014	0,119	0,093	0,728	0,005	0,069	0,040	0,908	0,023	0,150	0,075	0,566
P201	0,001	0,036	0,027	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P202	0,001	0,037	0,029	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P203	0,031	0,175	0,131	0,847	0,004	0,063	0,013	0,980	0,007	0,082	0,020	0,967
P204	0,002	0,045	0,036	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P205	0,021	0,144	0,077	0,832	0,028	0,167	0,074	0,772	0,025	0,157	0,093	0,799
P206	0,002	0,049	0,039	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
DPIT301	0,018	0,134	0,108	0,835	0,009	0,093	0,066	0,921	0,007	0,083	0,040	0,938
FIT301	0,023	0,151	0,120	0,825	0,004	0,066	0,030	0,966	0,029	0,171	0,072	0,775
LIT301	0,042	0,206	0,175	0,572	0,009	0,097	0,080	0,905	0,025	0,157	0,121	0,752
MV301	0,004	0,062	0,046	-1,544	0,003	0,057	0,040	-1,156	0,002	0,048	0,030	-0,508
MV302	0,017	0,129	0,097	0,614	0,023	0,150	0,104	0,481	0,021	0,143	0,076	0,527
MV303	0,008	0,087	0,067	-0,851	0,005	0,071	0,053	-0,234	0,006	0,079	0,045	-0,508
MV304	0,010	0,100	0,068	-0,175	0,008	0,089	0,057	0,074	0,019	0,137	0,076	-1,195
P301	0,002	0,048	0,037	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P302	0,030	0,172	0,133	0,798	0,010	0,100	0,042	0,932	0,055	0,235	0,093	0,623
AIT401	0,002	0,044	0,035	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
AIT402	0,003	0,058	0,046	0,046	0,012	0,108	0,092	-2,231	0,011	0,105	0,092	-2,094
FIT401	0,016	0,128	0,105	-0,003	0,013	0,116	0,092	0,174	0,017	0,132	0,105	-0,063
LIT401	0,034	0,184	0,148	0,353	0,010	0,101	0,086	0,804	0,055	0,235	0,160	-0,049
P401	0,002	0,039	0,032	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P402	0,002	0,042	0,032	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P403	0,002	0,045	0,036	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P404	0,001	0,039	0,030	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
UV401	0,002	0,047	0,037	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
AIT501	0,078	0,279	0,227	-1,462	0,069	0,262	0,219	-1,175	0,089	0,298	0,245	-1,807
AIT502	0,010	0,101	0,087	-3,095	0,004	0,064	0,051	-0,657	0,004	0,061	0,050	-0,493
AIT503	0,007	0,082	0,063	0,706	0,014	0,120	0,113	0,368	0,050	0,223	0,209	-1,185
AIT504	0,011	0,104	0,085	-0,496	0,012	0,111	0,095	-0,712	0,028	0,167	0,156	-2,875
FIT501	0,017	0,129	0,102	0,038	0,013	0,113	0,090	0,256	0,016	0,125	0,098	0,094
FIT502	0,015	0,123	0,098	-0,162	0,014	0,117	0,094	-0,057	0,014	0,118	0,095	-0,070
FIT503	0,027	0,163	0,140	-0,042	0,025	0,160	0,143	0,007	0,029	0,172	0,148	-0,149
FIT504	0,030	0,172	0,141	0,070	0,026	0,163	0,135	0,170	0,029	0,170	0,139	0,089
P501	0,002	0,045	0,036	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P502	0,001	0,038	0,030	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
PIT501	0,026	0,160	0,138	-1,726	0,018	0,134	0,107	-0,898	0,032	0,179	0,158	-2,396
PIT502	0,011	0,105	0,087	-4,581	0,003	0,058	0,046	-0,722	0,002	0,049	0,036	-0,204
PIT503	0,024	0,155	0,134	-1,862	0,016	0,127	0,101	-0,923	0,027	0,165	0,145	-2,219
FIT601	0,006	0,079	0,062	-0,718	0,000	0,021	0,007	0,877	0,001	0,026	0,004	0,816
P601	0,002	0,042	0,034	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000
P602	0,005	0,074	0,049	-0,391	0,001	0,032	0,008	0,735	0,002	0,040	0,005	0,600
P603	0,002	0,043	0,036	0,000	0,000	0,003	0,002	0,000	0,000	0,001	0,001	0,000

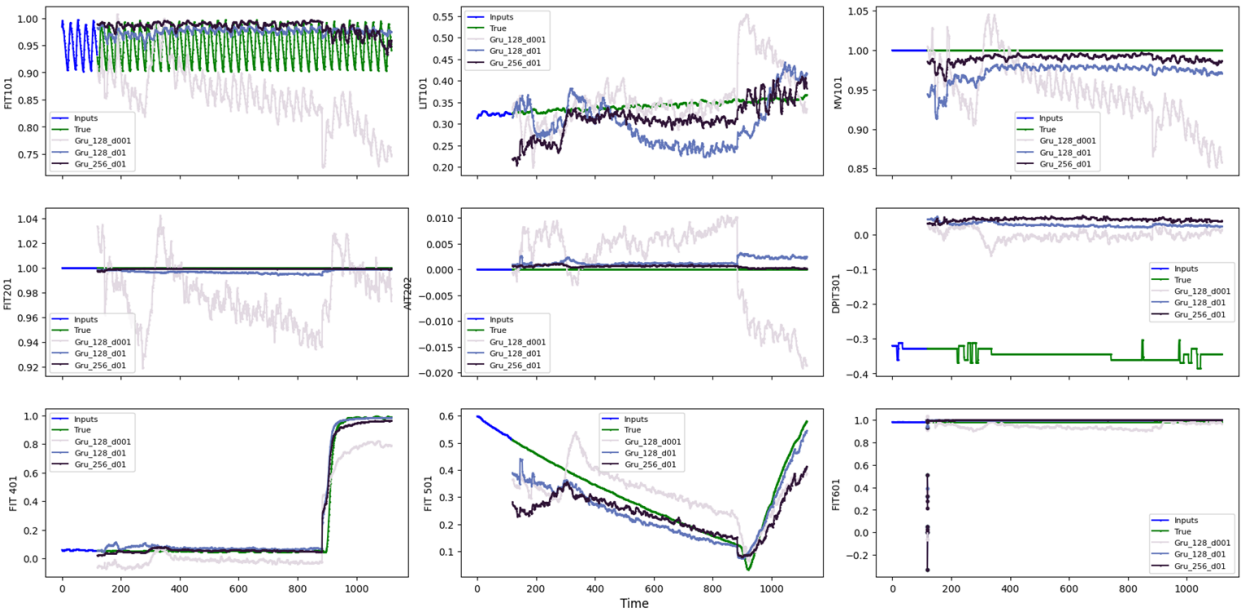

Рисунок 3.2.1 – Пример прогнозирования значений признаков набора данных SWaT

3.2.4 Задача № 4

В рамках данного примера решается задача *прогнозирования* состояний коробки передач, которая может применяться в системах диагностики неисправностей. Целью эксперимента является определение будущих характеристик работы коробки передач в стабильном режиме работы. Это может быть полезным в будущем для задач определения неисправностей путем поиска выбросов в данных.

Исходные данные:

Экспериментальным набором данных является Gearbox Fault Diagnosis[10]. Набор данных был записан с помощью четырех датчиков вибрации, расположенных в четырех разных направлениях и при изменении нагрузки от 0% до 90%. В данном эксперименте в качестве примера мы используем подвыборку данных с 90% нагрузкой.

Решение задачи:

Предобработка данных включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [0, 1] (*MinMaxScaler*). Разделение набора данных на тренировочную, валидационную и тестовую выборки осуществляется соотношением 7:1:2.

В данном эксперименте длина исторической последовательности равно 50 секундам (2 минуты), а горизонт прогнозирования – 50 секундам. Таким образом можно оценить

[10] https://www.kaggle.com/datasets/brjapon/gearbox-fault-diagnosis

прогнозирование по отдельным окнам. Для оптимизации гиперпараметров глубокой модели прогнозирования устанавливаться следующие множества значений:

– *block_type* = {'*LSTM*', '*GRU*'},

– *n_rec_layers* = {1,2}.

– *units*_0 = {128, 64},

– *units*_1 = {32, 16},

– *dropout* = {0.0, 0.01, 0.1},

– *hidden_activation* = {'*tanh*', '*relu*'},

– *output_activation* = {'*linear*', '*sigmoid*'}.

В качестве алгоритма поиска используется Байесовская оптимизация, так как этот метод на практике показал лучшие результаты с меньшими вычислениями по сравнению с поиском по решётке и случайным поиском. В качестве показателя качества во время оптимизации гиперпараметров прогнозирования используется средняя квадратичная ошибка (MSE). В результате определены 3 лучшие конфигурации моделей прогнозирования:

1) GRU – *n_rec_layers*: 2, *block_type: GRU, units*_0: 128, *units*_1: 32, *dropout*: 0.0, *hidden_activation: tanh, output_activation: sigmoid* (MSE: 0.0060);
2) LSTM-1 – *n_rec_layers*: 2, *block_type: LSTM, units*_0: 128, *units*_1: 32, *dropout*: 0.001, *hidden_activation: tanh, output_activation: sigmoid* (MSE: 0.0061);
3) LSTM-2 – *n_rec_layers*: 2, *block_type: LSTM, units*_0: 64, *units*_1: 32, *dropout*: 0.01, *hidden_activation: tanh, output_activation: sigmoid* (MSE: 0.0062).

Каждая из моделей обучается в течение 30 эпох с параметром *batch_size* равным 128.

Результаты качества прогнозирования для перечисленных моделей представлены в Таблице 3.2.4 как для всего набора данных, так и для отдельных признаков. Мы также сравниваем глубокие модели с наивным методом прогнозирования, когда последнее регистрируемое значение характеристики остается неизменным в течении всего горизонта прогноза. В качестве показателей качества для оценки модели используются средняя квадратичная ошибка (MSE), корень из MSE (RMSE), средняя абсолютная ошибка (MAE) и коэффициент детерминации (R^2). Можно отметить, что модель GRU дает немного лучшие результаты, чем остальные модели, и все глубокие модели прогнозирования превосходят в качестве наивный метод.

На рисунке 3.2.2 показана часть прогнозируемого временного ряда для каждой модели, а именно одно окно прогнозирования (50 секунд прогноза основаны на 50 секундах прошлых данных). Синим цветом обозначены входные данные (Inputs), зеленым – реальные

значения (True), серым – наивное прогнозирование (Naive). Для каждой модели дается обозначение, связанное с типом блока, количеством юнитов и размером дропаута без плавающей запятой.

Таблица 3.2.4. Результаты оценки качества прогнозирования состояний на наборе данных Gearbox Fault Diagnosis.

Модель	Показатель	Все признаки	a1	a2	a3	a4
GRU	MSE	**0,0057**	0,0039	0,0069	0,0061	0,0060
	RMSE	**0,0752**	0,0626	0,0832	0,0780	0,0772
	MSE	**0,0579**	0,0479	0,0639	0,0601	0,0597
	R2	**0,3206**	0,3352	0,2266	0,3221	0,3986
LSTM-1	MSE	0,0059	0,0041	0,0070	0,0063	0,0062
	RMSE	0,0764	0,0640	0,0839	0,0791	0,0786
	MSE	0,0588	0,0490	0,0645	0,0611	0,0607
	R2	0,2987	0,3038	0,2125	0,3018	0,3767
LSTM-2	MSE	0,0061	0,0043	0,0072	0,0065	0,0065
	RMSE	0,0778	0,0653	0,0848	0,0805	0,0806
	MSE	0,0599	0,0501	0,0652	0,0622	0,0623
	R2	0,2734	0,2757	0,1952	0,2776	0,3449
Наивный метод	MSE	0,0169	0,0118	0,0180	0,0181	0,0197
	RMSE	0,1295	0,1088	0,1341	0,1347	0,1403
	MSE	0,1001	0,0839	0,1040	0,1039	0,1085
	R2	-1,0071	-1,0108	-1,0105	-1,0237	-0,9836

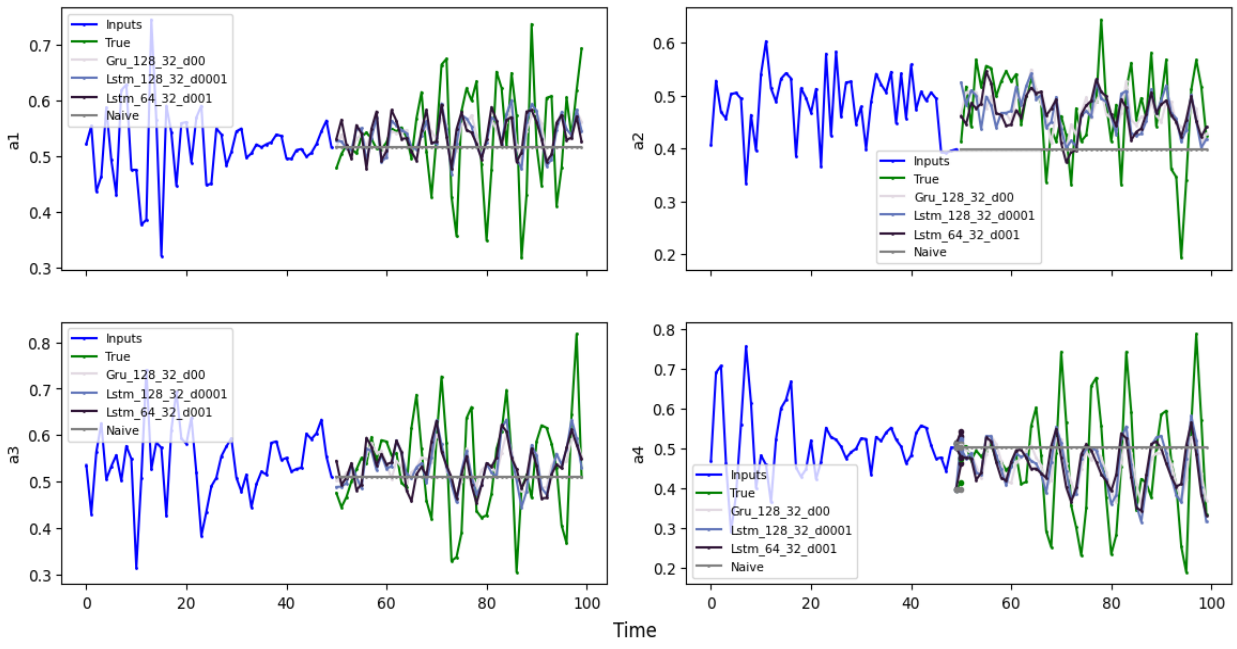

Рисунок 3.2.1 – Пример прогнозирования для набора данных Gearbox Fault Diagnosis

3.2.5 Задача № 5

В рамках данного примера решается задача *оценивания* и текущего состояния и *прогнозирования* будущих состояний на основе данных устройства индустриальной системы, а именно асинхронного двигателя переменного тока. В данном случае под состоянием понимается метка класса состояния устройства. Целью эксперимента является оценивание безопасности состояния устройств и определение будущих характеристик работы двигателя в стабильном режиме работы.

Исходные данные:

Экспериментальным набором данных является Condition Monitoring Dataset (AI4I 2021)[11]. Набор данных был получен мониторинга состояния асинхронного двигателя переменного тока, работающего от однофазного переменного тока 230 В 50 Гц. Всего двигатель может работать в восьми состояниях: (1)двигатель выключен (off); (2) двигатель работает от переменного тока 50 Гц (on); (3) во время работы двигателя конденсатор отключен (cap); (4) выпускной клапан компрессора вручную сужен (out); (5) винт вставлен с одной стороны вала для создания дисбаланса (unb); (6) незначительное засорение корпуса вентилятора (c25); (7) значительное засорение корпуса вентилятора (c75); (8) замена вентилятора на неисправный (vnt).

Для оценивания состояния используются 169 частотных характеристик, полученные кратковременным преобразованием Фурье. Для прогнозирования характеристик используются данные временных рядов как в необработанном (_raw.csv), так и гармонизированном виде (_hrm.csv). Пример отображения временного ряда для состояния off в гармонизированном виде представлено на Рисунке 3.2.3.

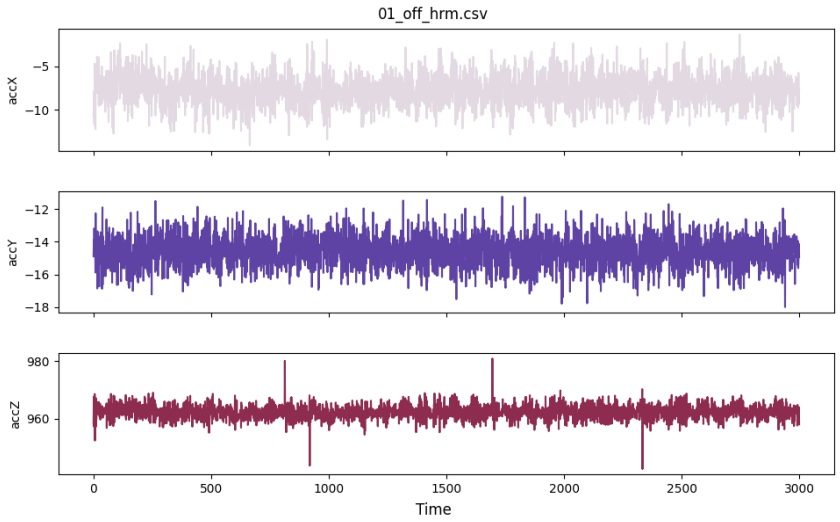

Рисунок 3.2.3 –Временной ряд AI4I 2021 в состоянии off в гармонизированном виде

[11] https://www.kaggle.com/datasets/stephanmatzka/condition-monitoring-dataset-ai4i-2021

Решение задачи:

Предобработка данных для оценивания включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [-1, 1] (*StandardScaler*). Разделение набора данных на тренировочнуюи тестовую выборки осуществляется соотношением 75:15.

В качестве моделей оценивая использовались следующие (приведенные параметры получены с использованием оптимизации с помощью бибилиотеки optuna):

– *DeepCNN* с параметрами *blocks*=1, *units*=96;

– *Hybrid_CNN_GRU* с параметрами *units*=32;

– *hybrid_variation* с параметрами *block*="*residual*", *units*=64, *loop_number*=1;

– *hybrid_variation* с параметрами *block*="*Xception*", *units*=64, *loop_number*=1.

Модели обучаются с использованием 50 эпох и параметром *batch_size* равным 128. В качестве показателей качества для оценки модели используются точность (precision, P), полнота (recall, R) и F-мера (F1). Результаты качества оценивания состояний устройства для перечисленных моделей представлены в Таблице 3.2.5.

Таблица 3.2.5. Результаты оценивания состояния на наборе AI4I 2021

	DeepCNN			***Hybrid_CNN_GRU***			***residual hybrid_variation***			***Xception hybrid_variation***		
Состояние	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
off	0,88	0,94	0,91	0,25	0,03	0,06	0,90	0,95	0,92	**0,94**	**0,95**	**0,94**
on	0,92	0,92	0,92	0,23	0,33	0,27	0,95	0,92	0,94	**0,94**	**0,97**	**0,95**
cap	**1,00**	**1,00**	**1,00**	0,22	0,21	0,21	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
out	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
unb	**1,00**	**1,00**	**1,00**	0,31	0,60	0,40	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
c25	**1,00**	**1,00**	**1,00**	0,28	0,43	0,34	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**	**1,00**
c75	0,98	0,94	0,96	0,40	0,23	0,29	1,00	0,97	0,98	**1,00**	**0,95**	**0,98**
vnt	1,00	0,98	0,99	0,20	0,10	0,13	1,00	1,00	1,00	**1,00**	**1,00**	**1,00**

Можно отметить, что модель *Xception hybrid_variation* дает ненамного лучшие результаты, но в то же время все модели, помимо *Hybrid_CNN_GRU*, демонстрируют сравнительно высокие результаты между собой. Среди отдельных классов состояний более высокие оценки получены для состояний out, cap, unb и c25. Также для всех моделей, кроме *Hybrid_CNN_GRU*, достигается поставленное требование с точности не менее 0,7 на малых выборках данных.

Предобработка данных для прогнозирования включает в себя нормализацию выборок с использованием масштабирования значений признаков на отрезок [0, 1] (*MinMaxScaler*). Разделение набора данных на тренировочную, валидационную и тестовую выборки осуществляется соотношением 70:10:20 для каждого из 16 временных рядов (2 для каждого состояния – в гармонизированном виде и нет).

В данном эксперименте длина исторической последовательности равно 50 секундам, а горизонт прогнозирования – 5 секундам. Таким образом можно оценить прогнозирование по отдельным окнам. Для оптимизации гиперпараметров глубокой модели прогнозирования устанавливаться следующие множества значений:

– *block_type* = {'*LSTM*', '*GRU*'},

– *n_rec_layers* = {1,2}.

– *units*_0 = {64, 32},

– *units*_1 = {24, 16},

– *dropout* = {0.0, 0.01, 0.1},

– *hidden_activation* = {'*tanh*', '*relu*'},

– *output_activation* = {'*linear*', '*sigmoid*'}.

В качестве алгоритма поиска используется Байесовская оптимизация. В качестве показателя качества во время оптимизации гиперпараметров прогнозирования используется средняя квадратичная ошибка (MSE).

В результате для каждого временного ряда определена лучшая модель прогнозирования Полученные параметры представлены в Таблице 3.2.6.

Таблица 3.2.6. Параметры архитектур модели прогнозирования для набора AI4I 2021

Временной ряд	***n_rec_layers***	***units***	***block_type***	***dropout***	***hidden_activation***	***output_activation***
01_off_hrm.csv	2	{64, 24}	*LSTM*	0,001	*relu*	*sigmoid*
01_off_raw.csv	2	{64, 16}	*GRU*	0,001	*relu*	*sigmoid*
02_on_hrm.csv	2	{32, 24}	*LSTM*	0,0	*tanh*	*linear*
02_on_raw.csv	1	64	*LSTM*	0,0	*relu*	*sigmoid*
03_cap_hrm.csv	2	{32, 16}	*LSTM*	0,001	*relu*	*sigmoid*
03_cap_raw.csv	1	64	*LSTM*	0,01	*tanh*	*sigmoid*
04_out_hrm.csv	2	{64, 16}	*LSTM*	0,001	*tanh*	*linear*
04_out_raw.csv	2	{32, 24}	*GRU*	0,001	*relu*	*linear*
05_unb_hrm.csv	2	{32, 16}	*GRU*	0,001	*tanh*	*sigmoid*
05_unb_raw.csv	2	{64, 16}	*GRU*	0,001	*tanh*	*sigmoid*
06_c25_hrm.csv	1	32	*LSTM*	0,001	*tanh*	*sigmoid*
06_c25_raw.csv	1	64	*GRU*	0,01	*tanh*	*linear*
07_c75_hrm.csv	2	{64, 16}	*GRU*	0,001	*tanh*	*sigmoid*
07_c75_raw.csv	1	64	*LSTM*	0,0	*relu*	*linear*
08_vnt_hrm.csv	2	{32, 24}	*GRU*	0,001	*tanh*	*linear*
08_vnt_raw.csv	1	32	*LSTM*	0,0	*tanh*	*sigmoid*

Каждая из моделей обучается в течение 100 эпох с параметром *batch_size* равным 32.

Результаты качества прогнозирования для перечисленных моделей и наивного метода прогнозирования представлены в Таблице 3.2.7. В качестве показателей качества для оценки модели используются средняя квадратичная ошибка (MSE), корень из MSE (RMSE) и средняя абсолютная ошибка (MAE).

	Модель прогнозирования			**Наивный метод**		
Временной ряд	**MSE**	**MAE**	**RMSE**	**MSE**	**MAE**	**RMSE**
01_off_hrm.csv	0,0138	0,0883	0,112	0,028	0,1257	0,1592
01_off_raw.csv	0,0132	0,0835	0,105	0,0304	0,1252	0,160
02_on_hrm.csv	**0,009**	**0,0727**	**0,0933**	0,0792	0,2195	0,2704
02_on_raw.csv	**0,011**	**0,0793**	**0,1024**	0,0783	0,2153	0,2661
03_cap_hrm.csv	0,0138	0,0895	0,1143	0,0742	0,2152	0,2650
03_cap_raw.csv	0,0136	0,0877	0,1143	0,0779	0,2168	0,2692
04_out_hrm.csv	0,0108	0,0798	0,1014	0,0822	0,2268	0,2771
04_out_raw.csv	0,0116	0,0833	0,107	0,0778	0,2155	0,2671
05_unb_hrm.csv	**0,0086**	**0,0703**	**0,0903**	0,0795	0,2276	0,2783
05_unb_raw.csv	**0,0088**	**0,0713**	**0,0922**	0,0765	0,2194	0,2702
06_c25_hrm.csv	**0,0093**	**0,0725**	**0,0936**	0,0775	0,2207	0,2713
06_c25_raw.csv	**0,0103**	**0,0775**	**0,0999**	0,0750	0,214	0,2655
07_c75_hrm.csv	0,0106	0,0787	0,1011	0,0759	0,2178	0,2673
07_c75_raw.csv	0,0106	0,0794	0,1019	0,0726	0,2063	0,2556
08_vnt_hrm.csv	**0,0098**	**0,0732**	**0,0941**	0,0925	0,2455	0,2987
08_vnt_raw.csv	**0,0097**	**0,0734**	**0,0957**	0,0925	0,2424	0,2966

Можно отметить, что модели для данных состояний unb, vnt, c25 и on дают немного лучшие результаты, чем остальные модели, и все глубокие модели прогнозирования превосходят в качестве наивный метод. Так для самой лучшей модели (05_unb_hrm.csv) значение MSE ниже в 10 раз, чем для наивного прогноза, а для модели с худшим показателем MSE (01_off_hrm.csv) – в 2 раза.

На Рисунке 3.2.4 показана часть прогнозируемого временного ряда ‘05_unb_hrm.csv’. Зеленым обозначены реальные значения признака (True), серым – наивное прогнозирование (Naive). Для каждой модели дается обозначение, связанное с типом блока, количеством юнитов и размером дропаута без плавающей запятой.

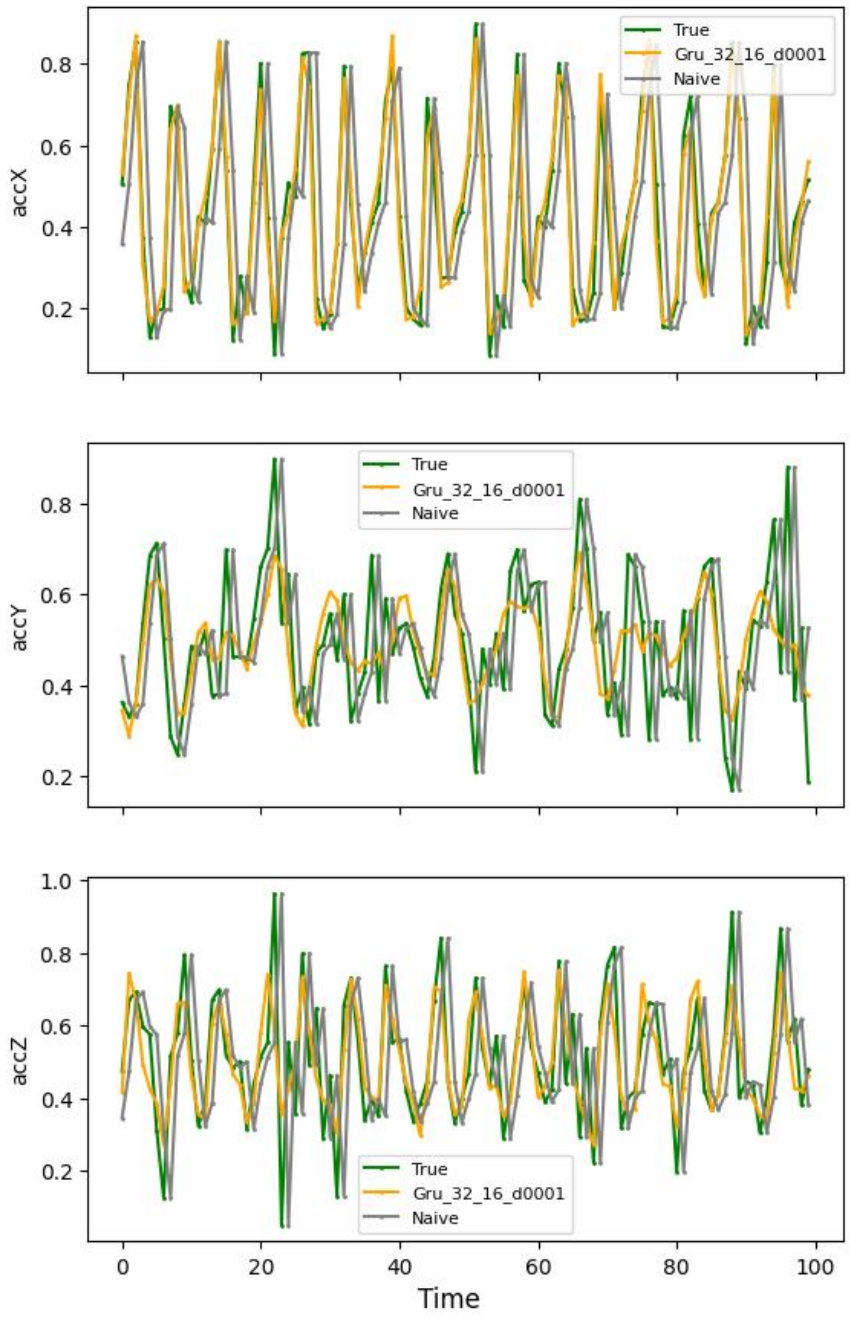

Рисунок 3.2.4 – Пример прогнозирования для набора данных AI4I 2021

4. ХАРАКТЕРИСТИКА ПРОГРАММЫ

4.1 Режимы работы ключевых алгоритмов

Разработанная программный компонент содержит четыре алгоритма – ПОИНД, ИЗСНД, АОССОП и АПССОП. При этом АОССОП и АПССОП являются ключевыми, а ПОИНД и ИЗСНД – вспомогательными. Более подробно принцип работы ключевых алгоритмов описан в Описании программы RU.СНАБ.00853-02 13 12 и статье, опубликованной в ведущем рецензируемом научном журнале[12].Эксперименты по проверке работоспособности и устойчивости результатов работы ключевых алгоритмов компонента сильного ИИ рассмотрены более подробно в следующих разделах.

Отметим, что в рамках определения характеристик работы ключевых алгоритмов компонента под «устойчивостью» результатов понимается их способность не менять свою адекватность (т. е. степень соответствия реальному объекту) при изменении параметров входных данных. На устойчивость напрямую влияет близость структур внутренних моделей алгоритмов (в данном случае по предобработке, оцениванию и прогнозированию) к структуре реального объекта, а также общая детализация элементов и их связей. Т. е. при малых возмущениях входных параметров результаты моделирования должны осуществлять незначительные колебания около точек равновесия. И наоборот, в случае удаления при малом входном возмущении результатов моделирования от равновесных состояний можно говорить о неустойчивости модели. Таким образом, для проверки устойчивости результатов необходимо проверить степень их отклонения при небольших вариациях входных данных. В ином случае могут наблюдаться не только некоторые отклонения в продуцируемых результатах, но и получение принципиально неверных решений.

Исходя из этого, экспериментальная проверка устойчивости результатов алгоритмов компонента может быть произведена по следующему многошаговому алгоритму.

Шаг 1. На всем входном диапазоне необходимо выбрать некоторый набор базовых входных данных, соответствующий максимальному числу режимов функционирования моделируемой системы (включая краевые точки).

Шаг 2. Получить с помощью алгоритмов результаты оценивания и прогнозирования для входных данных из Шага 1.

---

[12] Levshun D., Kotenko I. A survey on artificial intelligence techniques for security event correlation: models, challenges, and opportunities // Artificial Intelligence Review. 2023. P. 1–44. https://doi.org/10.1007/s10462-022-10381-4

Шаг 3. Произвести ряд небольших изменений входных данных из Шага 1 – получить тем самым набор вариативных входных данных.

Шаг 4. Для каждой вариации входных данных из Шага 3 получить собственные результаты оценивания и прогнозирования.

Шаг 5. Сравнить близость результатов работы алгоритмов, полученных для наборов вариативных входных данных – из Шага 4, с аналогичными результатами, но для наборов базовых входных данных – из Шага 1.

Шаг 6. Существенные отличия наборов вариативных и базовых входных данных, полученные на Шаге 5, будут сигнализировать о неустойчивости модели, лежащих в основе алгоритмов, и получаемых с помощью нее результатов. В ином случае, можно говорить об устойчивости результатов.

4.1.1 Алгоритм АОССОП

Оценка устойчивости проводилась на наборе данных Smart Crane Data (см. Пример 1 п. 3.2). К случайным 20% образцам из исходного экспериментального набора данных был добавлен случайный шум. Случайный шум генерируется с использованием нормального распределения со средним значением 0 и стандартным отклонением, основанным на стандартном отклонении выбранного признака исходных данных. Затем шум добавляется к указанным выборкам для случайно выбранных признаков для изменения данных.

В Таблице 4.1.1 представлены результаты оценки качества оценивания состояний набора данных Smart Crane Data, как для исходных данных, так для случая зашумления обучающих данных и зашумления тестовых данных. В качестве показателя качества используется аккуратность классификации циклов движения крана (ACC). Для зашумленных данных вычисляется разница между оценками на исходных данных и на зашумленных (Δ). На непосредственную работу алгоритма оценивания состояний зашумление данных признаков никак не влияет.

Таблица 4.1.1. Оценка устойчивости алгоритма АОССОП на наборе Smart Crane Data

Модель	**Исходные данные**	**Зашумление обучающей выборки**		**Зашумление тестовой выборки**	
	ACC	ACC	Δ	ACC	Δ
DeepCNN	0,9054	0,8571	0,0483	0,8699	0,0355
CNN-GRU	0,901	0,8834	0,0176	0,8692	0,0318
CNN Residual	0,9141	0,8883	0,0258	0,8724	0,0417
CNN Xception	0,9051	0,8811	0,024	0,8706	0,0345

Таким образом алгоритм оценивания соответствует требованию, что результат на зашумленных данных не должен понижаться более, чем на 10% точности (0.1).

4.1.2 Алгоритм АПССОП

Оценка устойчивости проводилась на наборе данных Gearbox Fault Diagnosis (см. Пример 4 п. 3.2). К случайным 20% образцам из исходного экспериментального набора данных был добавлен случайный шум. Случайный шум генерируется с использованием нормального распределения со средним значением 0 и стандартным отклонением, основанным на стандартном отклонении выбранного признака исходных данных. Затем шум добавляется к указанным выборкам для случайно выбранных признаков для изменения данных.

В Таблице 4.1.2 представлены результаты оценки качества прогнозирования состояний набора данных Gearbox Fault Diagnosis, как для исходных данных, так для случая зашумления обучающих данных и зашумления тестовых данных. В качестве показателя качества используется средняя квадратичная ошибка прогнозирования вибраций коробки передач (MSE). Для зашумленных данных вычисляется разница между оценками на исходных данных и на зашумленных в виде доли повышения ошибки прогнозирования (ε). На непосредственную работу алгоритма прогнозирования состояний зашумление данных признаков никак не влияет.

Таблица 4.1.2. Оценка устойчивости алгоритма прогнозирования состояний на наборе данных Gearbox Fault Diagnosis

Модель	**Показатель**	**Исходные данные**	**Зашумление обучающей выборки**	**Зашумление тестовой выборки**
GRU	MSE	0,0057	0,0059	0,0061
	ε	–	0,052	0,078
LSTM-1	MSE	0,0059	0,0062	0,0064
	ε	–	0,063	0,084
LSTM-2	MSE	0,0061	0,0065	0,0066
	ε	–	0,064	0,086

Таким образом алгоритм прогнозирования соответствует требованию, что результат на зашумленных данных не должен повышаться выше, чем на 10% (0.1).

4.2 Порядок оценки качества алгоритмов

Используемые метрики оценки и их обоснование:

Для апостериорной и априорной проверки качества алгоритмов используются два показателя. Для алгоритма АОССОП – аккуратность оценивания состояния СлОП:

$$\text{a}ccuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

где *TP* – количество правильно определенных состояний к правильному классу, *TN* – количество неправильно определенных состояний к правильному классу, *FP* – количество неправильно определенных состояний к правильному классу, FN – количество неправильно определенных состояний к неправильному классу.

Данная метрика является наиболее распространенной для оценки моделей и интуитивно понятной (доля верно определенных экземпляров всех классов).

Для АПССОП используется средняя квадратичная ошибка прогнозирования СлОП. Показатели эффективности прогнозирования основаны на вычислении разницы между реальными значениями образца (*X*) длиной *N* и прогнозируемыми значениями образца (*X*’):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(X_i - X_i')^2.$$

Выбор данного показателя обоснован тем, что он позволяют оценить отклонение прогнозируемых значений от реальных: чем ближе значение к 0, тем меньше отклонение.

Под априорной оценкой понимается оценка качества моделей на тех же данных, на которых была обучена модель (обучающая выборка). Под априорной оценкой понимается оценка качества моделей на данных, на которых модель не обучалась (тестовая выборка).

Шаги методики оценки:

1) загрузка обученных моделей прогнозирования и нормализации данных;

2) разделение исходного набора данных на тренировочную и тестовую выборки с заданным соотношением;

3.1) для алгоритма АОССОП: нормализация выборок с использованием стандартного отклонения и масштабирования значений признаков на отрезок [-1, 1];

3.2) для алгоритма АПССОП: нормализация выборок с использованием максимальных и минимальных значений и масштабирования значений признаков на отрезок [0, 1];

4.1) для алгоритма АОССОП: разделение признаков и меток данных;

4.2) для алгоритма АПССОП: генерация временных рядов;

5) обучение моделей и вывода значений целевых показателей во время обучения;

6) вычисление показателей качества алгоритма на тестовой выборке.

Эксперименты по априорной и апостериорной оценке качества работы ключевых алгоритмов компонента сильного ИИ приведены далее.

1) Алгоритм АОССОП

В табл. 4.2.1 представлены результаты оценки алгоритма АОССОП на наборе данных. Априорная оценка выполнялась на обучающей выборке, что представляло собой

67% выборку от исходного набора данных. Апостериорная оценка выполнялась на тестовой выборке, включающей оставшуюся выборку.

Таблица 4.2.1 – Априорная и апостериорная оценки алгоритма АОССОП

Набор данных	**Количество классов**	**Модель**	**Оценка**	**Значение точности (accuracy)**
Smart Crane Data	8	DeepCNN	Априорная	0,95
			Апостериорная	0,91
		Hybrid_CNN_GRU	Априорная	0,99
			Апостериорная	0,90
		residual hybrid_variation	Априорная	0,99
			Апостериорная	0,91
		Xception hybrid_variation	Априорная	0,99
			Апостериорная	0,91
Edge-IIoT	15	DeepCNN	Априорная	0,99
			Апостериорная	0,98
		Hybrid_CNN_GRU	Априорная	0,99
			Апостериорная	0,98
		residual hybrid_variation	Априорная	0,99
			Апостериорная	0,98
		Xception hybrid_variation	Априорная	0,99
			Апостериорная	0,96

2) Алгоритм АПССОП

В табл. 4.2.2 представлены результаты оценки алгоритма АПССОП на наборе данных. Априорная оценка выполнялась на обучающей выборке, что представляло собой 70% выборку от исходного набора данных. Апостериорная оценка выполнялась на тестовой выборке, включающей оставшуюся выборку. В качестве показателя качества используется средняя квадратичная ошибка (MSE).

Таблица 4.2.2 – Априорная и апостериорная оценки алгоритма АПССОП

Набор данных	**Модель**	**Оценка**	**MSE**
SWAT	GRU-1	Априорная	0,0075
		Апостериорная	0,0116
	GRU-2	Априорная	0,0046
		Апостериорная	0,0060
	GRU-3	Априорная	0,0055
		Апостериорная	0,0097
Gearbox Fault Diagnosis dataset	GRU	Априорная	0,0054
		Апостериорная	0,0053
	LSTM-1	Априорная	0,0056
		Апостериорная	0,0055
	LSTM-2	Априорная	0,0059
		Апостериорная	0,0057

5. ОБРАЩЕНИЕ К ПРОГРАММЕ

5.1 Точки входа в программу

Обращение к программе происходит путем создания объектов классов и вызова их них необходимых функций.

Алгоритм **ПОИНД**

Основными обращениям к программе через вызовы функций модуля ПОИНД являются следующие:

1. из состава класса *CheckDataTypes*: *correct_types* – алгоритм, объединяющий определение типа данных (1) по названию признака, (2) на основе анализа количества уникальных значений признака и (3) на основе анализа наличия float значений у признак, а также (4) на основе веса решений отдельных алгоритмов в единый процесс работы с признаками данных;
2. из состава класса *ClusterFilling*: *fill* – алгоритм, применяющий кластеризацию на основе центройдов или кластеризацию на основе расчета статистик на основе решения пользователя или значения по умолчанию;
3. из состава класса *Informativity*: *calculate_informativity* – алгоритм анализа информативности признаков данных;
4. из состава класса *MultiCollinear*: *remove_uninformative_features* – алгоритм устранения мультиколлинеарности данных.

Алгоритм ИЗСНД

Основными обращениям к программе через вызовы функций модуля ИЗСНД являются следующие:

1. из состава класса *RulesExtractor*:
    a. *__init__* – инициализация модели алгоритма ИЗСНД путем указания порога для построения предикатов;
    b. *fit()* – обучение модели алгоритма;
    c. *get_rules()* – вывод построенных правил в консоль в строковом виде;
    d. *transform()* – преобразование набора данных с использованием построенных правил к бинарный формат.

Алгоритм АОССОП

Обращение к программе происходит путем создания объектов классов и вызова их них необходимых функций. Основными обращениям к программе через вызовы функций модуля АОССОП являются следующие:

1. из состава класса *FormatDetector*:
   a. *__init__* – конструктор класса.
2. из состава класса *DataLoader*:
   a. *load* – загрузка данных,
   b. *__init__* – конструктор класса.
3. из состава класса *FeatureSelector*:
   a. *fit_transform* – подобрать метод выбора признаков к входным данным и преобразовать их,
   b. *feature_selection* – выполнить выбор признаков на основе указанного метода.
4. из состава класса *SAIClassifier*:
   a. *fit* – обучение модели;
   b. predict – прогнозирование класса состояния; save – сохранение модели во внешний файл;
   c. *load* – загрузка модели из внешнего файла;
   d. __init__ – конструктор класса.
4. из состава класса *DeepCNN*:
   a. *build_model* – построение модели классификации;
   b. *fit* – обучение модели;
   c. *test* – оценка качества модели на тестовых данных;
   d. *predict* – прогнозирование меток состояний объектов;
   e. *print_classification_report* – отображение показателей качества классификации;
   f. *draw_plot* – отображение графика процесса обучения с точностью или с потерями;
   g. *save_model* – сохранение модели во внешний файл;
   h. *load_model* – загрузка модели из внешнего файла;
   i. *__init__* – конструктор класса.
5. из состава класса *Hybrid_CNN_GRU*:
   a. *build_model* – построение модели классификации;
   b. *print_summary* – отображение сводной информации об архитектуре модели;

c. *fit* – обучение модели;
d. *test* – оценка качества модели на тестовых данных;
e. *predict* – прогнозирование меток состояний объектов;
f. *print_classification_report* – отображение показателей качества классификации;
g. *draw_plot* – отображение графика процесса обучения с точностью или с потерями;
h. *save_model* – сохранение модели во внешний файл;
i. *load_model* – загрузка модели из внешнего файла;
j. *__init__* – конструктор класса.

6. из состава класса *hybrid_variation*:
   a. *build_model* – построение модели классификации;
   b. *print_summary* – отображение сводной информации об архитектуре модели;
   c. *fit* – обучение модели;
   d. *test* – оценка качества модели на тестовых данных;
   e. *predict* – прогнозирование меток состояний объектов;
   f. *print_classification_report* – отображение показателей качества классификации;
   g. *draw_plot* – отображение графика процесса обучения с точностью или с потерями;
   h. *save_model* – сохранение модели во внешний файл;
   i. *load_model* – загрузка модели из внешнего файла;
   j. *__init__* – конструктор класса.
7. из состава класса *AutoEncoder*:
   a. *define_model* – определить структуру модели автокодировщика на основе указанного типа модели;
   b. *add_classifier* – добавить слой классификации;
   c. *cnn_ae* – установить CNN в качестве кодера и декодера;
   d. *dnn_ae* – установить DNN в качестве кодера и декодера;
   e. *rnn_ae* – установить RNN в качестве кодера и декодера;
   f. *lstm_ae* – установить LSTM в качестве кодера и декодера;
   g. *bilstm_ae* – установить Bi-LSTM в качестве кодера и декодера;
   h. *gru_ae* – установить GRU в качестве кодера и декодера;
   i. *viz_model* – визуализация архитектуры модели и сохранение в файл;

j. *fit* – обучение модели;
k. *test* – оценка качества модели на тестовых данных;
l. *predict* – прогнозирование меток состояний объектов;
m. *draw_mse_plot* – отображение графика процесса обучения с потерями;
n. *draw_clf_plot* – отображение графика процесса обучения с точностью классификации;
o. *__init__* – конструктор класса.

8. из состава класса *GAN*:
   a. *build_generator* – создать генератор с помощью указанной модели автокодировщика;
   b. *build_discriminator* – создать дискриминатор для классификации;
   c. *build_gan* – построить модель GAN, объединив генератор и дискриминатор;
   d. *train* – обучение модели;
   e. *predict* – прогнозирование меток состояний объектов;
   f. *print_classification_report* – отображение показателей качества классификации;
   g. *draw_plot* – отображение графика процесса обучения с точностью или с потерями;
   h. *__init__* – конструктор класса.
9. из состава класса *ClsEstimator*:
   a. *estimate* – оценка качества классификаторов;
   b. *__init__* – конструктор класса.

Алгоритм АПССОП

Обращение к программе происходит путем создания объектов классов и вызова их них необходимых функций. Основными обращениям к программе через вызовы функций модуля АПССОП являются следующие:

1. из состава класса *ForecasterParameters*:
   a. *to_dict* – преобразование в словарь;
   b. *from_dict* – преобразование из словаря;
   c. *read_json* – загрузка параметров из JSON файла;
   d. *save_json* – сохранение параметров в JSON файл;
   e. *__init__* – конструктор класса.

2. из состава класса *DeepForecasterParameters:*
   a. *__init__* – конструктор класса.
3. из состава класса *TSGenerator*:
   a. *change_horizon* – изменить горизонт прогнозирования;
   b. *get_data* – вернуть входные данные временных рядов;
   c. *get_targets* – вернуть выходные данные временных рядов;
   d. *__init__* – конструктор класса.
4. из состава класса *NaiveForecaster*:
   a. *forecasting* – прогнозирование заданной длины;
   b. *__init__* – конструктор класса.
5. из состава класса *DeepForecaster* :
   a. *forecasting* – прогнозирование заданной длины;
   b. *load_from_file* – загрузить модель из файла;
   c. *load_from_model_config* – определить модель через конфигурацию;
   d. *_set_model_params_from_config* – установить параметры модели из атрибута конфигурации;
   e. *save_model_config* – сохранить конфигурацию модели в файл;
   f. *save_model* – сохранить модель в файл;
   g. *build_model* – формирование модели;
   h. *summary* – описание модели;
   i. *train* – обучение модели;
   j. *__init__* – конструктор класса.
6. из состава класса *DeepForecasterTuner*:
   a. *set_tuned_hps* – установить множество возможных значений гиперпараметров модели прогнозирования;
   b. *build_hypermodel* – простроить гипермодель для оптимизации;
   c. *search_space_summary* – вывести сводку пространства поиска;
   d. *find_best_models* – определить ряд лучших моделей прогнозирования по заданному множеству конфигураций гиперпараметров;
   e. *get_best_deep_forecaster_parameters* – определить ряд лучших параметров моделей;
   f. *__init__* – конструктор класса.

7. из состава класса *ForecastEstimator*:

   a. *set_true_values* – установить истинные значения выходных данных временных рядов;

   b. *set_pred_values* – установить прогнозируемые значения выходных данных временных рядов для заданной модели прогнозирования;

   c. *set_first_batch* – установить значения первого входного временного окна;

   d. *estimate* – оценить качество моделей прогнозирования;

   e. *save_quality* – сохранить результаты оценки в файл;

   f. *save_pred_result* – сохранить прогнозируемые значения выходных данных временных рядов;

   g. *draw* – отобразить результаты прогнозирования для ряда признаков данных;

   h. *__init__* – конструктор класса.

5.2 Базовые функции

5.2.1 Модуль ПОИНД

Начало работы с набором данных:

```
# создание объекта данных
data = Data()
# передача пути к набору данных
titanic_path = '../datasets/titanic.csv'
# считывание данных в pd.Dataframe
titanic = pd.read_csv(titanic_path)
# сохранение наименований признаков данных
data.features_names = ["PassengerId", "Pclass", "Age", "SibSp", "Parch"]
# сохранение наименований меток данных
data.labels_names = ["Survived", "Fare"]
# сохранение матрицы признаков
data.features_matrix = np.array(titanic[data.features_names])
# сохранение матрицы меток
data.labels_matrix = np.array(titanic[data.labels_names])
# сохранение типов данных признаков
data.features_types = ["cat", "cat", "num", None, None]
# сохранение типов данных меток
data.labels_types = ["cat", None]
```

Подключение журналирования:

```
# подключение журналирования (True)
__Verbose__.PrintLog.instance().set_print_mode(True)
# задание степени подробности журналирования (status)
__Verbose__.PrintLog.instance().set_severity_level("status")
```

Запуск алгоритмов модуля:

```
# устранение некорректности типов данных
CheckDataTypes.CheckDataTypes.correct_types(data)
# устранение неполноты данных
ClusterFilling.ClusterFilling.fill(data)
# анализ информативности данных
Informativity.Informativity.calculate_informativity(data)
# устранение мультиколинеарности данных
Multicolinear.MultiCollinear.remove_uninformative_features(data)
```

5.1.2 Модуль ИЗСНД

Создание объекта *RulesExtractor*:

```
# probability_threshold = 0.1
algo = RulesExtractor(0.1)
```

Обучение модели:

```
# test_data: data to build predicates and rules for
# class_column: name of column with class labels
# positive_class_label: label for positive class
algo.fit(test_data,  class_column  =  class_column,  positive_class_label  =
positive_class_label)
```

5.1.3 Модуль АОССОП

Модель инициализируется путем создания объекта *SAIClassifier*, которому передается тип классификатора, количество нейронов на скрытых и выходном слоях:

```
SAIClassifier('neural_network', 10, 1)
```

Объект *FormatDetector* создается путем вызова соответствующего конструктора с наименованием файла, в котором содержится обрабатываемый набор данных:

```
FormatDetector('../hai/hai-20.07/test1.csv.gz')
```

Объект *DataLoader* создается путем вызова соответствующего конструктора с наименованием файла, в котором содержится обрабатываемый набор данных, количество полей в записи, разделитель полей в записи:

```
DataLoader('../hai/hai-20.07/test1.csv.gz', 64, ‘,’)
```

Объект *FeatureSelector* путем вызова соответствующего конструктора с наименованием метода выбора признаков и параметров для вызываемого метода.

```
featureselect = FeatureSelector(method='pca', params={"components":41})
```

Объект *ClsEstimator* создается путем вызова соответствующего конструктора с набором данных (векторы признаков, метки объектов) и типом классификатора:

```
classifiers = [SAIClassifier(c, in_size, out_size, plot=False)
               for c in classifier_types]
xor_ds = {'features': np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), 'labels':
np.array([[0], [1], [1], [0]])}
ClsEstimator(xor_ds['features'], xor_ds['labels'], xor_ds['labels'], [classifier]
```

Объект *DeepCNN* создается путем вызова соответствующего конструктора, которому передаются значения размера входных данных (*input_shape*), количества блоков нейронной сети (*blocks*), количества юнитов нейронной сети (*units*) и количество уникальных меток классов в данных (*classes*):

```
model = DeepCNN(input_shape=(10,1), blocks=2, units=64, classes=2)
```

Объект *Hybrid_CNN_GRU* создается путем вызова соответствующего конструктора, которому передаются значения размера входных данных (*input_shape*), количества блоков нейронной сети (*blocks*), количества юнитов нейронной сети (*units*) и количество уникальных меток классов в данных (*classes*):

```
model = Hybrid_CNN_GRU(input_shape=(10,1), units=64, classes=2)
```

Объект *hybrid_variation* создается путем вызова соответствующего конструктора, которому передаются значения размера входных данных (*input_shape*), количества блоков нейронной сети (*blocks*), количества юнитов нейронной сети (*units*), количество уникальных меток классов в данных (*classes*) тип блока сверточной нейронной сети (*block*) и количество раз применения выбранного блока (*loop_number*):

```
model = hybrid_variation(input_shape=(10,1), units=64,
                         classes=2, block="residual", loop_number=2)
```

Объект *AutoEncoder* создается путем вызова соответствующего конструктора, которому передаются значения размера входных данных (*input_shape*), тип нейронной сети для кодера и декодера (*model_type*), начальная размерность автокодировщика (*ae_start_dim*), необходимость включения слоя классификации (*classifier*) и количество уникальных меток классов в данных (*num_categories*):

```
model = AutoEncoder(input_shape=(10,1), model_type='dnn', ae_start_dim=64,
                    classifier=True, num_categories=2)
```

Объект *GAN* создается путем вызова соответствующего конструктора, которому передаются значения размера входных данных (*input_shape*), тип нейронной сети для кодера и декодера (*ae_model_type*), начальная размерность автокодировщика (*ae_start_dim*), глубина сверточных слоев для дискриминатора (*depth*) и количество уникальных меток классов в данных (*classes*):

```
model = GAN(input_shape=(10,1), ae_model_type='dnn', ae_start_dim=64, depth=3,
            classes=2)
```

5.1.4 Модуль АПССОП

Объект *ForecasterParameters* создается путем вызова соответствующего конструктора, которому передаются значения количества признаков (*n_features)*, число временных шагов до прогноза (*look_back_length)* и горизонт прогнозирования во временных шагах (*horizon):*

```
model_params = ForecasterParameters(4, 100, 10)  # Установка значений параметров
model_params.look_back_length = 200  # Изменения значений параметров
model_params.read_json('params.json') # Загрузка параметров из внешнего файла
model_params.from_dict({look_back_length=150, horizon=15}) # Загрузка из словаря
```

Объект *DeepForecasterParameters* создается путем вызова соответствующего конструктора, которому передаются значения количества признаков (*n_features)*, число временных шагов до прогноза (*look_back_length)* и горизонт прогнозирования во временных шагах (*horizon)*, число юнитов на слоях нейронной сети (*units)*, тип блока RNN (*block_type*), доля отбрасывания нейронов (*dropout*), функция активации на скрытом слое (*hidden_activation*) , функция активации на выходном слое (*output_activation*), функция потерь (*loss*)*:*

```
model_params = DeepForecasterParameters(n_features=4, look_back_length=100,
                                units=[256, 128], block_type = 'LSTM',
                                dropout = 0.1, hidden_activation = 'tanh',
                                output_activation = 'linear', loss = 'mse',
                                optimizer = 'adam')
model_params.block_type = 'GRU'  # Изменения значений параметров
model_params.read_json('params.json') # Загрузка параметров из внешнего файла
model_params.from_dict({block_type = 'GRU', dropout = 0.01}) # Загрузка из словаря
```

Объект *TSGenerator* создается путем вызова соответствующего конструктора, которому передаются значения входных данных временной последовательности (*x*) и параметры модели прогнозирования (*model_params*) или число временных шагов до прогноза (*look_back_length)* и горизонт прогнозирования во временных шагах (*horizon)*:

```
x = np.random.rand(1000, 2)
model_params = ForecasterParameters(2, 100, 1)
ts = TSGenerator(x, model_params)
ts = TSGenerator(x, look_back_length=100, horizon=1)
```

Модель наивного прогнозирования инициализируется путем создания объекта *NaiveForecaster*, которому передаются параметры модели прогнозирования (*model_params*):

```
model = NaiveForecaster(model_params)
```

Модель глубокого прогнозирования инициализируется путем создания объекта *DeepForecaster*, которому передаются параметры модели прогнозирования (*model_params*) или имеющаяся нейронная сеть (*model*) или путь к модели (*from_file*) или путь к конфигурации модели (*from_file*) или имеющаяся конфигурация модели (*from_config*):

```
# Инициализация модели прогнозирования через параметры
model_params = DeepForecasterParameters(4, 100, 10, block_type = 'GRU', units=[512])
model = DeepForecaster(model_params)
model = DeepForecaster(model=my_model) # Передача существующей модели
model = DeepForecaster(from_file='my_model.keras') # Загрузка модели из файла .keras
# Загрузка модели из JSON файла конфигурации
model = DeepForecaster(from_file_config='my_model_config.json)
# Загрузка модели из существующей конфигурации формата keras
model = DeepForecaster(from_file_config=my_model_config)
```

Обучение модели объекта *DeepForecaster* осуществляется путем вызова метода *train*, которому передаются значения входных временных окон (*x*) и выходных временных окон (*y*), а также число эпох обучения (*n_epochs*), размер пакетов данных для обучения (*batch_size*), переменная отображения процесса обучения (*verbose*), данные для валидации (*validation_data*) или доля от тренировочных данных для валидации (*validation_split*) и число эпох без улучшений для преждевременной остановки процесс обучения (*early_stop_patience*):

```
x = ts.get_data()
y = ts.get_targets()
model.train(x, y, n_epochs=10, batch_size=256, validation_data=0.1)
```

Прогнозирование данных осуществляется путем вызова метода forecasting объекта NaiveForecaster или DeepForecaster, которому передаются пакет данных для прогнозирования (current_batch) и длина прогнозируемой последовательности (forecasting_data_length):

```
pred = forecasting_model.forecasting(current_batch = x, forecasting_data_length = 5)
```

Оценка эффективности прогнозирования осуществляется путем создания объекта ForecastEstimator, передачи фактических значений признаков данных (*true*) и прогнозируемых значений признаков данных (*pred*) и выхода метода *estimate*:

```
estimator = ForecastEstimator()
estimator.set_true_values(y)
estimator.set_pred_values(pred, model_name='my_model')
estimator.estimate()
result = estimator.quality
```

5.3 Интеграция библиотеки в платформу проекта

Библиотека ПАОПС может использоваться как непосредственно, так и в рамках инструментальной облачной платформы для проектирования, быстрой разработки и обучения прикладных систем ИИ RU.СНАБ-00845 (DataMall 2.0).

Для интеграции библиотеки ПАОПС в платформе создается контейнер. Для сборки образа контейнера необходимо дополнительно подготовить файл Dockerfile с инструкциями, подробнее - https://docs.docker.com/engine/reference/builder/.

Образ должен соответствовать следующим требованиям:

а) должен быть установлен jupyter notebook или jupyter lab;

б) Entrypoint должен быть пустым;

в) CMD должен запускать jupyter;

г) пользователь, от имени которого запускается jupyter должен быть `jovyan`;

д) вся функциональность модуля должна быть доступна для пользователя `jovyan`;

е) должны быть добавлены / развернуты требуемое окружение и соответствующие программные библиотеки для функционирования основной библиотеки;

ж) при необходимости добавляются ноутбуки с примерами реализации функциональных возможностей.

6. ПРОВЕРКА ПРОГРАММЫ

6.1 Модульные и интеграционные тесты

Модульное и интеграционное тестирование элементов программы производилось с помощью библиотеки unittest, что является стандартным решения для языка Python. Разработанные тесты для всех модулей компонента приведены далее. При это при описании тестов приводятся фрагменты методов тестирования, которые являются элементами класса *unittest.TestCase.*

6.1.1 Модуль ПОИНД

Список тестов для модуля ПОИНД сведен в единую таблицу 6.1.1.

Таблица 6.1.1 – Тесты модуля ПОИНД

Номер теста	Назначение теста	Пройден ли тест
1	Проверка корректности исправления типов данных: запуск метода *test_check_data_types*	да
2	Проверка корректности устранения неполноты данных: запуск метода *test_cluster_filling*	да
3	Проверка корректности анализа информативности признаков данных: запуск метода *test_informativity*	да
4	Проверка корректности устранения мультиколлинеарности признаков данных: запуск метода *test_multicolinearity*	да

Модульный тест №1. Проверка корректности исправления типов данных (*test_check_data_types).*

Исходный код:

```
expected_features_types = ['cat', 'num', 'cat', 'cat', 'num']
CheckDataTypes.CheckDataTypes.correct_types(data)
self.assertEqual(data.features_types, expected_features_types)
```

Модульный тест № 2. Проверка корректности устранения неполноты данных (*test_cluster_filling*)

Исходный код:

```
expected_features_matrix = np.array(list(zip(*[
[1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
 [4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
 [4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
 [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 1, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2],
```

```
[4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
]))) 
ClusterFilling.ClusterFilling.fill(data)
self.assertTrue(np.array_equal(data.features_matrix, expected_features_matrix))
```

Модульный тест № 3. Проверка корректности анализа информативности признаков данных (*test_informativity*)

Исходный код:

```
expected_features_matrix = np.array(list(zip(*[
[1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
[4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 1, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2]
])))
Informativity.Informativity.calculate_informativity(data)
self.assertTrue(np.array_equal(data.features_matrix, expected_features_matrix))
```

Модульный тест № 4. Проверка корректности устранения мультиколлинеарности признаков данных (*test_multicolinearity*)

Исходный код:

```
expected_features_matrix = np.array(list(zip(*[
[1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
 [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 1, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2]
])))
Multicolinear.MultiCollinear.remove_uninformative_features(data)
self.assertTrue(np.array_equal(data.features_matrix, expected_features_matrix))
```

6.1.2 Модуль ИЗСНД

Список тестов для модуля ИЗСНД сведен в единую таблицу 6.1.2.

Таблица 6.1.2 – Тесты модуля ИЗСНД

Номер теста	Назначение теста	Пройден ли тест
1	Проверка корректности подсчета коэффициента регрессии: запуск метода *test_regression_coefficient*	да
2	Проверка корректности подсчета меры Клозгена: запуск метода *test_klosgen_measure*	да
3	Проверка корректности определения алгоритмом строковых признаков в данных: запуск метода *test_string_column*	да

4	Проверка корректности определения алгоритмом числовых признаков в данных: запуск метода *test_number_column*	да
5	Проверка корректности расчета статистик класса: запуск метода *test_class_stats*	да
6	Проверка корректности расчета статистик значений признаков: запуск метода *test_data_stats*	да
7	Проверка корректности извлечения правил из данных: запуск метода *test_rules*	да
8	Проверка корректности построения предикатов: запуск метода *test_predicates*	да

Модульный тест № 1. Проверка корректности подсчета коэффициента регрессии (*test_regression_coefficient*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The regression coefficient is calculated incorrectly'
self.assertEqual(round(RulesExtractor().calculate_regression_coefficient(nA=200,
nB=200, nAB=100, N=1000), 3), 0.375, message)
```

Модульный тест № 2. Проверка корректности подсчета меры Клозгена (*test_klosgen_measure*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The klosgen measure is calculated incorrectly'
self.assertEqual(round(RulesExtractor().calculate_klosgen_measure(nA=200, nB=200,
nAB=100, N=1000), 3), 0.134, message)
```

Модульный тест № 3.

Модульный тест № 3. Проверка корректности определения алгоритмом строковых признаков в данных (*test_string_column*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'String columns were identified incorrectly'
data = make_classification(n_samples=200, n_features=4, n_informative=2,
                           n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = '1')
self.assertEqual(len(algo.string_columns), 0, message)
```

Модульный тест № 4. Проверка корректности определения алгоритмом числовых признаков в данных (*test_number_column*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Numerical columns were identified incorrectly'
data = make_classification(n_samples=200, n_features=4, n_informative=2,
                           n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = '1')
self.assertEqual(len(algo.number_columns), 4, message)
```

Модульный тест № 5. Проверка корректности расчета статистик класса (*test_class_stats*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Class stats was calculated incorrectly'
data = make_classification(n_samples=200, n_features=4,
                           n_informative=2, n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = 1)
self.assertEqual(algo.class_stats[1], df['class'].value_counts()[1] , message)
```

Модульный тест № 6. Проверка корректности расчета статистик значений признаков (*test_data_stats*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Data stats was calculated incorrectly'
data = make_classification(n_samples=200, n_features=4,
                           n_informative=2, n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = 1)
self.assertEqual(len(algo.data_stats), 4, message)
```

Модульный тест № 7. Проверка корректности извлечения правил из данных (*test_rules*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Rules were extracted incorrectly
data = make_classification(n_samples=200, n_features=4,
                           n_informative=2, n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = 1)
rule = "if col1 in [Interval(1.032, 1.661, closed='right'), Interval(0.404, 1.032,
closed='right'), Interval(-0.224, 0.404, closed='right'), Interval(1.661, 2.289,
closed='right'), Interval(2.289, 2.917, closed='right')] then 1
(regression_coef=0.759)"
self.assertEqual(algo.get_rules()[0], rule, message)
```

Модульный тест № 8. Проверка корректности построения предикатов (*test_predicates*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Predicates is not checking data correctly'
data = make_classification(n_samples=200, n_features=4,
                           n_informative=2, n_classes=2, random_state=42)
df = pd.DataFrame(data[0], columns = ['col1','col2','col3','col4',])
df['class'] = pd.Series(data[1])
algo = RulesExtractor(0.2)
algo.fit(df, class_column = "class", positive_class_label = 1)
predicate = algo.predicates[0]
row = df[0]
self.assertEqual(predicate.is_true(df[:1]).iloc[0], 1, message)
```

6.1.3 Модуль АОССОП

Список тестов для модуля АОССОП сведен в единую таблицу 6.1.3.

Таблица 6.1.3 – Тесты модуля АОССОП

Номер теста	Назначение теста	Пройден ли тест
1	Создание модели оценивания (*test_sc_is_not_none*)	да
2	Инициализация модели оценивания (*test_sc_type_is_correct*)	да
3	Корректность модели оценивания (*test_sc_fit*)	да
4	Оценивание тестового набора (*test_sc_predict*)	да
5	Сохранение модели оценивания (*test_sc_save*)	да
6	Загрузка из файла (*test_sc_load*)	да
7	Проверка разделителя (*test_fd_is_not_none*)	да
8	Существование файла для загрузки (*test_fd_file_exists*)	да

9	Корректность разделителя (*test_fd_delimiter_is_correct*)	да
10	Инициализация объекта загрузки данных (*test_dl_is_not_none*)	да
11	Инициализация показателей эффективности (*test_ce_is_not_none*)	да
12	Вычисление показателей эффективности (*test_ce_estimate*)	да
13	Инициализация элемента класса GAN (*test_build_generator*)	да
14	Инициализация элемента класса GAN (*test_build_discriminator*)	да
15	Инициализация элемента класса GAN (*test_build_gan*)	да
16	Выполнение метода класса GAN (*test_train*)	да
17	Выполнение метода класса GAN (*test_print_classification_report*)	да
18	Выполнение метода класса GAN (*test_draw_plot*)	да
19	Выполнение метода класса DeepCNN (*test_model_build*)	да
20	Выполнение метода класса DeepCNN (*test_model_fit*)	да
21	Выполнение метода класса DeepCNN (*test_model_predict*)	да
22	Выполнение метода класса DeepCNN (*test_classification_report*)	да
23	Выполнение метода класса DeepCNN (*test_plot*)	да
24	Выполнение метода класса DeepCNN (*test_save_load_model*)	да
25	Выполнение метода класса DeepCNN (*test_load_model*)	да
26	Выполнение метода класса hybrid_variation (*test_build_model*)	да
27	Выполнение метода класса hybrid_variation (*test_fit*)	да
28	Выполнение метода класса hybrid_variation (*test_test*)	да
29	Выполнение метода класса hybrid_variation (*test_predict*)	да
30	Выполнение метода класса hybrid_variation (*test_print_classification_report*)	да
31	Выполнение метода класса hybrid_variation (*test_draw_plot*)	да
32	Выполнение метода класса hybrid_variation (*test_save_and_load_model*)	да
33	Выполнение метода класса Hybrid_CNN_GRU (*test_build_model*)	да
34	Выполнение метода класса Hybrid_CNN_GRU (*test_fit*)	да
35	Выполнение метода класса Hybrid_CNN_GRU (*test_test*)	да

36	Выполнение метода класса Hybrid_CNN_GRU (*test_print_classification_report*)	да
37	Выполнение метода класса Hybrid_CNN_GRU (*test_draw_plot*)	да
38	Выполнение метода класса Hybrid_CNN_GRU (*test_save_load_model*)	да
39	Выполнение метода класса AutoEncoder (*test_define_model*)	да
40	Выполнение метода класса AutoEncoder (*test_test*)	да
41	Выполнение метода класса AutoEncoder (*test_predict*)	да
42	Выполнение метода класса AutoEncoder (*test_draw_plot*)	да

Модульный тест № 1. Создание модели оценивания (*test_sc_is_not_none*)

Исходный код:

```
self.assertFalse(SAIClassifier('neural_network', 10, 1) is None, 'The object of
class SAIClassifier could not be created')
```

Модульный тест № 2. Инициализация модели оценивания (*test_sc_type_is_correct*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
for c in classifier_types:
        classifier = SAIClassifier(c, 10, 1, plot=False)
        self.assertTrue(classifier.cls_type in classifier_types, 'The classifier
type is not correct')
```

Модульный тест № 3. Корректность модели оценивания (*test_sc_fit*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
for c in classifier_types:
        classifier = SAIClassifier(c, in_size, out_size, plot=False)
        try:
            classifier.fit(self.xor_ds['features'], self.xor_ds['labels'])
        except:
            self.assertTrue(False, 'Error while calling fit')
```

Модульный тест № 4. Оценивание тестового набора (*test_sc_predict*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
for c in classifier_types:
        classifier = SAIClassifier(c, in_size, out_size, plot=False)
        classifier.fit(self.xor_ds['features'], self.xor_ds['labels'])
        try:
            classifier.predict(self.xor_ds['features'])
        except:
            self.assertTrue(False, 'Error while calling predict')
```

Модульный тест № 5. Сохранение модели оценивания (*test_sc_save*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
for c in classifier_types:
        classifier = SAIClassifier(c, in_size, out_size, plot=False)
        classifier.fit(self.xor_ds['features'], self.xor_ds['labels'])
        f = './' + c + '.bin'
if os.path.isfile(f):
        os.remove(f)
        classifier.save(f)
        self.assertTrue(os.path.isfile(f) or os.path.isfile(f + '.index'),
'The classifier could not be saved into the file')
```

Модульный тест № 6. Загрузка из файла (*test_sc_load*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
for c in classifier_types:
        classifier = SAIClassifier(c, in_size, out_size, plot=False)
        f = './' + c + '.bin'
        classifier.save(f)
        try:
            classifier.load(f)
        except:
            self.assertTrue(False, 'Error while calling load')
```

Модульный тест № 7. Проверка разделителя (*test_fd_is_not_none*)

Исходный код:

```
f = '../hai/hai-20.07/test1.csv.gz'
if os.path.isfile(f):
        self.assertFalse(FormatDetector(f) is None, 'The object of class
FormatDetector could not be created')
```

Модульный тест № 8. Существование файла для загрузки (*test_fd_file_exists*)

Исходный код:

```
tmp_file = tempfile.NamedTemporaryFile()
try:
        FormatDetector(tmp_file.name)
except:
        self.assertTrue(False, 'Error: file "{}" must exist'.format(tmp_file))
```

Модульный тест № 9. Корректность разделителя (*test_fd_delimiter_is_correct*)

Исходный код:

```
f = '../hai/hai-20.07/test1.csv.gz'
if os.path.isfile(f):
        fd = FormatDetector(f)
        self.assertTrue(fd.d in [';', ','], 'The delimiter is not correct')
```

Модульный тест № 10. Инициализация объекта загрузки данных (*test_dl_is_not_none*)

Исходный код:

```
f = '../hai/hai-20.07/test1.csv.gz'
class DataLoaderExample(DataLoader):
        def load(self, file):
            pass
if os.path.isfile(f):
        self.assertFalse(DataLoaderExample(f, 0, 0) is None, 'The object of class
DataLoaderExample could not be created')
```

Модульный тест № 11. Инициализация показателей эффективности (*test_ce_is_not_none*)

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
classifiers = [SAIClassifier(c, in_size, out_size, plot=False) for c in
classifier_types]
for classifier in classifiers:
```

```
        self.assertFalse(ClsEstimator(self.xor_ds['features'],
self.xor_ds['labels'], self.xor_ds['labels'], [classifier]) is None,
                         'The object of class ClsEstimator could not be created')
```

Модульный тест № 12. Вычисление показателей эффективности *(test_ce_estimate).*

Исходный код:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression',
'neural_network']
in_size = np.shape(self.xor_ds['features'])[1]
out_size = np.shape(self.xor_ds['labels'])[1]
classifiers = [SAIClassifier(c, in_size, out_size, plot=False) for c in
classifier_types]
for classifier in classifiers:
        classifier.fit(self.xor_ds['features'], self.xor_ds['labels'])
    try:
        ClsEstimator(self.xor_ds['features'], self.xor_ds['labels'],
self.xor_ds['labels'], classifiers).estimate()
    except:
        self.assertTrue(False, 'Error while calling estimate')
```

Модульный тест № 13. Инициализация элемента класса GAN (*test_build_generator*)

Исходный код:

```
self.assertIsNotNone(self.gan.build_generator())
```

Модульный тест № 14. Инициализация элемента класса *GAN* (*test_build_discriminator*)

Исходный код:

```
self.assertIsNotNone(self.gan.build_discriminator())
```

Модульный тест № 15. Инициализация элемента класса *GAN* (*test_build_gan*)

Исходный код:

```
self.assertIsNotNone(self.gan.build_gan())
```

Модульный тест № 16. Выполнение метода класса *GAN* (*test_train*)

Исходный код:

```
history = self.gan.train(self.X_train, self.y_train, self.X_val, self.y_val,
epochs=5, batch_size=32)
self.assertIsNotNone(history)
```

Модульный тест № 17. Выполнение метода класса *GAN* (*test_print_classification_report*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
model = foras.GAN(input_shape=self.input_shape)
model.train(self.X_train, self.y_train, self.X_val, self.y_val, epochs=5,
batch_size=32)
model.print_classification_report(X_test, y_test)
self.assertIsNone(model.print_classification_report(X_test, y_test,
target_names=['class1', 'class2']))
mock_print.assert_called()
```

Модульный тест № 18. Выполнение метода класса *GAN* (*test_draw_plot*)

Исходный код:

```
model = foras.GAN(input_shape=self.input_shape)
model.train(self.X_train, self.y_train, self.X_val, self.y_val, epochs=5,
batch_size=32)
self.assertIsNone(model.draw_plot("accuracy"))
mock_show.assert_called()
```

Модульный тест № 19. Выполнение метода класса *DeepCNN* (*test_model_build*)

Исходный код:

```
model = foras.DeepCNN(input_shape=self.input_shape, blocks=1, units=64,
classes=self.classes)
self.assertIsNotNone(model.model)
```

Модульный тест № 20. Выполнение метода класса *DeepCNN* (*test_model_fit*)

Исходный код:

```
history = self.model.fit(self.X_train, self.y_train, validation_data=(self.X_val,
self.y_val), epochs=10, batch_size=128, verbose=0)
self.assertIsInstance(history.history, dict)
self.assertGreater(len(history.history), 0)
```

Модульный тест № 21. Выполнение метода класса *DeepCNN* (*test_model_predict*)

Исходный код:

```
model = foras.DeepCNN(input_shape=self.input_shape, blocks=2, units=64,
classes=self.classes)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val, self.y_val),
epochs=10, batch_size=128, verbose=0)
y_pred = model.predict(self.X_val)
self.assertEqual(y_pred.shape[1], self.classes)
```

Модульный тест № 22. Выполнение метода класса *DeepCNN* (*test_classification_report*)

Исходный код:

```
model = foras.DeepCNN(input_shape=self.input_shape, blocks=2, units=128,
classes=self.classes)
```

```
model.fit(self.X_train, self.y_train, validation_data=(self.X_val, self.y_val),
epochs=10, batch_size=128, verbose=0)
model.print_classification_report(self.X_val, self.y_val)
```

Модульный тест № 23. Выполнение метода класса *DeepCNN* (*test_plot*)
Исходный код:

```
model = foras.DeepCNN(input_shape=self.input_shape, blocks=3, units=64,
classes=self.classes)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val, self.y_val),
epochs=10, batch_size=128, verbose=0)
model.draw_plot(plot_type="accuracy")
 model.fit(self.X_train, self.y_train, validation_data=(self.X_val, self.y_val),
epochs=10, batch_size=128, verbose=0)
model.print_classification_report(self.X_val, self.y_val)
```

Модульный тест № 24. Выполнение метода класса *DeepCNN* (*test_save_load_model*)
Исходный код:

```
self.assertIsNone(self.model.save_model(self.temp_model_path))
```

Модульный тест № 25. Выполнение метода класса *DeepCNN* (*test_load_model*)
Исходный код:

```
loaded_model = foras.DeepCNN.load_model(self.temp_model_path)
self.assertIsInstance(loaded_model, foras.DeepCNN)
```

Модульный тест № 26. Выполнение метода класса *hybrid_variation* (*test_build_model*)
Исходный код:

```
self.assertIsNotNone(
self.model.build_model(input_shape=(100, 1), units=64, classes=2, block="Xception",
loop_number=1))
self.assertIsNotNone(self.model.build_model(input_shape=(100, 1), units=128,
classes=4, block="Xception", loop_number=3))
self.assertIsNotNone(self.model.build_model(input_shape=(50, 1), units=64,
classes=2, block="residual", loop_number=2))
self.assertIsNotNone(self.model.build_model(input_shape=(50, 1), units=128,
classes=4, block="residual", loop_number=4))
```

Модульный тест № 27. Выполнение метода класса *hybrid_variation* (*test_fit*)
Исходный код:

```
history = self.model.fit(self.X_train, self.y_train, validation_data=(self.X_val,
self.y_val), epochs=5, batch_size=32, verbose=0)
self.assertIsNotNone(history)
assert len(history.history['loss']) == 5
```

Модульный тест № 28. Выполнение метода класса *hybrid_variation* (*test_test*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
loss = self.model.test(X_test, y_test)
self.assertIsNotNone(loss)
print(loss)
assert len(loss) == 5  # should have 5 items
```

Модульный тест № 29. Выполнение метода класса *hybrid_variation* (*test_predict*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
model = foras.hybrid_variation(input_shape=self.input_shape, units=64, classes=2)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val, self.y_val),
epochs=5, batch_size=32, verbose=0)
model.print_classification_report(X_test, y_test)
```

Модульный тест № 30. Выполнение метода класса hybrid_variation (*test_print_classification_report*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
model = foras.hybrid_variation(input_shape=self.input_shape, units=64, classes=2)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val,self.y_val),
epochs=5, batch_size=32, verbose=0)
model.print_classification_report(X_test, y_test)
```

Модульный тест № 31. Выполнение метода класса *hybrid_variation* (*test_draw_plot*)

Исходный код:

```
model = foras.hybrid_variation(input_shape=self.input_shape, units=64, classes=2)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val,self.y_val),
epochs=5, batch_size=32, verbose=0)
self.assertIsNone(model.draw_plot("accuracy"))
self.assertIsNone(model.draw_plot("loss"))
self.assertIsNone(model.draw_plot("auc"))
self.assertIsNone(model.draw_plot("invalid"))
```

Модульный тест № 32. Выполнение метода класса *hybrid_variation* (*test_save_and_load_model*)

Исходный код:

```
self.assertIsNone(self.model.save_model(self.temp_model_path))
loaded_model = foras.hybrid_variation.load_model(self.temp_model_path)
self.assertIsInstance(loaded_model, foras.hybrid_variation)
```

Модульный тест № 33. Выполнение метода класса *Hybrid_CNN_GRU* (*test_build_model*)

Исходный код:

```
assert self.model.build_model((100, 1), 64, 2).input_shape == (None, 100, 1)
assert self.model.build_model((200, 1), 128, 3).input_shape == (None, 200, 1)
assert self.model.build_model((300, 1), 256, 4).input_shape == (None, 300, 1)
```

Модульный тест № 34. Выполнение метода класса *Hybrid_CNN_GRU* (*test_fit*)

Исходный код:

```
history = self.model.fit(self.X_train, self.y_train,
validation_data=(self.X_val,self.y_val), epochs=5, batch_size=32, verbose=0)
 assert len(history.history['loss']) == 5
```

Модульный тест № 35. Выполнение метода класса *Hybrid_CNN_GRU* (*test_test*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
loss = self.model.test(X_test, y_test)
self.assertIsNotNone(loss)
assert len(loss) == 5 # should have 5 items
```

Модульный тест № 36. Выполнение метода класса *Hybrid_CNN_GRU* (*test_print_classification_report*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=15, random_state=42)
model = foras.Hybrid_CNN_GRU(input_shape=self.input_shape, units=64, classes=2)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val,self.y_val),
epochs=5, batch_size=32, verbose=0)
model.print_classification_report(X_test, y_test)
```

Модульный тест № 37. Выполнение метода класса *Hybrid_CNN_GRU* (*test_draw_plot*)

Исходный код:

```
model = foras.Hybrid_CNN_GRU(input_shape=self.input_shape, units=64, classes=2)
model.fit(self.X_train, self.y_train, validation_data=(self.X_val,self.y_val),
epochs=5, batch_size=32, verbose=0)
model.draw_plot("accuracy")
model.draw_plot("loss")
model.draw_plot("auc")
model.draw_plot("invalid")  # Should print "Invalid plot_type. Choose 'accuracy',
'loss', or 'auc'."
```

Модульный тест № 38. Выполнение метода класса *Hybrid_CNN_GRU* (*test_save_load_model*)

Исходный код:

```
self.model.save_model("model_save_test.h5")
```

Модульный тест №39. Выполнение метода класса *AutoEncoder* (*test_define_model*)

Исходный код:

```
self.assertIsNotNone(self.autoencoder.model)
self.assertIsNotNone(self.autoencoder_clf.model)
```

Модульный тест № 40. Выполнение метода класса *AutoEncoder* (*test_fit*)

Исходный код:

```
epochs = 5
# the recnstruction mode of AE
self.autoencoder.fit(self.X_train_, self.X_train_, validation_data = (self.X_val_,
self.X_val_), epochs=epochs)
# the classification mode of AE
self.autoencoder_clf.fit(self.X_train, self.y_train, validation_data = (self.X_val,
self.y_val), epochs=epochs)
```

Модульный тест № 41. Выполнение метода класса *AutoEncoder* (*test_test*)

Исходный код:

```
X_test, y_test = make_classification(n_samples=100, n_features=20, random_state=42)
re = self.autoencoder
re.fit(self.X_train_, self.X_train_, validation_data = (self.X_val_, self.X_val_),
epochs=2)
print(re.history.history)
loss = re.test(X_test,X_test)
self.assertIsNotNone(loss)
print("reconstruction loss exits.")
clf = self.autoencoder_clf
clf.fit(self.X_train, self.y_train, validation_data = (self.X_val, self.y_val),
epochs=2)
loss = clf.test(X_test,y_test)
self.assertIsNotNone(loss)
print("reconstruction loss exits.")
```

Модульный тест № 42. Выполнение метода класса *AutoEncoder* (*test_predict*)

Исходный код:

```
# Testing the predict() method with dummy data
X_test, y_test = make_classification(n_samples=100, n_features=20, random_state=42)
re = self.autoencoder
re.fit(self.X_train_, self.X_train_, validation_data = (self.X_val_, self.X_val_),
epochs=2)
decoded_X = re.predict(X_test)
self.assertIsNotNone(decoded_X)
```

```
self.assertEqual(decoded_X.shape, X_test.shape)
clf = self.autoencoder_clf
clf.fit(self.X_train, self.y_train, validation_data = (self.X_val, self.y_val),
epochs=2)
pred_y = clf.predict(X_test)
self.assertIsNotNone(pred_y)
```

Модульный тест № 43. Выполнение метода класса *AutoEncoder* (*test_draw_plot*)
Исходный код:

```
re = self.autoencoder
re.fit(self.X_train_, self.X_train_, validation_data = (self.X_val_, self.X_val_),
epochs=2)
re.draw_mse_plot(plot_type="loss")
clf = self.autoencoder_clf
clf.fit(self.X_train, self.y_train, validation_data = (self.X_val, self.y_val),
epochs=2)
clf.draw_clf_plot(plot_type="auc")
```

6.1.4 Модуль АПССОП

Список тестов для модуля АПССОП сведен в единую таблицу 6.1.4.

Таблица 6.1.4 – Тесты модуля АПССОП

Номер теста	Назначение теста	Пройден ли тест
1	Инициализация объекта класса *ForecasterParameters* (*test_is_not_none*)	да
2	Установка атрибутов объекта класса *ForecasterParameters* (*test_set_incorrect_params*)	да
3	Установка атрибутов объекта класса *ForecasterParameters* (*test_set_incorrect_argument*)	да
4	Инициализация объекта класса *DeepForecasterParameters* (*test_is_not_none*)	да
5	Установка атрибутов объекта класса *DeepForecasterParameters* (*test_set_incorrect_params*)	да
6	Установка атрибутов объекта класса *DeepForecasterParameters* (*test_set_incorrect_argument*)	да
7	Инициализация объекта класса *TSGenerator* (*test_is_not_none*)	да
8	Инициализация объекта класса *TSGenerator* (test_set_incorrect_data)	да
9	Инициализация объекта класса *TSGenerator* (*test_set_insufficient_data*)	да
10	Выполнение метода класса *TSGenerator* (*test_get_data_and_targets*)	да
11	Инициализация объекта класса *NaiveForecaster* (*test_is_not_none*)	да

12	Выполнение метода класса *NaiveForecaster* (*test_forecasting*)	да
13	Выполнение метода класса *NaiveForecaster* (*test_forecasting_incorrect_data*)	да
14	Выполнение метода класса *NaiveForecaster* (*test_forecasting_incorrect_horizon*)	да
15	Инициализация объекта класса *DeepForecaster* (*test_is_not_none*)	да
16	Выполнение метода класса *DeepForecaster* (*test_forecasting_model*)	да
17	Инициализация объекта класса *DeepForecasterTuner* (*test_is_not_none*)	да
18	Выполнение метода класса *DeepForecasterTuner* (*test_set_tuned_hps*)	да
19	Выполнение метода класса *DeepForecasterTuner* (test_set_tuned_hps)	да
20	Инициализация объекта класса *ForecastEstimator* (*test_is_not_none*)	да
21	Выполнение метода класса *ForecastEstimator* (*test_is_not_none*)	да

Модульный тест №11. Инициализация объекта класса *NaiveForecaster* (*test_is_not_none*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The object of class NaiveForecaster can\'t not be created'
params = ForecasterParameters()
self.assertIsNotNone(NaiveForecaster(params), message)
```

Модульный тест №12. Выполнение метода класса *NaiveForecaster* (*test_forecasting*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Forecasting failed'
params = ForecasterParameters()
nf = NaiveForecaster(params)
data = np.random.randint(1, 10, size=(10, 10, 1))
with self.subTest(0):
            self.assertIsNotNone(nf.forecasting(data), message)
with self.subTest(1):
            self.assertIsNotNone(nf.forecasting(data, 10), message)
```

Модульный тест №13. Выполнение метода класса *NaiveForecaster* (*test_forecasting_incorrect_data*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Forecasting failed'
params = ForecasterParameters()
```

```
nf = NaiveForecaster(params)
data = np.random.randint(1, 10, size=(100, 1, 1, 1))
with self.assertRaises(AssertionError, msg=message):
    nf.forecasting(data)
```

Модульный тест №14. Выполнение метода класса *NaiveForecaster* (*test_forecasting_incorrect_horizon*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'Forecasting failed'
params = ForecasterParameters()
nf = NaiveForecaster(params)
data = np.random.randint(1, 10, size=100)
with self.assertRaises(AssertionError, msg=message):
       nf.forecasting(data, forecasting_data_length=-10)
```

Модульный тест №15. Инициализация объекта класса *DeepForecaster* (*test_is_not_none*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The object of class DeepForecaster can\'t not be created'
params = DeepForecasterParameters()
self.assertIsNotNone(DeepForecaster(params), message)
```

Модульный тест №16. Выполнение метода класса *DeepForecaster* (*test_forecasting_model*)

Исходный код:

```
print(inspect.stack()[0][3])
params = DeepForecasterParameters()
ts = TSGenerator(np.random.randint(1, 10, 10000), params)
x = ts.get_data()
y = ts.get_data()
with self.subTest(0):
   aif = DeepForecaster(params)
   aif.build_model()
   self.assertIsNotNone(aif.model, 'The model of class DeepForecaster can\'t not be
created')
with self.subTest(1):
    aif.train(x, y, n_epochs=3, batch_size=256)
    self.assertIsNotNone(aif.history, 'Model training failed')
 with self.subTest(2):
    new_data = np.random.randint(1, 10, size=(10, 10, 1))
    self.assertIsNotNone(aif.forecasting(new_data), 'Forecasting failed')
with self.subTest(3):
    self.assertIsNotNone(aif.forecasting(new_data, 10), 'Forecasting failed')
```

Модульный тест №17. Инициализация объекта класса *DeepForecasterTuner* (*test_is_not_none*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The object of class DeepForecaster can\'t not be created'
params = DeepForecasterParameters()
self.assertIsNotNone(DeepForecasterTuner(params), message)
```

Модульный тест №18. Выполнение метода класса *DeepForecasterTuner* (*test_set_tuned_hps*)

Исходный код:

```
params = DeepForecasterParameters()
tuner = DeepForecasterTuner(params)
tuner.set_tuned_hps(units=[[512, 400, 316], [256, 160, 80], [64, 32, 16]],
                    n_rec_layers=[1, 2, 3],
                    dropout=[0.0, 0.01, 0.5],
                     hidden_activation=['tanh', 'relu'],
                     output_activation=['linear', 'sigmoid'])
self.assertIsNotNone(tuner.model_params)
```

Модульный тест №19. Выполнение метода класса *DeepForecasterTuner* (*test_set_tuned_hps*)

Исходный код:

```
params = DeepForecasterParameters()
ts = TSGenerator(np.random.randint(1, 10, 1000), params)
x = ts.get_data()
y = ts.get_data()
tuner = DeepForecasterTuner(params)
tuner.set_tuned_hps(n_rec_layers=[1, 2], units=[[64, 32], [24, 16]])
tuner.find_best_models(x, y, epochs=3, n_models=1, max_trials=3)
```

Модульный тест №20. Инициализация объекта класса *ForecastEstimator* (*test_is_not_none*)

Исходный код:

```
print(inspect.stack()[0][3])
message = 'The object of class ForecastEstimator can\'t not be created'
self.assertIsNotNone(ForecastEstimator(), message)
```

Модульный тест №21. Выполнение метода класса *ForecastEstimator* (*test_is_not_none*)

Исходный код:

```
test_estimate(self):
message = 'Forecasting quality evaluation failed'
est = ForecastEstimator()
est.set_true_values(np.array([0, 0, 1, 0, 2, 1, 1, 0, 0, 1]))
est.set_pred_values(np.array([0, 1, 1, 0, 0, 0, 1, 0, 0, 1]))
est.estimate()
self.assertFalse(est.quality.empty, message)
```

6.2 Интеграционные тесты

Возможности интеграции между модулями компонента представлены на рис. 5.1.1 Как показано на рисунке, каждый ПОИНД, ИЗСНД, АОССОП и АПССОП могут работать с исходными данными. При этом данные от ПОИНД могут быть в начале приняты на выход ИЗСНД, или же сразу переданы АОССОП /АПССОП. При этом модули ПОИНД и ИЗСНД решают задачи повышения качества анализируемых данных, а также снижения их объема и выполняют вспомогательную функцию. А модули АОССОП и АПССОП решают ключевую задачу сильного ИИ – оценивание или прогнозирование состояния СлОП, соответственно.

Отметим, что потребность в интеграции отдельных алгоритмов библиотеки между собой определяется в первую очередь данными, которые являются основой для осуществления оценивания или прогнозирования. К примеру, решение о необходимости выполнение алгоритма прогнозирования предварить вызовом одного из имеющихся алгоритмов предобработки, как предполагается, должно приниматься пользователем библиотеки, исходя из особенности имеющихся у него данных. В частности, пользователь может действовать итеративно – попробовать запустить прогнозирование сначала на необработанных данных. И если пользователь получает в недостаточной степени удовлетворяющий результат прогнозирования, он может в зависимости от специфики своих данных выбрать и применить ту или иную предобработку с последующей проверкой того, что это в итоге улучшит качество прогнозирования. В качестве примера такой интеграции на имеющихся в процессе экспериментов данных, мы показали успешный пример интеграции алгоритмов ИЗСНД и АПССОП.

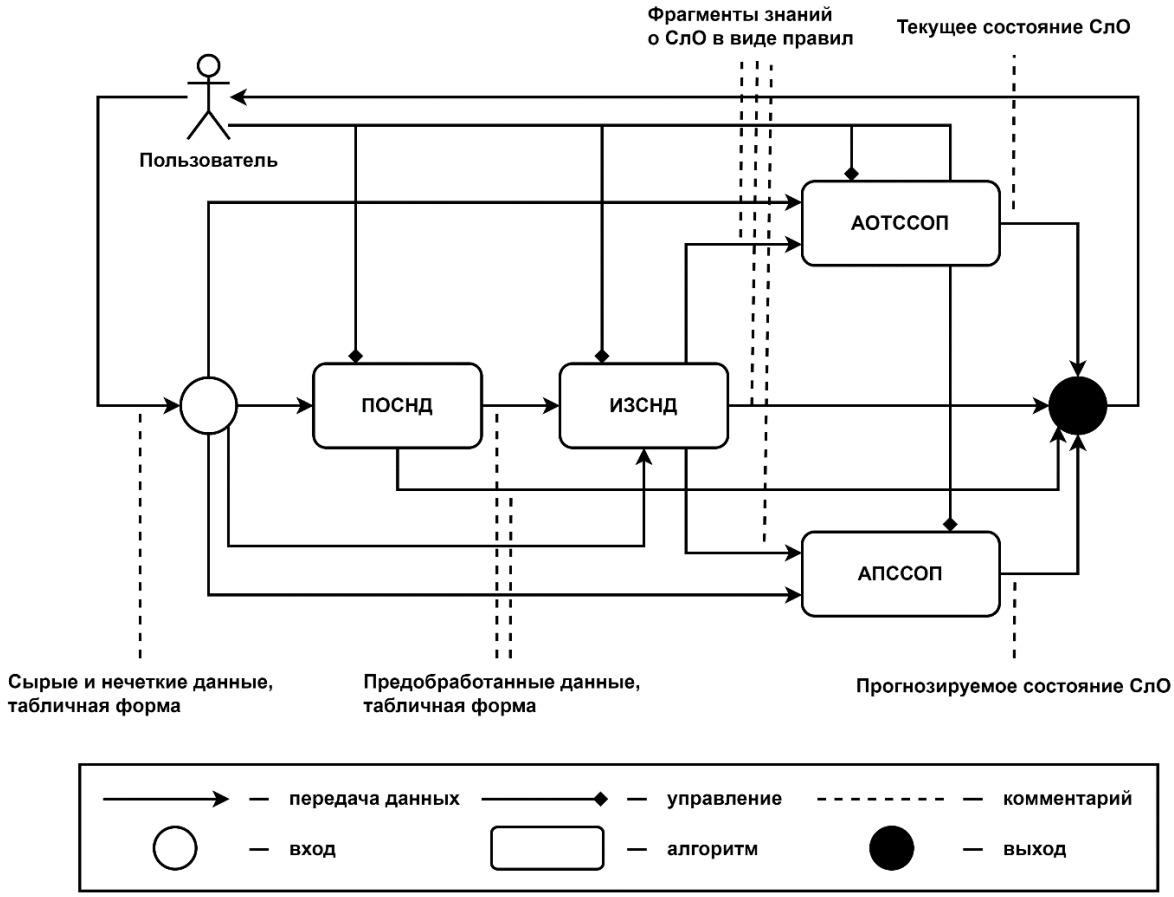

Рисунок 6.1.1 – Возможности интеграции между модулями компонента

6.2.1 Интеграция алгоритмов ИЗСНД и АПССОП

Описание теста:

В данном тесте представлена интеграция алгоритмов ИЗСНД и АПССОП. Перед обучением модели и для прогнозирования используются данные, предобработанные алгоритмом ИЗСНД на основе правил.

Исходный код:

```
from foressment_ai import TSGenerator, DeepForecaster, ForecastEstimator
from foressment_ai import RulesExtractor
from foressment_ai import AopssopData as PDATA
# loading data
data = PDATA.features_matrix
algo = RulesExtractor(probability_threshold)
algo.fit(data, class_column='Attack', positive_class_label=1)
transformed_data = algo.transform(data)
train, test = train_test_split(transformed_data, train_size=0.9)
```

```
model_params = DeepForecaterParameters(n_features= transformed_data.shape[1],
                                       look_back_lengh=PDATA.time_window_length)
forecasting_model = DeepForecaster(model_params)
train_generator = TSGenerator(train)
test_generator = TSGenerator(test)
forecasting_model.train(train_generator.get_data())


current_batch = test_generator.get_data()
pred = forecasting_model.forecasting(current_batch, forecasting_data_length=1)
```

6.2.2 Интеграция алгоритмов АОТСОП и АПССОП

Описание теста:

В данном тесте представлена интеграция алгоритмов АОТСОП и АПССОП. Для обучения моделей оценивания и прогнозирования используются одинаковые данные.

Исходный код:

```
from foressment_ai import TSGenerator, DeepForecaster
from foressment_ai import DeepCNN

from foressment_ai import RulesExtractor
from foressment_ai import ForessmentData

# loading data
fd = ForessmentData().
data = fd.features_matrix
labels = fd.labels_matrix
n_features = len(fd.features_matrix)
n_labels = len(fd.labels_names)
batch_size = 32

X_train, X_test, y_train, y_test = train_test_split(data, labels, train_size=0.9,
                                                    shuffle=False)
assessor_model = DeepCNN(input_shape=(n_features,1), blocks=1, units=96,
                         classes= n_labels)
history= assessor_model.fit(X_train, y_train, validation_data=(X_test, y_test),
                            epochs=50, batch_size=batch_size, verbose=0)
assessor_result = assessor_model.predict(X_test)

forecasting_model_params = DeepForecaterParameters(n_features= n_features,
                                       look_back_lengh=PDATA.time_window_length)
forecasting_model = DeepForecaster(model_params)

train_generator = TSGenerator(X_train)
test_generator = TSGenerator(X_test)
forecasting_model.train(train_generator.get_data())

current_batch = test_generator.get_data()
forecasting_result = forecasting_model.forecasting(current_batch, 1)
```

6.3 Контрольные примеры

Контрольные примеры для демонстрации работы алгоритмов компонента сильного ИИ приведены далее.

6.3.1 Алгоритм ПОИНД

Контрольный пример №1.

Описание: Обработка набора данных Titanic.

Входные данные

```
data = Data()
titanic_path = '../datasets/titanic.csv'
titanic = pd.read_csv(titanic_path)
data.features_names = ["PassengerId", "Pclass", "Age", "SibSp", "Parch"]
data.labels_names = ["Survived", "Fare"]
data.features_matrix = np.array(titanic[data.features_names])
data.labels_matrix = np.array(titanic[data.labels_names])
data.features_types = ["cat", "cat", "num", None, None]
data.labels_types = ["cat", None]
```

Настройка алгоритма

```
data.n_jobs = 2
data.feature_names_substrings = {
"num": ["age", "id"],
 "cat": ["surv", "tick", "cabin"]
}
data.feature_max_cat = 10
data.types_priority = {
       "manual": 0.5,
"substring": 1,
              "unique": 1,
               "float": 0.3
}
data.fill_method = "mean_mode"
data.n_clusters = 10
data.cluster_max_iter = 5
data.thresholds_correlation_with_label = {
       "num_num": [0.2] * len(data.labels_names),
              "cat_cat": [0.1] * len(data.labels_names),
        "num_cat": [0.2] * len(data.labels_names)
}
data.thresholds_min_number_of_predicted_labels = [1] * len(data.features_names)
data.thresholds_multicolinear = {
       "num_num": 0.9,
        "cat_cat": 0.7,
              "num_cat": 0.8}
```

Обработка

```
CheckDataTypes.CheckDataTypes.correct_types(data)
ClusterFilling.ClusterFilling.fill(data)
Informativity.Informativity.calculate_informativity(data)
Multicolinear.MultiCollinear.remove_uninformative_features(data)
```

Выходные данные, которые будут получены при выполнении данного контрольного примера представлены ниже.

```
START ANALYSIS OF DATA TYPES (N of substr[num,cat]=[2,3], max_cat=10,
       priority[manual,substring,unique,float]=[0.5,1,1,0.3])
       PassengerId with FEATURE_type=cat is incorrect. Changing to num
(num=1.0,cat=0.8).
SibSp with FEATURE_type=None is incorrect. Changing to cat (num=0.0,cat=1.3).
Parch with FEATURE_type=None is incorrect. Changing to cat (num=0.0,cat=1.3).
Fare with LABEL_type=None is incorrect. Changing to num(num=1.0,cat=0.3).
ANALYSIS OF DATA TYPES IS COMPLETE
START CLUSTER FILLING (fill_method=mean_mode, n_clusters=10, max_iter=5)
 iteration 0, fill 177 NaNs
iteration 1, fill 177 NaNs
iteration 2, fill 177 NaNs
 iteration 3, fill 177 NaNs
=> conversged on iteration 3
DONE CLUSTER FILLING
START INFORMATIVITY ANALYSIS
NO UNINFORMATIVE FEATURES
START MULTICOLINEARITY ANALYSIS
       delete feature=SibSp as it correlates with feature=PassengerId
       delete feature=Age as it correlates with feature=Pclass
REMOVE MILTICOLINEAR FEATURES: [Age, SibSp]
```

6.3.2 Алгоритм ИЗСНД

Контрольный пример №1.

Описание: применение алгоритма ИЗСНД совместно с алгоритмом Random Forest на наборе данных IEEE Smart Crane.

Входные данные:

```
DATA_PATH = "../datasets/IEEE_smart_crane.csv"
ieee_data = pd.read_csv(DATA_PATH)
```

Работа Random Forest с данными без предобработки ИЗСНД:

```
X_train, X_test, y_train, y_test = train_test_split
(ieee_data.drop(columns=['Alarm']), ieee_data.Alarm, test_size = 0.3, shuffle =
False)
clf = RandomForestClassifier(random_state=5)
```

```
clf.fit(X_train, y_train)
y_test_pred = clf.predict(X_test)
y_test_proba = clf.predict_proba(X_test)
```

Выходные данные, которые будут получены при выполнении данного контрольного примера представлены ниже.

```
Random forest without IZSND preproccesing
              precision    recall  f1-score   support
           0   0.953165  0.995644  0.973942      8953
           1   0.000000  0.000000  0.000000       438

    accuracy                       0.949207      9391
   macro avg   0.476583  0.497822  0.486971      9391
weighted avg   0.908709  0.949207  0.928517      9391

ROC-AUC: 0.439477698605656
Accuracy: 0.9492066872537536
```

Работа Random Forest с данными, которые были предобработаны ИЗСНД:

```
algo = RulesExtractor(0.1)
algo.fit(ieee_data, class_column = "Alarm", positive_class_label = 1)
transformed_data = algo.transform(ieee_data)
X_train, X_test, y_train, y_test = train_test_split
(transformed_data.drop(columns=['Alarm']),     transformed_data.Alarm, test_size =
0.3, shuffle = False)
clf.fit(X_train, y_train)
y_test_pred = clf.predict(X_test)
y_test_proba = clf.predict_proba(X_test)
```

Выходные данные, которые будут получены при выполнении данного контрольного примера представлены ниже.

```
Random forest with IZSND preproccesing

              precision    recall  f1-score   support
           0   0.953360  1.000000  0.976123      8953
           1   0.000000  0.000000  0.000000       438

    accuracy                       0.953360      9391
   macro avg   0.476680  0.500000  0.488061      9391
weighted avg   0.908895  0.953360  0.930596      9391

ROC-AUC: 0.5
Accuracy: 0.9533595996166543
```

```
Random forest without IZSND preproccesing
memory usage: 3.3 MB
ROC-AUC: 0.439477698605656
Accuracy: 0.9492066872537536
```

```
Random forest with IZSND preproccesing
memory usage: 2.9+ MB
ROC-AUC: 0.5
Accuracy: 0.9533595996166543
```

6.3.3 Алгоритм АОССОП

Примеры далее запускаются последовательно, поскольку в примере № 2 используются программные объекты, созданные в примере № 1.

Контрольный пример №1.

Описание: Загрузка набора данных. Для загрузки данных используются вспомогательные классы FormatDetector и DataLoader, которые осуществляют определение типа разделителя полей и загрузку набора данных.

```
import numpy as np
import pandas as pd
from collections import OrderedDict
from sklearn.utils import shuffle
from aossop import SAIClassifier, FormatDetector, DataLoader, ClsEstimator
class DataLoaderHai(DataLoader):
        def __init__(self, file, n, d):
                DataLoader.__init__(self, file, n, d)
        def load(self, file):
                n, d = self.n, self.d
                if n in [64, 84]:
                  self.features = np.genfromtxt(file, delimiter=d, dtype=float, skip_header=1,
                                                usecols=range(1,n-4))
                  attacks = np.genfromtxt(file, delimiter=d, dtype=float, skip_header=1,
                                          usecols=range(n-4,n))
                  self.labels = np.array([[0] if sum(a) == 0 else [1] for a in attacks])
                  self.num_labels = self.labels
                elif n == 88:
                   self.features = np.genfromtxt(file, delimiter=d, dtype=float, skip_header=1,
                                                 usecols=range(1,n-1))
                   self.labels = np.genfromtxt(file, delimiter=d, dtype=float, skip_header=1,
                                               usecols=range(n-1,n))
                if len(np.shape(self.labels)) == 1:
                   self.labels = np.array([[e] for e in self.labels])
                   self.num_labels = self.labels
class DataLoaderEdgeIIoTSet(DataLoader):
        def __init__(self, file, n, d):
                DataLoader.__init__(self, file, n, d)
         def load(self, file):
            df = pd.read_csv(file, low_memory=False, sep=self.d)
            drop_columns = ["frame.time", "ip.src_host", "ip.dst_host", "arp.src.proto_ipv4",
                            "arp.dst.proto_ipv4", "http.request.method", "http.file_data",
                            "http.referer", "http.request.full_uri", "http.request.version",
                            "icmp.transmit_timestamp", "http.request.uri.query", "tcp.options",
                            "tcp.payload", "tcp.srcport", "tcp.dstport", "udp.port",
                            "dns.qry.name", "dns.qry.name.len", "dns.qry.qu",
                            "mqtt.conack.flags", "mqtt.msg", "mqtt.protoname", "mqtt.topic",
                            "Attack_label"]
            df.drop(drop_columns, axis=1, inplace=True)
            df.dropna(axis=0, how='any', inplace=True)
```

```
           df.drop_duplicates(subset=None, keep="first", inplace=True)
           df = shuffle(df)
           str_labels = df.iloc[:,-1].tolist()
           self.features = np.array(df.iloc[:,:-1].values.tolist())
           unique_labels = list(OrderedDict.fromkeys(str_labels))
           labels = list(map(lambda x: unique_labels.index(x), str_labels))
           self.labels = np.array([np.zeros(len(unique_labels))] * len(labels))
for i in range(len(labels)): # one-hot encoding
   self.labels[i][labels[i]] = 1
   self.num_labels = labels

class DataLoaderDataPort(DataLoader):
    def __init__(self, file, n, d):
        DataLoader.__init__(self, file, n, d)
    def load(self, file):
       self.features = np.genfromtxt(file, delimiter=self.d, dtype=float, skip_header=1,
                                     usecols=range(1,self.n-1))
       labels = np.genfromtxt(file, delimiter=self.d, dtype=float, skip_header=1,
                              usecols=range(self.n-1,self.n))
       self.labels = np.array([np.zeros(len(set(labels)))] * len(labels))
       for i in range(len(labels)): # one-hot encoding
       self.labels[i][int(labels[i]) - 1] = 1
       self.num_labels = labels

fd = FormatDetector(file)
dl = None
if dataset == 'hai':
    dl = DataLoaderHai(file, fd.n, fd.d)
elif dataset == 'edge-iiotset':
    dl = DataLoaderEdgeIIoTSet(file, fd.n, fd.d)
elif dataset == 'dataport':
dl = DataLoaderDataPort(file, fd.n, fd.d)
else:
    assert(False)
```

Входные данные: *file* = «hai.csv»– название набора данных (текстовая строка)

Выходные данные: *fd* – объект *FormatDetector*, содержащий информацию о формате входных данных; *dl* – объект *DataLoaderDataPort*, содержащий загруженный набор данных в виде массива значений.

Контрольный пример №2.

Описание: Обучение и тестирование модели оценивания. Предварительно должны быть созданы объекты *fd* и *dl* классов *FormatDetector* и *DataLoader*.

```
import numpy as np
import pandas as pd
from collections import OrderedDict
from sklearn.utils import shuffle
from aossop import SAIClassifier, FormatDetector, DataLoader, ClsEstimator
classifiers = [SAIClassifier(cls_type=c, in_size=np.shape(dl.features)[1],
out_size=np.shape(dl.labels)[1]) for c in ['neural_network']]
ce = ClsEstimator(dl.features, dl.labels, dl.num_labels, classifiers)
r = ce.estimate(print_metrics=False)
```

```
print(r)
```

Входные данные: *fd* – объект *FormatDetector*,; *dl* – объект *DataLoaderDataPort*. Данные объекты получены в результате прошлого примера. Выходные данные: *r* – объект, содержащий результаты оценивания экземпляров обучающей и тестовой выборок.

6.3.4 Алгоритм АПССОП

Контрольный пример №1.

Описание: Обучение модели прогнозирования. Если при работе алгоритма не было получено сообщение об ошибках, то алгоритм отработал корректно.

```
params = DeepForecasterParameters()
ts = TSGenerator(np.random.randint(1, 10, 10000), params)
x = ts.get_data()
y = ts.get_data()

model = DeepForecaster(params)
model.build_model()
model.train(x, y)
model.save_model('my_model.keras')
```

Входные данные: значения установлены по умолчанию, *filename ='my_model.keras'* – имя файла для сохранения модели.

Выходные данные: обученная модель прогнозирования, записанную в формате модели keras.

Контрольный пример №2.

Описание: Прогнозирование данных с использованием обученной модели прогнозирования. Прогнозируется последовательность длиной один и 10 временных единиц. Если при работе алгоритма не было получено сообщение об ошибках, то алгоритм отработал корректно.

```
new_data = np.random.randint(1, 10, size=(10, 10, 1))

model = DeepForecaster(from_file='my_model.keras')
pred_1 = model.forecasting(new_data)
pred_10 = model.forecasting(new_data, 10)
```

Входные данные: *filename* – имя файла для сохранения модели.

Выходные данные: *pred_1* и *pred_10* – последовательности прогнозируемых значений (многомерный массив).

7. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

7.1 Состав и структура входных данных

1) Входные данные модуля ПОИНД сведены в единую Таблицу 7.1.1.

Таблица – 7.1.1. Входные данные модуля ПОИНД

Наименование данных	Обозначение	Структура данных	Способ ввода данных	Ограничения
Матрица признаков	F^M	Двумерный массив, состоящий из объектов и их признаков *(numpy.array)*	В виде аргумента *features_matrix*, через интерфейс вызова функции через оперативную память	Только численные значения, не более 1×10^3 столбцов и 5×10^6 строк
Типы данных признаков	F^T	Одномерный массив, длина которого соответствует количеству признаков *(numpy.array)*	В виде аргумента *features_types*, через интерфейс вызова функции через оперативную память	Возможные значения элементов *["num", "cat", None]*, не более 1×10^3 элементов
Наименования признаков	F^S	Одномерный массив, длина которого соответствует количеству признаков *(numpy.array)*	В виде аргумента *features_names*, через интерфейс вызова функции через оперативную память	Только строковые значения, не более 1×10^3 элементов
Обозначение		**Структура данных**	**Способ ввода данных**	**Ограничения**
Матрица меток	L^R	Двумерный массив, состоящий из объектов и их меток *(numpy.array)*	В виде аргумента *labels_matrix*, через интерфейс вызова функции через оперативную память	Только численные значения, не более 1×10^2 столбцов и 5×10^6 строк
Типы данных меток	L^T	Одномерный массив, длина которого соответствует количеству меток*(numpy.array)*	В виде аргумента *labels_types*, через интерфейс вызова функции через оперативную память	Возможные значения элементов *["num", "cat", None]*, не более 1×10^2 элементов
Наименования меток	L^S	Одномерный массив, длина которого соответствует количеству меток*(numpy.array)*	В виде аргумента *labels_names*, через интерфейс вызова функции через оперативную память	Только строковые значения, не более 1×10^2 элементов
Наименование данных				

Матрица признаков	F^M	Двумерный массив, состоящий из объектов и их признаков *(numpy.array)*	В виде аргумента *features_matrix*, через интерфейс вызова функции через оперативную память	Только численные значения, не более 1×10^3 столбцов и 5×10^6 строк
Типы данных признаков	F^T	Одномерный массив, длина которого соответствует количеству признаков *(numpy.array)*	В виде аргумента *features_types*, через интерфейс вызова функции через оперативную память	Возможные значения элементов *["num", "cat", None]*, не более 1×10^3 элементов
Наименования признаков	F^S	Одномерный массив, длина которого соответствует количеству признаков *(numpy.array)*	В виде аргумента *features_names*, через интерфейс вызова функции через оперативную память	Только строковые значения, не более 1×10^3 элементов
Матрица меток	L^R	Двумерный массив, состоящий из объектов и их меток *(numpy.array)*	В виде аргумента *labels_matrix*, через интерфейс вызова функции через оперативную память	Только численные значения, не более 1×10^2 столбцов и 5×10^6 строк
Типы данных меток	L^T	Одномерный массив, длина которого соответствует количеству меток*(numpy.array)*	В виде аргумента *labels_types*, через интерфейс вызова функции через оперативную память	Возможные значения элементов *["num", "cat", None]*, не более 1×10^2 элементов
Наименования меток	L^S	Одномерный массив, длина которого соответствует количеству меток*(numpy.array)*	В виде аргумента *labels_names*, через интерфейс вызова функции через оперативную память	Только строковые значения, не более 1×10^2 элементов

2) Входные данные модуля ИЗСНД сведены в единую Таблицу 7.1.2.

Таблица 7.1.2 – Входные данные модуля ИЗСНД

Наименование данных	**Обозначение**	**Структура данных**	**Способ ввода данных**	**Ограничения**
Вероятность, задающая порог отсечения значений для построения предикатов	δ	Вещественное число	В виде аргумента *probability_threshol*, через интерфейс вызова функции через оперативную память	$\delta \in [0, 1]$
Метрика оценки правил (название в строковом формате)	μ	Строковое значение	В виде аргумента *rule_metric*, через интерфейс вызова функции через оперативную память	Строка из фиксированного списка реализованных метрик
Признаки и метки обучающей выборки	$X_{i,}\ i=1..N,$ $Y_{i,}\ i=1..N$	Многомерный массив	В виде аргумента *data*, через интерфейс вызова функции через оперативную память	Объем массива данных ограничен объемом оперативной и жесткого диска
Метка положительного класса	Y_j	Вещественное число или строковое значение	В виде аргумента *positive_class_label*, через интерфейс вызова функции через оперативную память	Метка должна присутствовать в обучающей выборке
Имя признака, содержащего метки обучающей выборки	–	Строковое значение	В виде аргумента *class_column*, через интерфейс вызова функции через оперативную память	Признак должен присутствовать в обучающей выборке

3) Входные данные модуля АОССОП сведены в единую табл. 7.1.3

Таблица 7.1.3 – Входные данные модуля АОССОП

Наименование данных	**Обозначение**	**Структура данных**	**Способ ввода данных**	**Ограничения**
Признаки обучающей выборки	X_{ij}	Двумерный массив	В виде аргументов *x_train* и *X_train*, через интерфейс вызова функции через оперативную память	Не более 1×10^3 столбцов и 5×10^6 строк
Метки обучающей выборки	Y_{ij}	Одномерный массив	В виде аргумента *y_train*, через интерфейс вызова функции через оперативную память	Не более 1×10^3 столбцов и 5×10^6 строк
Признаки тестовой выборки	Z_{ij}	Двумерный массив	В виде аргументов *x_test* и *X_test*, через интерфейс вызова функции через оперативную память	Не более 1×10^3 столбцов и 5×10^6 строк
Метки тестовой выборки	–	Одномерный массив	В виде аргумента *y_test*, через интерфейс вызова функции через оперативную память	Не более 1×10^3 столбцов и 5×10^6 строк

Файл для сериализации	–	keras и pickle	В виде аргумента *saved_file* с путем к бинарному файлу	Существование пути к файлу с моделью
Файл для десериализации	–	keras и pickle	В виде аргумента *loaded_file* с путем к бинарному файлу	Существование пути к файлу с моделью; Корректность формата файла
Файл для определения его типа и загрузки данных	–	Набор строк из элементов, разделенных символами ‘,’ или ‘;’	В виде аргумента *file* с путем к файлу	Существование файла с моделью; Корректность формата файла
Признаки данных сложного объекта	–	Массив с данными	В виде аргумента *features*	Не более $1 \cdot 10^3$ столбцов и $5 \cdot 10^6$ строк
Метки данных сложного объекта	–	Массив с данными	В виде аргумента *labels(num_labels)*	Не более $1 \cdot 10^2$ столбцов и $5 \cdot 10^6$ строк
Выбранные ранее классификаторы	–	Программный объект	В виде аргумента *classifiers*, через интерфейс вызова функции через оперативную память	Нет

4) Входные данные модуля АПССОП сведены в единую табл. 7.1.4

Таблица 7.1.4 – Входные данные модуля АПССОП

Наименование данных	**Обозначение**	**Структура данных**	**Способ ввода данных**	**Ограничения**
Количество признаков данных	M	Число	В виде аргумента *n_features*, через интерфейс вызова функции через оперативную память	Должен быть равен количеству столбцов в матрице признаков
Размер временного окна при обучении, длина исторической последовательности	L	Число	В виде аргумента *look_back_length*, через интерфейс вызова функции через оперативную память	Сумма значений длины исторической последовательности для обучения и длины горизонта прогнозирования модели не может быть больше длины обучающей выборки
Горизонт прогнозирования	τ	Число	В виде аргумента *horizon*, через интерфейс вызова функции через оперативную память	
Тип блоков модели прогнозирования	–	Текстовая строка	В виде аргумента *block_type*, через интерфейс вызова функции через оперативную память	Должен быть равен ‘SimpleRNN’, ‘LSTM’ или ‘GRU’
Количество юнитов на каждом слое сети	–	Список чисел или словарь формата {‘units_<n>’}: Число}, где <n> – порядковый номер слоя	В виде аргумента *units*, через интерфейс вызова функции через оперативную память	–

Доля отбрасывания нейронов	–	Число с плавающий запятой	В виде аргумента *dropout*, через интерфейс вызова функции через оперативную память	В промежутке между 0 и 1
Функция активации на скрытом слое	–	Текстовая строка	В виде аргумента *hidden_activation*, через интерфейс вызова функции через оперативную память	Должна быть названием функции в keras.activations
Функция активации на выходном слое	–	Текстовая строка	В виде аргумента *output_activation*, через интерфейс вызова функции через оперативную память	Должна быть названием функции в keras.activations
Функция потерь	–	Текстовая строка	В виде аргумента *loss*, через интерфейс вызова функции через оперативную память	Должна быть равно ‘mse’ или ‘mae’
Путь к модели прогнозирования	–	Текстовая строка	В виде аргумента *from_file* с путем к бинарному файлу	Не более 255 символов
Путь к конфигурации модели прогнозирования	–	Текстовая строка	В виде аргумента *from_file_config* с путем к бинарному файлу	Не более 255 символов
Конфигурация рекуррентной нейронной сети	–	Словарь значений	В виде аргумента *from_config*, через интерфейс вызова функции через оперативную память	Нет
Размер временного окна для прогнозирования	Δ	Число	В виде аргумента *forecasting_data_length*, через интерфейс вызова функции через оперативную память	Нет
Матрица признаков исходного набора данных	**X**	Многомерный массив числовых значений	В виде аргумента *x*, через интерфейс вызова функции через оперативную память	Не более $1 \cdot 10^3$ столбцов и $5 \cdot 10^6$ строк

7.1.1 Подготовка входных данных

Дополнительной предобработки входные данные не требуют, однако модули компонента загружают часть данных их внешних файлов следующим образом.

1) Модуль ПОИНД: данные для работы модуля передаются в виде пути к файлу, содержащему данные, нуждающиеся в предобработке. При этом пользователю необходимо отделить признаки данных от их меток в соответствии с описанием использованного набора данных.

2) Модуль ИЗСНД: данные для работы модуля ИЗСНД передаются в виде пути к файлу, содержащему обучающие данные.

3) Модуль АОССОП: модель сериализуется во внешний файл, путь к которому передается через аргумент – saved_file, и который имеет бинарный формат, реализованный в библиотеке pickle (файл создается автоматически и не требует формирования

пользователем); модель десериализуется из внешнего файла, путь к которому передается через аргумент – loader_file, и который имеет бинарный формат, реализованный в библиотеке pickle (файл создается автоматически и не требует формирования пользователем). Входной набор данных для определения типа и загрузки загружается из файла, путь к которому передается через аргумент – file, и который текстовый формат в виде набора строк, элементы которых разделяются символами ‘,’ или ‘;’ (также, он может быть запакован с помощью gzip).

4) Модуль АПССОП: параметры моделей прогнозирования могут быть загружены из файла формата JSON, путь к которому передается через аргумент filename. Модель глубокого прогнозирования может быть загружена из бинарного файла библиотеки keras.models, путь к которому передается через аргумент from_file. Модель глубокого прогнозирования может быть загружена из файла конфигурации библиотеки keras.models, путь к которому передается через аргумент from_file_config.

7.2 Состав и структура выходных данных

1) Выходные данные модуля ПОИНД сведены в единую Таблицу 7.3.1.

Таблица 7.3.1 – Выходные данные модуля ПОИНД

Наименование данных	**Обозначение**	**Структура данных**	**Способ вывода данных**	**Ограничения**
Результаты анализа корректности типов данных	O^T	Строковые данные *(str)*	Вывод в консоль	Нет
Результаты устранения неполноты данных	O^E	Строковые данные *(str)*	Вывод в консоль	Нет
Результаты анализа информативности данных	O^M	Строковые данные *(str)*	Вывод в консоль	Нет
Предобработанные данные	–	Двумерный массив, состоящий из объектов и их признаков *(numpy.array)*	файл формата CSV	Не более 1×10^3 столбцов и 5×10^6 строк

2) Выходные данные модуля ИЗСНД сведены в единую табл. 7.3.2.

Таблица 7.3.2 – Выходные данные модуля ИЗСНД

Наименование данных	**Обозначение**	**Структура данных**	**Способ вывода данных**	**Ограничения**
Множество построенных	$U_k^{(j)}\left(d_k^l \in A_k^{(j)}\right) \rightarrow Y_j$	Массив объектов класса Rules	Вывод в консоль	Нет

ассоциативных правил класса				
Преобразованные данные	$X^m = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$	Объект pandas.DataFrame	Возвращаемый атрибут функции	Нет

3) Выходные данные модуля АОССОП сведены в единую табл. 7.3.3.

Таблица 7.3.3 – Выходные данные модуля АОССОП

Наименование данных	**Обозначение**	**Структура данных**	**Способ вывода данных**	**Ограничения**
Оцениваемые значения	X_{ij}	Двумерный массив данных	Вывод в консоль	Нет
Показатели качества оценивания	–	Словарь, список или текстовая строка	Вывод в консоль	Нет
Обученная модель по некоторому датасету	F	Внутренняя программная структура (keras, pickle)	Бинарный файл в файловой системе ОС	Нет
Графики параметров процесса обучения	–	Объект matplotlib.pypot	Вывод в консоль	Нет

4) Выходные данные модуля АПССОП сведены в единую табл. 7.3.4.

Таблица 7.3.4 – Выходные данные модуля АПССОП

Наименование данных	**Обозначение**	**Структура данных**	**Способ вывода данных**	**Ограничения**
Прогнозируемые значения	X^*	Многомерный массив числовых значений	Консоль, файл формата CSV	Нет
Показатели качества прогнозирования	–	Многомерный массив числовых значений	Консоль, файл формата CSV	Нет
Результаты прогнозирования	X'	Многомерный массив числовых значений	Консоль, файл формата CSV	Нет
Обученная модель прогнозирования по некоторому датасету	$RNN^{[N]}$	Программный формат модели (библиотека keras)	Файл формата Keras	Нет
Сформированная конфигурация модели прогнозирования	–	Словарь значений	Файл формата JSON	Нет

7.2.1 Интерпретация выходных данных

Дополнительной постобработки выходные данные не требуют, однако модули компонента сохраняют часть данных во внешние файлы следующим образом.

1) Модуль ПОИНД не сохраняет выходные данные во внешние файлы.

2) Модуль ИЗСНД не сохраняет выходные данные во внешние файлы.

3) Модуль АОССОП: Модель оценивания сериализуется во внешний файл, путь к которому передается через аргумент – filepath; и файл, который имеет бинарный формат, реализованный в библиотеке keras. В консоль выводятся графики параметров процесса обучения с использованием библиотеки matplotlib.pypot. Пример такого графика представлен на рисунке 7.2.1. По оси X отмечено количество эпох обучения, по оси Y – точность (аккуратность) оценивания. Синим цветом обозначены полученные значения при обучении, оранжевым – при валидации.

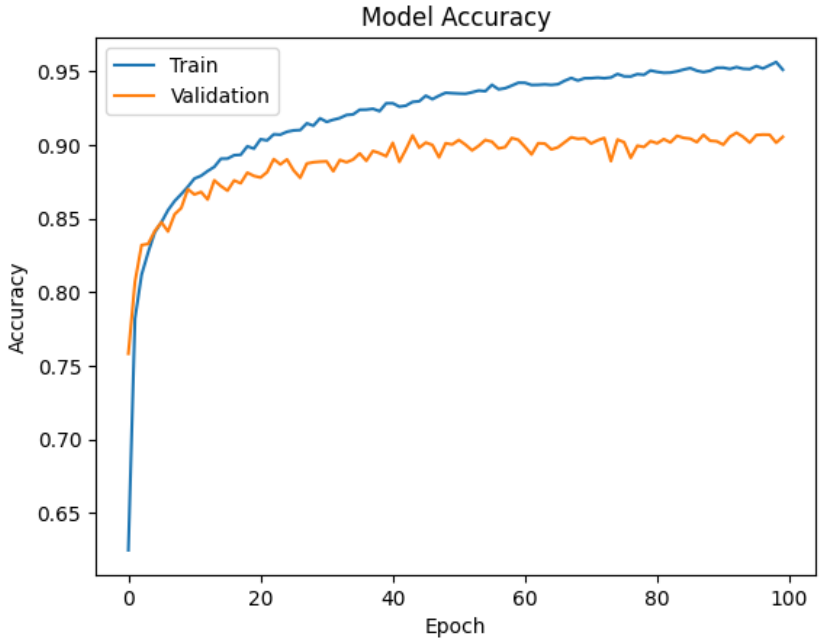

Рисунок 7.2.1 – Пример графика параметров обучения модели оценивания

Также в консоль выводятся результаты оценивания модели оценивания состояния СлОП в виде матрицы показателей, представленной на Рисунке 7.2.2. В верхней части матрицы каждая строка относится к обозначенной метке состояния (для приведенного примера от 0 до 7). Каждый столбец содержит значения для показателей точности (precision), полноты (recall), F-меры (f1-score) и общее количество экземпляров данного состояния в выборке (support). Строка «accuracy» показывает значения аккуратности для всех экземпляров. Строка «macro avg» содержит усреднение невзвешенного среднего значения для всех состояний. Строка «weighted avg» содержит усреднение взвешенного среднего значения для всех состояний.

	precision	recall	f1-score	support
0	0.87	0.82	0.84	822
1	0.87	0.92	0.90	1224
2	0.93	0.83	0.88	909
3	0.88	0.94	0.91	1211
4	0.87	0.88	0.87	1358
5	0.93	0.92	0.93	2372
6	0.94	0.88	0.91	1060
7	0.91	0.95	0.93	1374
accuracy			0.90	10330
macro avg	0.90	0.89	0.89	10330
weighted avg	0.90	0.90	0.90	10330

Рисунок 7.2.2 – Пример вывода матрицы показателей качества оценивания СлОП

4) Модуль АПССОП: Модель прогнозирования сериализуется во внешний файл, путь к которому передается через аргумент – filename; и файл который имеет бинарный формат, реализованный в библиотеке keras. В консоль выводятся графики параметров процесса обучения с использованием библиотеки matplotlib.pypot. Пример такого графика представлен на рисунке 7.2.1. По оси X отмечено количество эпох обучения, по оси Y – ошибка прогнозирования (MSE). Серым цветом обозначены полученные значения при обучении, фиолетовым – при валидации. Заголовок рисунка содержит название модели прогнозирования. В данном случае установлено по умолчанию для модели с параметрами: block_type=”GRU”, units = [128], dropout=0,1 и порядковым номером «0».

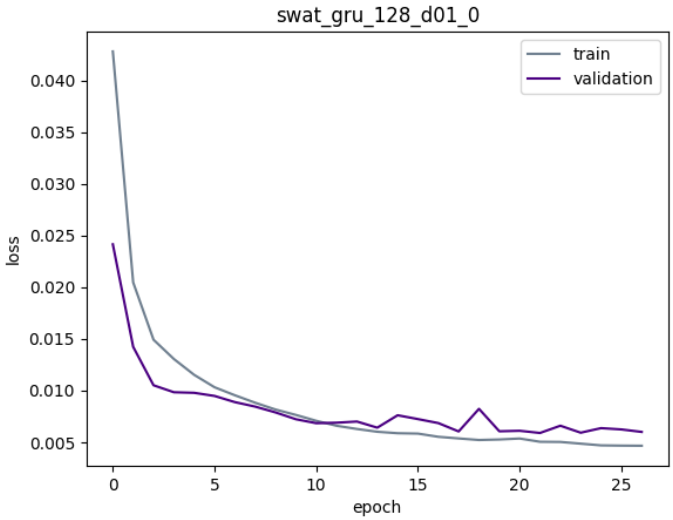

Рисунок 7.2.3 – Пример графика параметров обучения модели прогнозирования

Результаты оценки прогнозирования сохраняются во внешний файл формата CSV – filename. Пример отображения результата при выводе в консоль представлен на Рисунке 7.2.4. Каждая строка содержит показатели качества для отдельного признака данных, а последняя строка с индексом “ALL_FAETURES” – для всего набора данных. Столбцы

обозначены как «название_модели»_«показатель», где в качестве показателей выступают MSE, RMSE, MAE и R^2.

	gru_128_d001_MSE	gru_128_d001_RMSE	gru_128_d001_MAE	gru_128_d001_R2	gru_128_d01_MSE	gru_128_d01_RMSE
PIT501	0.025732	0.160411	0.137742	-1.725839	0.017916	0.133851
PIT502	0.010939	0.104588	0.087272	-4.580774	0.003375	0.058091
PIT503	0.024104	0.155254	0.134149	-1.862248	0.016191	0.127245
FIT601	0.006207	0.078784	0.062481	-0.718036	0.000444	0.021082
P601	0.001778	0.042165	0.033565	0.000000	0.000009	0.003019
P602	0.005472	0.073973	0.049060	-0.391316	0.001043	0.032299
P603	0.001892	0.043496	0.035521	0.000000	0.000009	0.003015
ALL_FEATURES	0.015569	0.108738	0.086770	-1.380874	0.010576	0.074726

Рисунок 7.2.4 – Пример вывода таблицы показателей качества прогнозирования СлОП

Результаты прогнозирования, полученные в виде массива значений, сериализуются во внешний бинарный файл формата NumPy. В консоль выводятся модно вывести графики полученных результатов с использованием библиотеки matplotlib.pypot. Пример таких графиков представлен на Рисунке 7.2.5. На оси X находятся отметки временных шагов, на оси Y – нормированные значения для каждого обозначенного признака. Синим цветом обозначено входное окно данных, зеленным – реальные значения. Полученные для каждой модели прогнозирования значения обозначены цветом, отраженным на легенде.

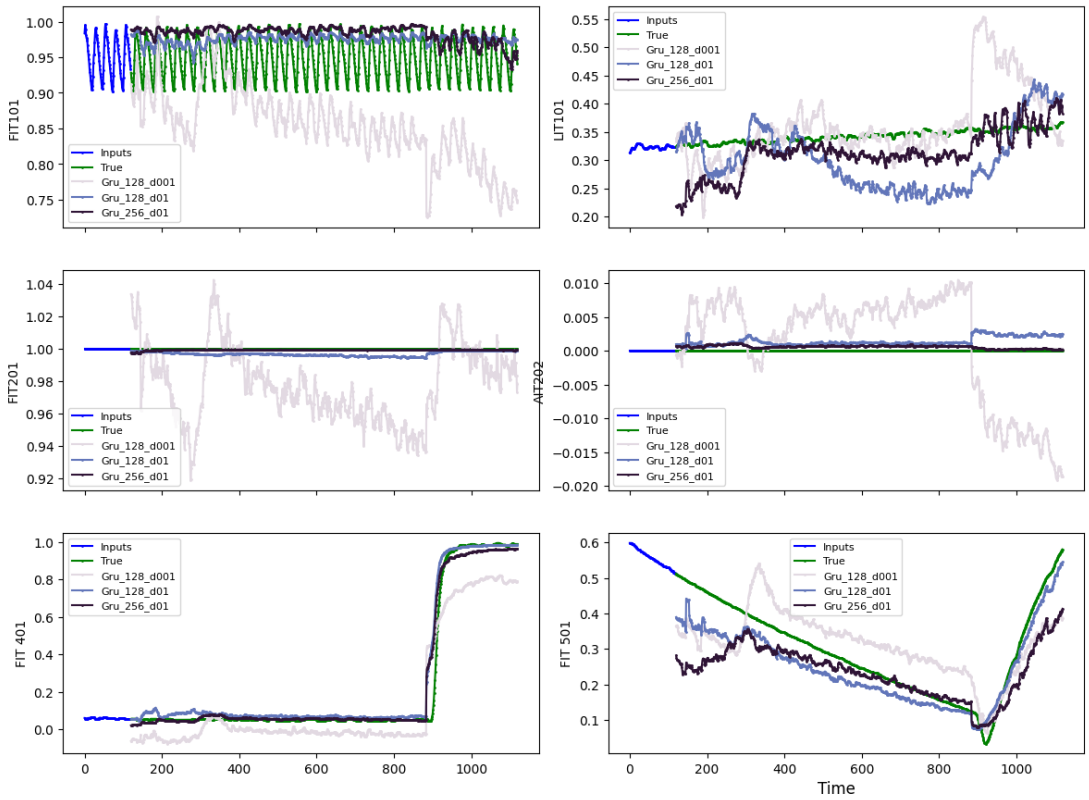

Рисунок 7.2.4 – Пример вывода таблицы показателей качества прогнозирования СлОП

8. СООБЩЕНИЯ

Компонент выводит следующие сообщения:

№	Текст сообщение	Значение сообщения
1	PassengerId with FEATURE_type=cat is incorrect. Changing to num (num=1.0, cat=0.8).	Сообщение содержит информацию о проверке корректности типа данных признака
2	iteration 2, fill 177 NaNs	Сообщение содержит информацию о процессе устранения неполноты данных, а именно – об итерации процесса кластеризации и количестве заполненных пустых значений
3	price is uninformative, can predict 1, while required 3	Сообщение содержит информацию о процессе анализа информативности признаков, неинформативные признаки рекомендуется удалить
4	delete feature=SibSp as it correlates with feature=PassengerId	Сообщение содержит информацию о процессе устранения мультиколлинеарности данных, а именно рекомендацию об удалении признака, если он коррелирует с другим признаком
5	“Test accuracy: N”, где N – дробное число до 3-го знака	Сообщение содержит информацию о точности обучения модели оценивания состояния сложного объекта.
6	“{MN} of {CN} on {N} sample ({L} instances): {M}”, где MN – имя метрики (precision/accuracy – точность, recall – полнота, fscore – F-метрика); CN – имя классификатора; N – тип процесса для вычисления метод (training – обучение, testing – тестирование); L – число меток; M – значение метрики	Сообщение содержит информацию о точности обучения и тестирования модели оценивания состояния сложного объекта.
7	“File with forecasting model does not exist”	Сообщение сигнализирует об отсутствии модели предсказания по заданному пути.
8	“Too many values for categorical feature '{F}'. Delete feature from data”, где F – число признаков	Сообщение сигнализирует о превышении допустимого числа признаков.
9	“The proportion of the training sample is not in the interval (0, 1)”	Сообщение сигнализирует о том, что доля обучающей выборки не входит в интервал от 0 до 1.
10	“File with normalization model does not exist”	Сообщение сигнализирует об отсутствии модели нормализации по заданному пути.

11	“The length of the samples is not equal”	Сообщение сигнализирует о том, что длина реальных и спрогнозированных данных не совпадают.
12	“{FC}”, где FC – показатели качества предсказания	Сообщение содержит показатели качества проведенного предсказания.
13	“Done”	Сообщение сигнализирует о завершении процесса тестирования.
14	“Value of the "{0}" argument must be more than {1}”	Сообщение предупреждает об ошибке передачи значения (слишком маленькое)
15	“Value of the "{0}" argument must be less than {1}”	Сообщение предупреждает об ошибке передачи значения (слишком большое)
16	“Value of the "{0}" argument must be more than {1} or equal”	Сообщение предупреждает об ошибке передачи значения (должно быть больше или равно заданному)
17	“Value of the "{0}" argument must be less than {1} or equal”	Сообщение предупреждает об ошибке передачи значения (должно быть меньше или равно заданному)
18	“Value of the "{0}" argument must be in list {1}”	Сообщение предупреждает об ошибке передачи значения (должно быть значением из списка)
19	“Type of the "{0}" argument must be {1}”	Сообщение предупреждает об ошибке передачи значения (некорректный тип переменной)
20	"Attribute '{name}' must be an integer"	
21	"Attribute '{name}' must be string"	
22	"Attribute '{name}' must be an list or dictionary"	
23	"Attribute '{name}' must be float or int or None"	
23	"Values of attribute '{name} must be integers"	
24	"Attribute '{name}' must be 'SimpleRNN' or 'LSTM' or 'GRU'"	Сообщение предупреждает об ошибке передачи значения (должно быть значением из списка)
25	"Invalid attribute '{name}'"	Сообщение предупреждает об ошибке присвоения атрибута класса (несуществующий атрибут)

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
	измененных	замененных	новых	Аннулированных					