МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

СОГЛАСОВАНО
Генеральный директор Ассоциации «Искусственный интеллект в промышленности»

_____ Т.М. Супатаев
___ _____ 2024 г.

УТВЕРЖДАЮ
Научный руководитель ИЦ СИИП Университета ИТМО

_____ А.В. Бухановский
___ _____ 2024 г.

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

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.00853-02 13 21 - ЛУ

Представители
Организации-разработчика

Руководитель разработки

_____ И.В. Котенко
_____ 2024 г.

Нормоконтролер

_____А.В. Киреева
_____ 2024 г.

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН
RU.СНАБ.00853-02 13 21 - ЛУ

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

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ.00853-02 13 21

ЛИСТОВ 51

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

2024

АННОТАЦИЯ

Документ содержит описание программы «Компонент автономного оценивания и прогнозирования состояния сложных объектов и процессов на основе интеллектуальной обработки событий в условиях неопределенности и недостоверности данных» RU.СНАБ.00853-02 13 21. Данное программное обеспечение предназначено для предобработки, автономного оценивания и прогнозирования состояния сложных объектов и процессов и 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

— Программа «Компонент предобработки, автономного 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.00853-01 02 21 разработан в соответствии с мероприятием М1.3.3 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. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

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 сырых и нечетких данных[1], что может быть задействовано на подготовительной стадии оценивания и прогнозирования состояния СлОП (алгоритм ПОСНД);

[1] 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 (коэффициентами дискретного вейвлет-преобразования) и действиями в предметной области оценивания и прогнозирования состояния СлОП, а также извлечением множества действий (выводов), вытекающих из этих знаний (алгоритм ИЗСНД);

— автономное[2] 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 Классы решаемых задач

Компонент позволяет решить следующие классы задач:

1) Предобработка сырых и нечетких данных о СлОП (алгоритмы ПОСНД (коррекция типов, устранение неполноты и мультиколлинеарности входных данных) и ИЗСНД (извлечения фрагментов знаний, имеющихся в данных, в виде ассоциативных правил вида «ЕСЛИ <посылка>, ТО <следствие>»)).

2) 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)).

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

3) 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 каждый алгоритм в отдельности уже может быть полезен разработчикам, однако для решения классов технических задач выгодно совместное их использование.

2.3 Функциональные ограничения на применение

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

1) для алгоритма ПОСНД устанавливаются:

— порог уникальности значений признаков, устанавливаемый 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 АОССОП 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 Область применения

Модули разработанного компонента могут быть применены для таких объектов, как компьютерные системы и сети большой размерности (включая Интернет вещей, киберфизические системы), автономные робототехнические комплексы (беспилотные летательные аппараты, беспилотные автомобили и др.) и т.п.

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.

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.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

3.1 Используемые методы

Для удобства описания, представление задействованных методов было условно разделено в соответствии с классами задач, на решение которых они направлены: предобработка данных о СлОП и снижение их размерности; автономное оценивание состояния СлОП; автономное прогнозирование состояния СлОП. Рассмотрим каждый набор методов более подробно.

1) Предобработка данных о СлОП и снижение их размерности.

Используемые в данном классе задач методы решают задачи коррекции типов данных, устранения неполноты и мультиколлинеарности данных, а также извлечения фрагментов знаний в виде ассоциативных правил. При этом для работы методов используются входные данные в табличной форме следующего вида:

$$D^I = (F^M, F^T, F^S, L^R, L^T, L^S, E^C),$$

где $F^M = \{F^M_{on}\},\ o \in 1..O, n \in 1..N$ – матрица признаков, O – количество объектов, N – количество признаков; $F^T = \{F^T_n\}, n \in 1..N$ – вектор типов признаков; $F^S = \{F^S_n\}, n \in 1..N$ – вектор имен признаков; $L^R = \{L^R_{om}\},\ o \in 1..O, m \in 1..M$ – матрица лейблов, M – количество лейблов; $L^T = \{L^T_m\}, m \in 1..M$ – вектор типов лейблов; $L^S = \{L^S_m\}, m \in 1..M$ – вектор имен лейблов; $E^C = \{E^C_k\}, k \in 1..K$ – вектор координат пустых значений, K – количество пустых значений.

Рассмотрим данные методы более подробно.

Коррекция типов данных происходит на основе статистических критериев, связанных с количеством уникальных значений признака, обнаружением последовательностей значений признака, анализом текстовых названий признаков, а также анализом дробных значений для представления некорректных FLOAT (*.0(0)) в качестве INT.

Устранение неполноты данных происходит на основе статистического усреднения данных в рамках отдельных кластеров. Разбиение данных на кластеры возможно на основе таких подходов, как K-means, Mini Batch K-means, Mean-shift, OPTICS, Bisecting K-means.

Устранение мультиколлинеарности данных происходит на основе коэффициентов корреляции Пирсона и Спирмена, а также дисперсионного анализа (ANOVA).

В основе методов извлечения фрагментов знаний в виде ассоциативных правил лежат следующие математические соотношения. Обучающие данные, характеризующие СлОП, заданы множеством описаний $\mathbf{X} = \{X_i\}$, $i = 1, \ldots, I$, СлОП (экземпляров), каждому из которых поставлена в соответствие метка класса из множества $\mathbf{Y} = \{Y_j\}$, $j = 1, \ldots, J$, задающая состояние СлОП в некоторый момент времени. Метки классов $Y_j \in \mathbf{Y}$ (состояния СлОП) могут быть представлены в категориальный шкале (в простейшем случае, булевыми) или быть линейно упорядоченными.

Признаки $F_k \in \mathbf{F}$, каждый из которых имеет область допустимых значений $\mathbf{D}_k = \{d_k^l\}$, $k = 1, \ldots, K$, $l = 1, \ldots, L$, характеризующие описания СлОП, $X_i \in \mathbf{X}$, также могут быть представлены в категориальной, булевой и/или числовой шкалах, а также могут быть текстами на естественном языке. Числовые признаки должны предварительно быть квантованы и далее могут рассматриваться как дискретные линейно упорядоченные признаки. Для этого можно использовать стандартные процедуры дискретизации. Для текстов необходимо предварительно выполнить процедуру векторизации с помощью стандартной процедуры, например, используя метрику TF-IDF.

Для построения ассоциативных правил на множестве обучающих данных на первом шаге необходимо объединить отдельные значения $d_k^l \in \mathbf{D}_k$ каждого признака в агрегаты $\mathbf{A}_k^{(j)}$, обладающие общим свойством по отношению к некоторому состоянию СлОП, заданному меткой класса $Y_j \in \mathbf{Y}$. Таким образом, в ходе построения агрегатов область значений $\mathbf{D}_k$ каждого атрибута $F_k \in \mathbf{F}$ разделяется на непересекающиеся подмножества $\mathbf{A}_k^{(j)} \subseteq \mathbf{D}_k$, поставленные в соответствие некоторым классам множества $Y_j \in \mathbf{Y}$. Подмножества $\mathbf{A}_k^{(j)}$ строятся таким образом, чтобы каждое из них содержало те значения признака $F_k \in \mathbf{F}$ из множества его значений $\mathbf{D}_k$, которые более характерны для описаний СлОП, $X_i \in \mathbf{X}$, принадлежащих классу Y_j, чем для описаний, принадлежащих другим классам.

Для построений подмножеств-агрегатов могут быть использованы различные метрики оценки частотности. При этом предлагается использовать эвристический метод, основанный на наивном байесовском подходе, выбор которого обусловлен результатами, полученными в выполненных ранее экспериментах. А именно, предполагается, что некоторое значение $d_k^l \in \mathbf{D}_k$ признака $F_k \in \mathbf{F}$ принадлежит подмножеству $\mathbf{A}_k^{(j)}$, $d_k^l \in \mathbf{A}_k^{(j)}$, в том случае, когда

$$p(Y_j \mid d_k^l) > p(\bar{Y}_j \mid d_k^l) + \delta. \qquad (3.1.1)$$

где $p(Y_j \mid d_k^l)$ – условная вероятность класса Y_j при $F_k = d_k^l$, $p(\bar{Y}_j \mid d_k^l)$ – это условная вероятность любого класса, кроме класса Y_j, при $F_k = d_k^l$, а δ – некоторая положительная константа, значение которой позволяет регулировать мощность формируемых подмножеств-агрегатов.

Отметим, что с помощью теоремы Байеса выражение (3.1.1) можно привести к виду, требующему вычисления только $p(d_k^l \mid Y_j)$ *и* $p(d_k^l \mid \bar{Y}_j)$ и априорных вероятностей классов. И такие вероятности могут быть подсчитаны всего за один проход по обучающим данным, что делает описываемый подход вычислительно эффективным.

Далее предлагается построить унарные предикаты $U_k^{(j)}(d_k^l \in \mathbf{A}_k^{(j)})$, каждый из которых принимает истинное значение в том случае, когда значение d_k^l признака F_k для некоторого описания СлОП, $X_i \in \mathbf{X}$ принадлежит подмножеству-агрегату $\mathbf{A}_k^{(j)}$, $d_k^l \in \mathbf{A}_k^{(j)}$. Таким образом, подмножество $\mathbf{A}_k^{(j)}$ является областью истинности унарного предиката $U_k^{(j)}$, а для значений $d_k^l \notin \mathbf{A}_k^{(j)}$ предикат $U_k^{(j)}$ принимает значение «ложь».

Используя построенные предикаты далее в качестве посылки, а метки класса $F_k \in \mathbf{F}$, соответствующие некоторому состоянию СлОП, в качестве следствия, могут быть сформированы ассоциативные правила класса следующего вида:

$$U_k^{(j)}\left(d_k^l \in \mathbf{A}_k^{(j)}\right) \to Y_j, j = 1, \dots, J;\ k = 1, \dots, K;\ l = 1, \dots, L. \qquad (3.1.2)$$

Такие правила можно интерпретировать как знания, описывающие связи между понятиями и действиями в предметной области оценивания и прогнозирования состояния СлОП, а также множество действий (выводов), вытекающих из знаний.

С целью сокращения множества подученных правил и выявления наиболее информативных из них для каждого правила далее необходимо рассчитать значение метрики информативности $\mu(U_k^{(j)}, Y_j)$. Предполагается, что метрика информативности поступает на вход метода в качестве параметра. В простейшем случае может использоваться классическая метрика уверенности или другая, реже используемая метрика, например, мера Клозгена или коэффициент регрессии событий, которые претендуют на то, чтобы оценивать также силу причинной связи между посылкой и следствием правила. Важным условием для метрики является ее несимметричность.

После вычисления значений выбранной метрики для каждого правила, множество правил сортируется в порядке убывания модуля значения метрики. В итоговом множестве правил останутся правила вида (3.1.1), удовлетворяющие условию: $\left|\mu\left(U_k^{(j)}, Y_j\right)\right| \geq$ M, где M

– это пороговое значение метрики, которое выбирается экспериментально, исходя из мощности полученного множества правил и среднего значения модуля метрики для правил.

В результате такой фильтрации итоговое множество правил будет содержать только наиболее «сильные» правила, отражающие наиболее «сильные» связи параметров описаний СлОП с его состоянием. Полученное множество правил можно также интерпретировать как новое булево признаковое пространство, соответственно к нему можно применять любые методы сокращения пространства признаков, позволяющие обрабатывать признаки в булевой шкале.

2) Автономное оценивание состояния СлОП

Автономное оценивание текущего состояния сложных объектов и процессов основывается на математических соотношениях из линейной алгебры и дифференциального исчисления. При этом автономность процесса заключается в решении задачи при минимальном участии человека. В частности, метод градиентного спуска является основой метода обратного распространения ошибки, который используется для обучения нейросетевых структур данного вида. Концептуальная модель системы, на оценивание и прогнозирование состояний и процессов функционирования которой направлены разрабатываемые методы, включает элементы $\{V_t\}_{t\in T}$, где $V_t = V_t(p_1, \dots, p_n)$ – кортеж, включающий n переменных $p_i, i = 1 \dots n$, описывающих состояние системы в момент времени t. Множество переменных $P = P_d \cup P_e$, где P_d – характеристики (параметры) системы, пригодные для технического диагностирования системы [17], P_e – параметры внешнего окружения системы и внешних воздействий. Множество рассматриваемых кортежей включает следующие выборки: обучающую выборку признаков $X_{ij}, = 1 \dots N$; обучающую выборку меток классов состояний сложных объектов $Y_{ij} = 1 \dots N$; тестовую выборку признаков $Z_{ij} = 1 \dots M$.

Решаемая научная задача по разработке методов оценивания состояния сложных объектов и процессов описывается функцией F, сопоставляющей следующие входные и выходные данные:

$$F: \left\{\left(p_1^{(t_i)}, \dots p_n^{(t_i)}, t_i\right)\right\}_{i\in\mathbb{N}, t\in T_0, T_0\subseteq T} \to \left\{\left(m_1^{(t_i)}, \dots m_N^{(t_i)}, pr_i\right)\right\}_{i\in\mathbb{N}, t\in T_0, T_0\subseteq T}, \quad (3.1.3)$$

где $m_j^{(t_i)}$ – значение целевого показателя системы в момент времени t_0. Величина pr_i – скалярная (либо векторная) априорно формируемая вероятностная оценка соответствия результирующего вектора $(m_1^{(t_i)}, \dots m_r^{(t_i)})$ фактическому состоянию системы в момент времени t_0. В рамках решаемой задачи набор представленных параметров $p_1^{(t_i)}, \dots p_n^{(t_i)}$

представляет собой вектор признаков, характеризующих состояние системы в момент времени t_0, а параметры $m_1^{(t_i)}, \dots m_N^{(t_i)}, pr_i$ задают вероятностную принадлежность к возможным классам состояния системы в этот момент времени. Тем самым функция F предназначена для классификации объекта или процесса в соответствии с его описательными параметрами $p_1^{(t_i)}, \dots p_n^{(t_i)}$.

Каждый из показателей $m_j^{(t_i)}$ может принимать бинарное или числовое значение. Примерами показателей со значениями $m_j^{(t_i)}$ являются:

— наличие или отсутствие функциональных неисправностей в системе в момент времени t_0;

— наличие или отсутствия в системе дефектов материалов физических конструкций системы в момент времени t_0;

— наличие или отсутствие непреднамеренных и преднамеренных воздействий на систему в момент времени t_0.

Задача оценивания включает оценивание текущего состояния в условиях обладания данными, предваряющими данное состояние, описывающими данное состояние и, возможно, некоторой порцией данных, описывающих несколько последующих состояний.

Для решения задачи оценивания текущего состояния сложных объектов и процессов выбран ряд методов:

— традиционного МО – наивный Байесовский классификатор (Naïve Bayes, NB), линейная регрессия (Logistic Regression, LR) и дерево решений (Decision Tree, DT);

— глубоких нейронных сетей (ГНС) – сверточная нейронная сеть (Convolutional Neural Network (CNN), автокодировщик (Autoencoder, AE), гибридная сверточная нейронная сеть с управляемым рекуррентным блоком (Gated Recurrent Unit, GRU) и генеративно-состязательная сеть (Generative Adversarial Network, GAN).

3) Автономное прогнозирование состояния СлОП

Автономное прогнозирование состояний сложных объектов и процессов основывается на следующих математических соотношениях. Обучающие данные, характеризующие СлОП, заданы множеством описаний (экземпляров), представляющих собой упорядоченную во времени последовательность векторов характеристик СлОП:

$$\mathbf{X} = \{X_t\},\ t = 1, \dots, T, \qquad (3.1.3)$$

где T – длина временного ряда (количество экземпляров).

Каждый экземпляр содержит числовой вектор характеристик состояния СлОП:

$$X_t = \{x_{mt}\},\ m = 1,\ \ldots,\ M, \tag{3.1.4}$$

где M – фиксированная длина вектора (количество характеристик).

Для обучаемой модели АПССОП задается параметр длины исторической последовательности данных для прогнозирования L – количество упорядоченных во времени экземпляров $\{X_t, \ldots, X_{t+L}\}$, необходимых для прогнозирования последующего экземпляра X_{t+L+1}.

На основе обучающих данных при помощи функции G производится формирование генератора временных рядов, представляющего собой формат данных, состоящих из пары (X^p, X^d):

$$\begin{gathered} G\colon \mathbf{X} \rightarrow (\mathbf{X}^p, \mathbf{X}^d) \\ \mathbf{X}^p = \left\{X_1^p, \ldots, X_{T-L-1}^p\right\},\ X_t^p = \{X_t, \ldots, X_{t+L}\},\ X_t^p \in \mathbf{X}^p \\ \mathbf{X}^d = \left\{X_1^d, \ldots, X_{T-L-1}^d\right\},\ X_t^d = X_{t+L+1},\ X_t^d \in \mathbf{X}^d, \end{gathered} \tag{3.1.5}$$

где $\mathbf{X}^p$ – множество пакетов, упорядоченных во времени исторических экземпляров для прогнозирования, $\mathbf{X}^d$ – множество целевых экземпляров для прогнозирования. Целевые экземпляры представляют собой ожидаемый результат прогнозирования: любому пакету X_i^d, соответствует последующий во времени экземпляр X_i^d.

Множество рассматриваемых кортежей включает следующие выборки: обучающую выборку $\boldsymbol{X}_r = \{X_1, \ldots, X_t\},\ t < T$, и тестовую выборку $\boldsymbol{X}_e = \{X_{t+1}, \ldots, X_T\}$. Выборки строятся таким образом, что обучающая выборка содержит временной ряд данных, предшествующий временному ряду в тестовой выборке. Соотношение объемов обучающей и тестовой выборок является опциональным. Классическим способом разделения исходной выборки является разделение с соотношением 8:2 (80% выборки является обучающей, 20% – тестовой) или 9:1 (90% выборки является обучающей, 10% – тестовой).

Прогноз осуществляется таким образом, что для предсказания параметров событий в следующий момент времени необходимо проанализировать предшествующие события во временном окне размерностью L. Задача прогнозирования состояний сложных объектов и процессов может быть описана при помощи функции ϕ, сопоставляющей следующие входные и выходные данные:

$$\begin{gathered} \phi\colon\ \{X_t, \ldots, X_{t+L}\} \rightarrow X_{t+L+1}, \\ \text{или } \phi\colon\ X_t^p \rightarrow X_t^d, t \in T. \end{gathered} \tag{3.1.6}$$

В качестве архитектур нейронных сетей предлагается использовать следующие:

— простая рекуррентная сеть (Simple Recurrent Network, SimpleRNN);

— блок долгой краткосрочной памяти (Long Short-Term Memory, LSTM);

— управляемый рекуррентный блок (Gated Recurrent Units, GRU).

Рекуррентная сеть содержит набор блоков, в которых сохраняются предыдущие значения узлов скрытого слоя. Фиксированные обратные связи передают значения предыдущих состояний узлов скрытого слоя сети, что позволяет сети сохранять свое предыдущее состояние, обеспечивая тем самым предсказание последовательностей.

Простую RNN можно описать как:

$$h_t = \sigma_h\left(W_{xh}X_t^h + W_{hh}h_{t-1} + b_h\right)$$
$$y_t = \sigma_y\left(W_{hy}h_t + b_y\right) \quad (3.1.7)$$

где y_t – вектор выходного слоя, h_t –вектор текущего скрытого состояния, h_{t-1} – вектор предыдущего скрытого состояния, W_{jk} – матрица весов для преобразования вектора j в компоненту вектора k, b_j – вектор смещения вектора j, σ_h, σ_y – функции активации.

Блоки LSTM применяются с целью кодирования признаков предшествующих экземпляров X^h, подающихся на входной слой нейронной сети. LSTM-блоки включают вентили (гейты) i_t, f_t, o_t, в форме логистических функций для проверки потоков данных, а также вектор состояний c_t, который выступает в виде внутренней памяти для обновления информации о текущем и предыдущем состояниях:

$$i_t = \sigma_g(W_{xi}X_t^h + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$
$$f_t = \sigma_g\left(W_{xf}X_t^h + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$o_t = \sigma_g\left(W_{xo}X_t^h + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o\right) \quad (3.1.8)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c\left(W_{xc}X_t^h + W_{hc}h_{t-1} + b_c\right)$$
$$h_t = o_t \circ \sigma_t(c_t)$$

где h_t –вектор текущего скрытого состояния, h_{t-1} – вектор предыдущего скрытого состояния, i_t – вектор входного вентиля (вес получения новых данных), f_t – вектор вентиля забывания (вес сохранения предшествующих данных), o_t – вектор выходного вентиля (кандидат на выход), W_{jk} – матрица весов для преобразования вектора j в компоненту вектора k, b_j – вектор смещения вектора j, σ_g – функция активации, использующая сигмоиды, σ_c – функция активации, использующая гиперболический тангенс, $\circ$ – произведение Адамара (покомпонентное произведение двух матриц).

Блоки GRU содержат меньше параметров:

$$z_t = \sigma_g(W_{xz}X_t^h + W_{hz}h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_{xr}X_t^h + W_{hr}h_{t-1} + b_r) \quad (3.1.9)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_c(W_{xh}X_t^h + W_{hh}(r_t \circ h_{t-1})$$

где h_t – вектор текущего скрытого состояния, h_{t-1} – вектор предыдущего скрытого состояния, z_t – вектор входного вентиля (вес получения новых данных), r_t – вектор вентиля забывания (вес сохранения предшествующих данных), W_{jk} – матрица весов для преобразования вектора j в компоненту вектора k, b_j – вектор смещения вектора j, σ_g – функция активации, использующая сигмоиды, σ_c – функция активации, использующая гиперболический тангенс, $\circ$ – произведение Адамара (покомпонентное произведение двух матриц).

Обозначим нейронную сеть из N слоев как $LSTM^{[N]}$, тогда ее скрытые состояния вычисляются рекуррентно и последнее скрытое состояние является выходным вектором:

$$\phi = \begin{cases} h_t^{[1]} = LSTM^{[1]}\left(X_t^d\right) \\ h_t^{[2]} = LSTM^{[2]}\left(h_t^{[1]}\right) \\ \dots \\ X_t^d = h_t^{[N]} = LSTM^{[N]}\left(h_t^{[N-1]}\right) \end{cases}, \tag{3.1.10}$$

где $[n], n = 1, \dots, N$ – порядковый номер слоя нейронной сети.

Таким образом,

$$X_t^d = \phi\left(X_t^h\right).$$

Выходными данными являются прогнозируемые состояния СлОП за заданный отрезок времени Δ:

$$\begin{gathered} \Phi: \ \mathbf{X} \to \mathbf{X}^*, \\ \mathbf{X}^* = \{X_{T+1}^*, \dots, X_{T+\Delta}^*\}, \end{gathered} \tag{3.1.11}$$

где **X*** – множество прогнозируемых векторов состояний СлОП.

3.2 Алгоритмы компонента

3.2.1 Алгоритм ПОСНД

Алгоритм предназначен для работы с подаваемыми на вход наборами данных и включает в себя коррекцию типов данных, а также устранение неполноты и мультиколлинеарности данных о сложных объектах или процессах.

Алгоритм является вспомогательным для других алгоритмов, реализует функции сильного ИИ в части улучшения качества предоставляемых данных и может быть задействован на подготовительной стадии оценивания и прогнозирования состояния.

Блок-схема алгоритма представлена на рис. 3.2.1.

На первом шаге алгоритма осуществляется анализ корректности типов данных для каждого признака отдельно. Корректность типа данных оценивается на основе количества уникальных значений признака (блок «Уникальные значения» на рис. 3.2.1),

обнаруженных последовательностей (блок «Последовательности» на рис. 3.2.1), анализа названий признаков (блок «Названия признаков» на рис. 3.2.1), а также на основе обоснованности использования дробных значений, в случае их наличия (блок «Дробные значения» на рис. 3.2.1). Вердикт о корректности типа данных выносится алгоритмами оценки отдельно, в то время как итоговое решение принимается на основе всех четырех оценок с учетом их весовых коэффициентов. Незаполненные поля игнорируются. Информация о корректировке типов данных сохраняется. И т.к. решение по каждому признаку принимается независимо от решений по остальным, то обработка каждого признака происходит в параллельном режиме.

Второй шаг алгоритма направлен на устранение неполноты данных на основе статистического усреднения данных в рамках отдельных кластеров. Вначале данные разбиваются на отдельные кластеры на основе таких подходов, как K-means, Mini Batch K-means, Mean-shift, OPTICS, Bisecting K-means. Метод кластеризации может быть задан вручную оператором или выбран алгоритмом автоматически в соответствии с заданным по умолчанию значением. В дальнейшем, каждый признак каждого кластера обрабатывается параллельно. Устранение неполноты данных представляет собой автоматическое заполнение пропущенных значений признаков на основе среднего, медианного или регрессионного значения по кластеру, а также на основе замены специальным значением, указанным оператором (например, EMPTY). Данные о произведенных преобразованиях сохраняются.

На третьем шаге осуществляется устранение мультиколлинеарности данных за счет удаления признаков низкой информативности. Анализ информативности признаков осуществляется параллельно для каждой пары признаков. При этом в зависимости от типов признаков (только численные, только категориальные, численные и категориальные), информативность вычисляется с помощью критерия корреляции Пирсона или Спирмена, а также методом дисперсионного анализа (Крамер). По умолчанию, логика работы алгоритма, следующая: если признаки категориальные, то используется дисперсионный анализ Крамера, численные – корреляция Пирсона, один численный, другой категориальный (или типы неизвестны) – Спирмена. Если необходимо, для пользователя предусмотрена возможность изменения задействованных критериев. Порог информативности также может быть задан пользователем, в противном случае алгоритм использует значение по умолчанию, равное 0.70. Итогом работы алгоритма являются плоские данные в табличной форме, в которых указаны корректные типы данных, заполнены пропущенные значения, а также удалены неинформативные признаки.

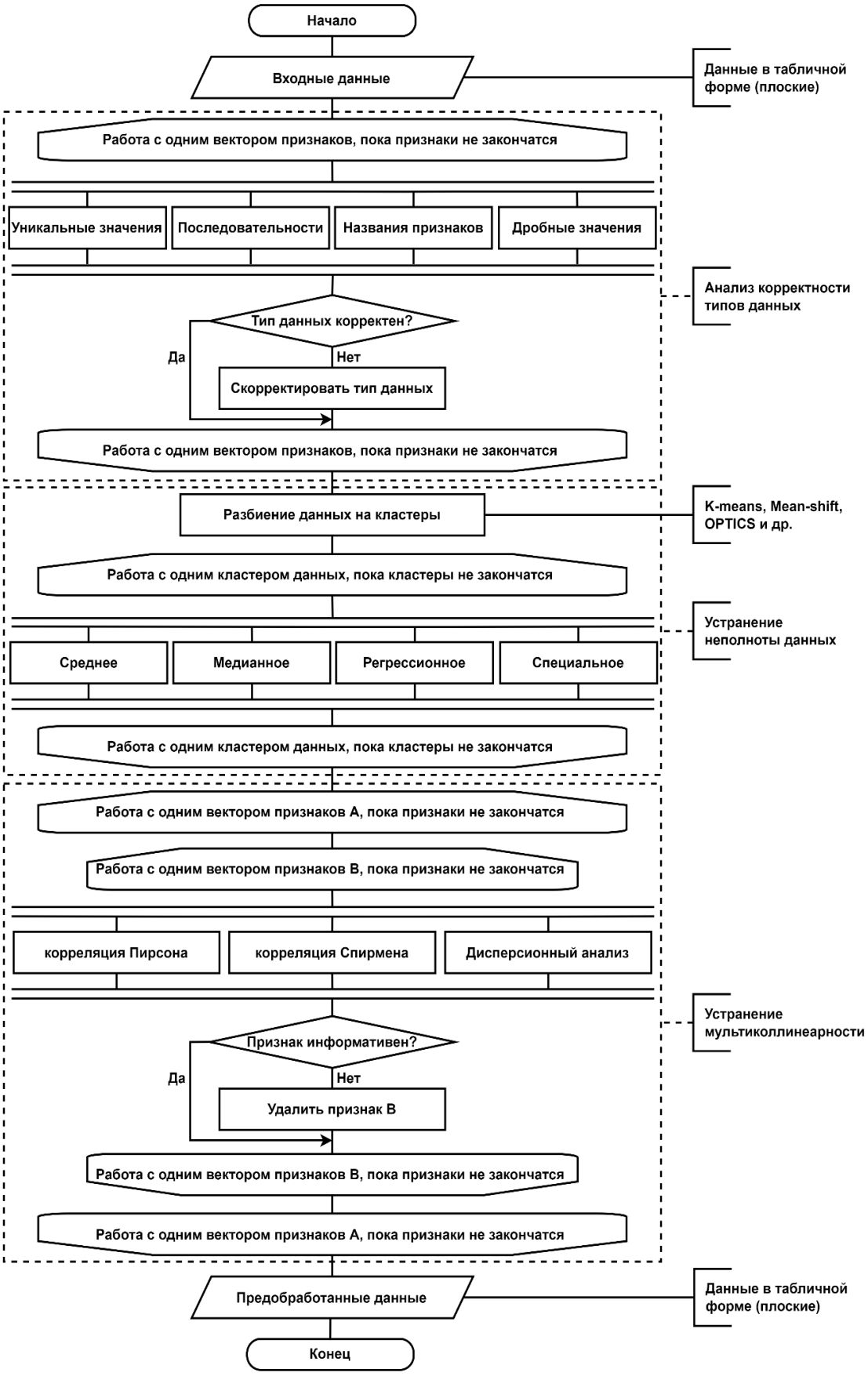

Рисунок 3.2.1 – Блок-схема алгоритма ПОСНД

3.2.2 Алгоритм ИЗСНД

Алгоритм предназначен для извлечения фрагментов знаний, имеющихся в данных о сложных объектах, в виде ассоциативных правил (вида «ЕСЛИ <посылка>, ТО <следствие>»), содержащих в правой части (следствии) метку класса (англ. class association rules). Блок-схема алгоритма ИЗСНД представлена на рис. 3.2.2.

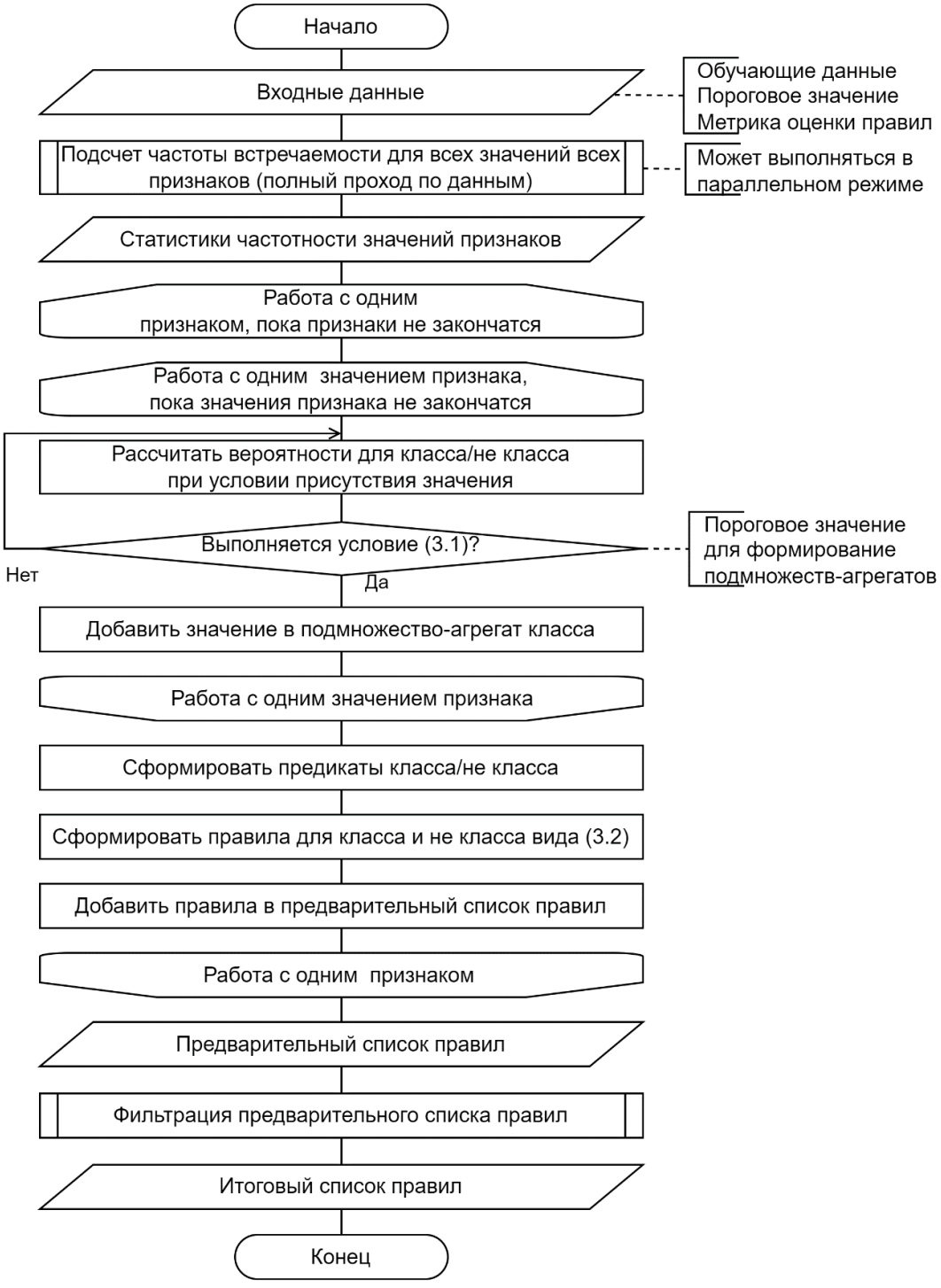

Рисунок 3.2.2 – Блок-схема алгоритма ИЗСНД

На первом шаге алгоритма вычисляется частота встречаемости всех уникальных дискретных значений всех признаков и условные вероятности для классов на их основе. Это может быть выполнено за один проход по данным. Далее в циклах выполняется проход по всем атрибутам и всем уникальным значениям этих признаков, подсчет условных вероятностей для этих значений и проверка условия (3.1.1). Эти шаги могут выполняться в параллельном режиме. Значения, удовлетворяющие условию (3.1.1), добавляются в подмножества-агрегаты, для которых в конце каждой итерации внешнего цикла формируются предикаты и правила класса вида (3.1.2) на их основе. В результате выполнения циклов формируется предварительный список правил. Далее для каждого правила из предварительного списка рассчитывается метрика оценки силы правила и выполняется их фильтрация на основании значения метрики. Для этого не требуется дополнительный проход по данным, так как можно использовать статистики, рассчитанные на первом шаге алгоритма. Результатом работы алгоритма будет являться множество ассоциативных правил класса, отражающих наиболее сильные связи параметров описаний СлОП с его состоянием.

3.2.3 Алгоритм АОССОП

Алгоритм предназначен для автономного оценивания текущего состояния сложных объектов и процессов, включающего наличие или отсутствие в текущий момент определенного вида функциональных неисправностей, дефектов и атакующих воздействий, свойственных целевой системе. Алгоритм строится на основе глубокой нейронной сети (ГНС) прямого распространения сигнала, обучение которой производится на объединенном наборе данных, полученном из открытых источников и включающем данные о функционировании нескольких видов индустриальных систем.

Блок-схема алгоритма представлена на рис. 3.2.3.

Начало алгоритма состоит в чтении входных данных: последовательность векторов характеристик СлОП (X). При необходимости обучения новой модели оценивания выполняется разбиение исходного набора данных в заданном пользователем соотношении, одна из образовавшихся выборок используется в качестве обучающей (X_{ij} – признаки, Y_{ij} – метки классов) для настройки весовых коэффициентов ГНС, другая часть – для тестирования. При использовании существующей модели входные данные используются для оценивания текущего состояния.

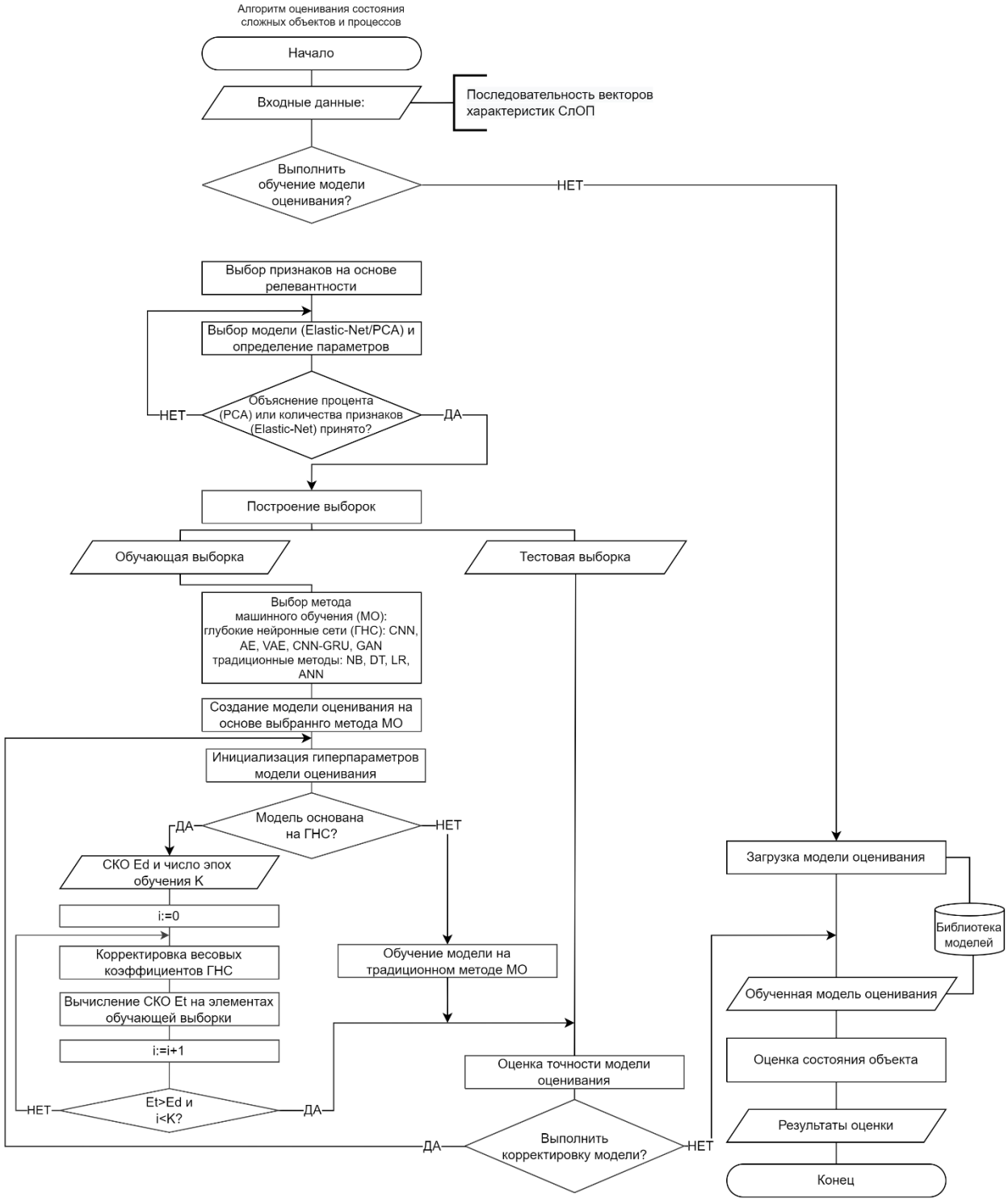

Рисунок 3.2.3 – Блок-схема алгоритма АОССОП

Далее осуществляется выбор признаков на основе их релевантности, а также методов главных компонент (Principal Component Analysis, PCA) или эластичной сети Elastic-Net. Отбор на основе PCA определяет наиболее важные признаки, объясняющие максимальную

дисперсию данных. Эластичная сеть сочетает регуляризацию L1 и L2 для преодоления ограничений каждого метода и может помочь минимизировать влияние нерелевантных функций в данных. Перечисленные методы показали свою эффективность при решении аналогичных задач, могут уменьшить размерность данных и обеспечить интерпретируемость процесса выбора.

Далее выполняется разбиение исходного набора данных в заданном пользователем соотношении, одна из образовавшихся подвыборок используется в качестве обучающей (X_{ij} – признаки, Y_{ij} – метки классов) для настройки весовых коэффициентов НС, другая часть – для тестирования.

После осуществляется выбор метода машинного обучения (МО) для оценивания состояний и инициализация гиперпараметров выбранной модели. Для настройки модели ГНС определяется величина среднеквадратичной ошибки (СКО, *MSE*) и числа эпох обучения НС. В процессе обучения выполняется корректировка весовых коэффициентов нейронной сети до тех пор, пока не будет достигнута требуемая величина СКО или число эпох обучения не превысит заданного значения. После этого вычисляется показатель точности (acc) НС на элементах тестовой выборки (Z_{ij}), на основании этого значения принимается решение о повторном обучении нейросетевой модели или ее применении для оценки состояния объекта. Обученная модель сохраняется в библиотеку моделей оценивания, и может в дальнейшем использоваться для решения задач оценивания. Результат выполнения алгоритма – сведения о текущем состоянии СлОП.

После обучения НС выполняется считывание атрибутов контролируемого сложного объекта, что подразумевает мониторинг параметров его состояния в дискретные моменты времени. На основе этих параметров формируется входной сигнал для НС. Далее выполняется построение выходного сигнала НС, с этой целью последовательно применяются композиции линейных и нелинейных преобразований над входным сигналом. Выходное значение НС рассматривается как признак наличия или отсутствия аномального состояния у контролируемого объекта.

3.2.4 Алгоритм АПССОП

Алгоритм предназначен для автономного прогнозирования состояний сложных объектов и процессов, включающего наличие или отсутствие в некоторый момент времени в будущем определенного вида функциональных неисправностей, дефектов и влияния как непреднамеренных воздействий, так и преднамеренных (атакующих) воздействий, свойственных целевой системе.

Алгоритм строится на основе рекуррентной глубокой нейронной сети, обучение которой производится на семействе наборов данных, полученных из открытых источников и включающих данные о функционировании нескольких видов индустриальных систем.

Алгоритм АПССОП реализует функции сильного ИИ в части выполнения автономного прогнозирования состояний сложных объектов и процессов. Наличие большого числа гиперпараметров и настраиваемых весов, свойственных для глубоких нейронных сетей, позволяет с достаточно высоким качеством выявлять закономерности между признаками прогнозируемого состояния системы и ожидаемой меткой его класса.

Блок-схема алгоритма представлена на рис. 3.2.4.

Начало реализации алгоритма автономного прогнозирования состояний сложных объектов и процессов состоит в чтении следующих входных данных: упорядоченную во времени последовательность векторов характеристик СлОП (X) и параметра длины исторической последовательности данных для прогнозирования (L).

Далее производится формирование генератора временных рядов (3.1.6). При необходимости обучить модель прогнозирования на входных данных осуществляется построение выборок – обучающей ($\boldsymbol{X}r$) и тестовой ($\boldsymbol{X}e$) – таким образом, что обучающая выборка содержит временной ряд данных, предшествующий временному ряду в тестовой выборке. Разбиение исходного набора выполняется в соответствии с заданным пользователем соотношением.

После осуществляется создание конфигурации рекуррентной нейронной сети, если она была передана в качестве входных данных, или загрузка готового пресета (фиксированной конфигурации сеть).

Гиперпараметры конфигурации включают в себя:

— тип блоков рекуррентной нейронной сети: SimpleRNN (3.1.7), LSTM (3.1.8), GRU (3.1.9);

— количество слоев нейронной сети (ограничено минимальным и максимальным значениями);

— количество узлов на каждом слое нейронной сети (ограничено минимальным и максимальным значениями):

— процент исключения случайных нейронов («дропаут»);

— функции активации на скрытом и выходном слоях;

— функция потерь.

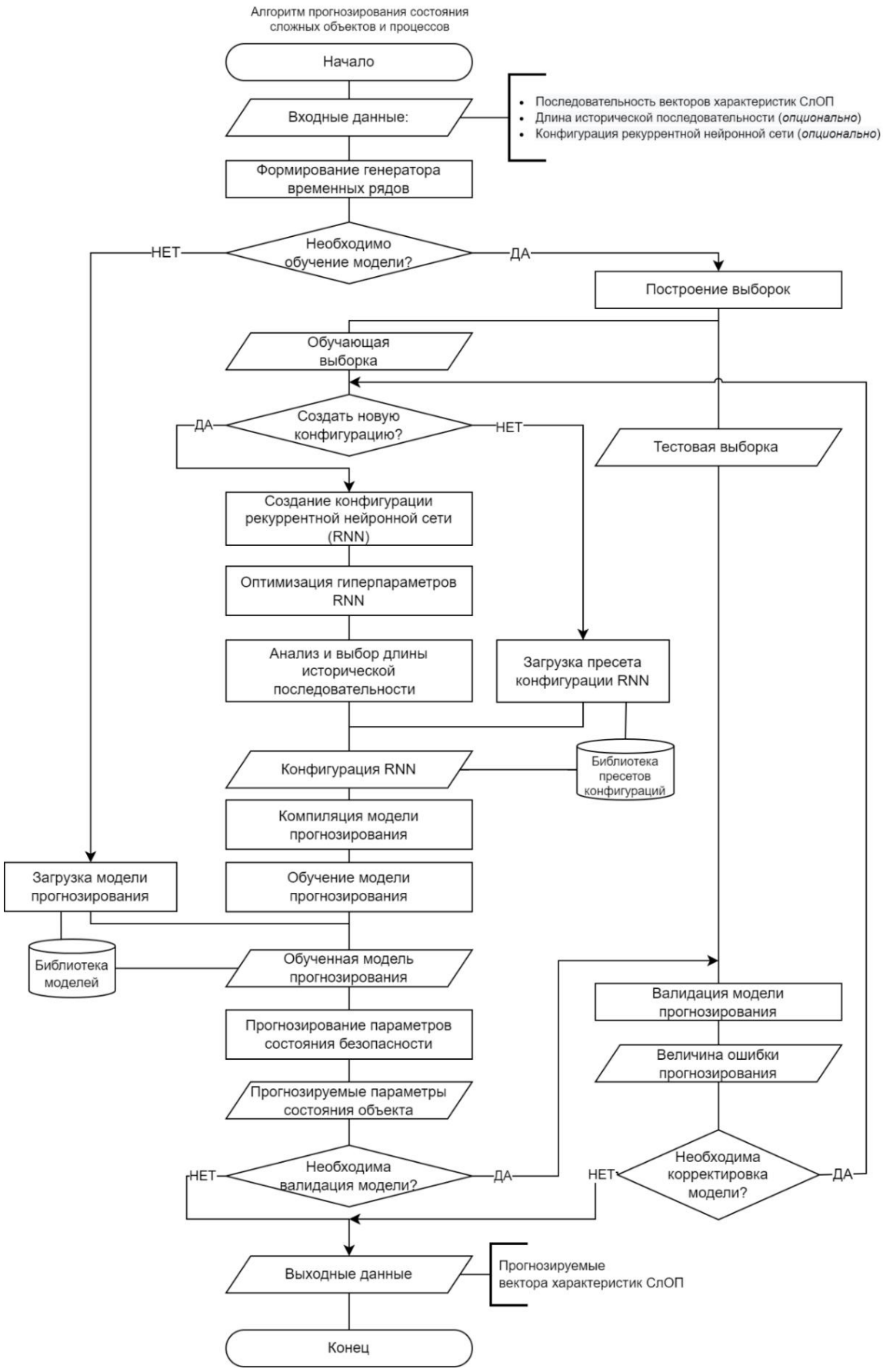

Рисунок 3.2.4 – Блок-схема алгоритма АПССОП

Модель прогнозирования обучается путем поиска закономерностей в векторах характеристик состояний СлОП для заданного временного окна и прогнозированием вектора значений в последующий момент времени.

Обученная модель прогнозирования LSTM применяется затем для прогнозирования параметров состояния. Далее, при необходимости, выполняется валидация модели прогнозирования на тестовых данных, результатом которой является величина ошибки прогнозирования, представленная кортежем <*mse, mae*>, где *mse* – среднеквадратичная ошибка, *mae* – средняя абсолютная ошибка. После этого принимается решение о необходимости корректировки модели прогнозирования, и при положительном решении заново происходит настройка параметров модели прогнозирования. Результатом выполнения данного алгоритма является набор прогнозируемых характеристик состояний объекта.

3.3 Структура компонента

Компонент состоит из модулей, каждый из которых реализует описанные ранее алгоритмы, а именно ПОСНД, ИЗСНД, АОССОП И АПССОП (см. раздел 3.2).

1) Модуль ПОСНД состоит из следующих программных элементов (реализованных с помощью программных классов), обладающих собственным целевым предназначением (см. рис. 3.3.1):

— *CheckDataTypes* – класс для коррекции типов данных;

— *ClusterFilling* – класс устранения неполноты данных;

— *Informativity* – класс для анализа информативности признаков;

— *MultiCollinear* – класс для устранения мультиколлинеарности;

— *Data* – класс для хранения данных и результатов их обработки;

— *PrintLog* – класс для журналирования работы модуля и представления результатов пользователю.

2) Модуль ИЗСНД состоит из следующих программных элементов (реализованных с помощью программных классов), обладающих собственным целевым предназначением (см. рис. 3.3.2):

— *Predicate* – класс для представления логических предикатов;

— *IzdapAlgo* – класс, реализующий логику алгоритма ИЗДАП.

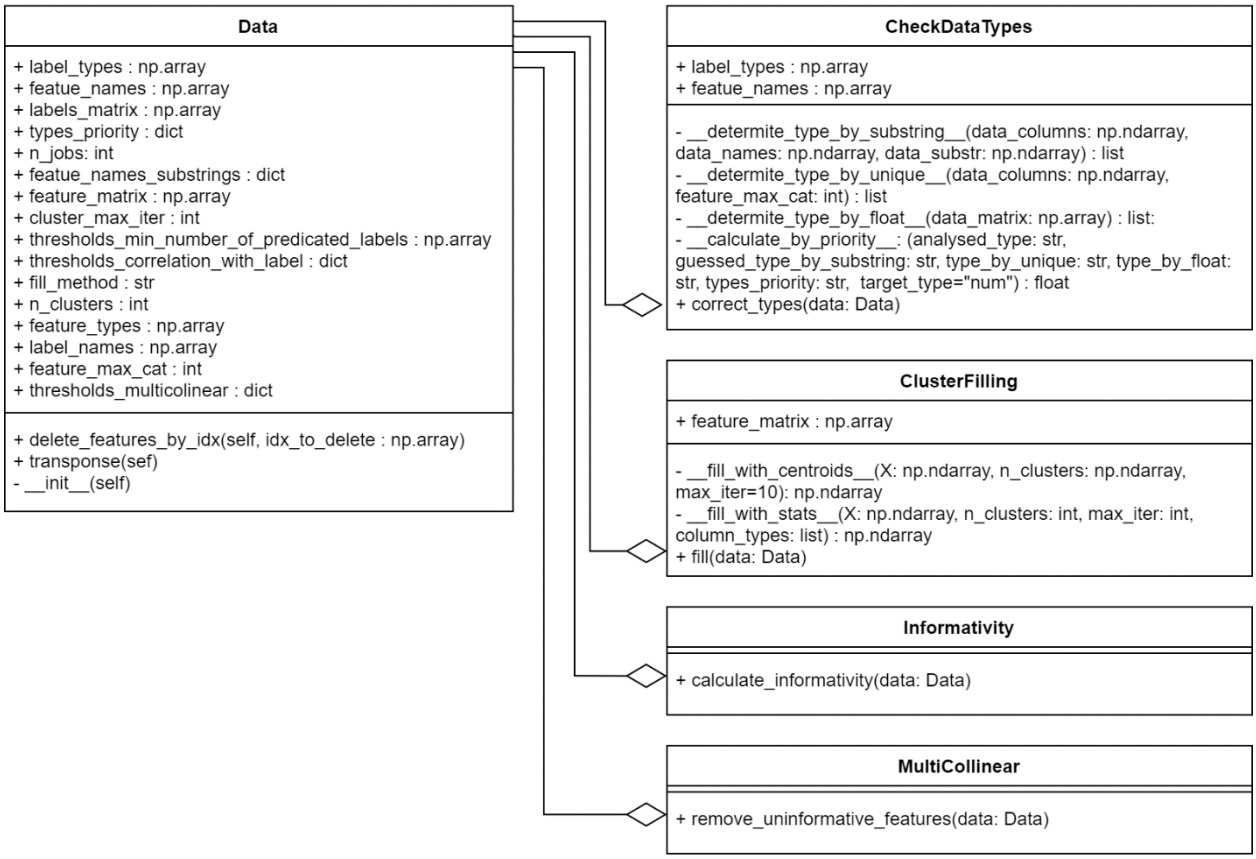

Рисунок 3.3.1 – Диаграмма классов модуля ПОСНД

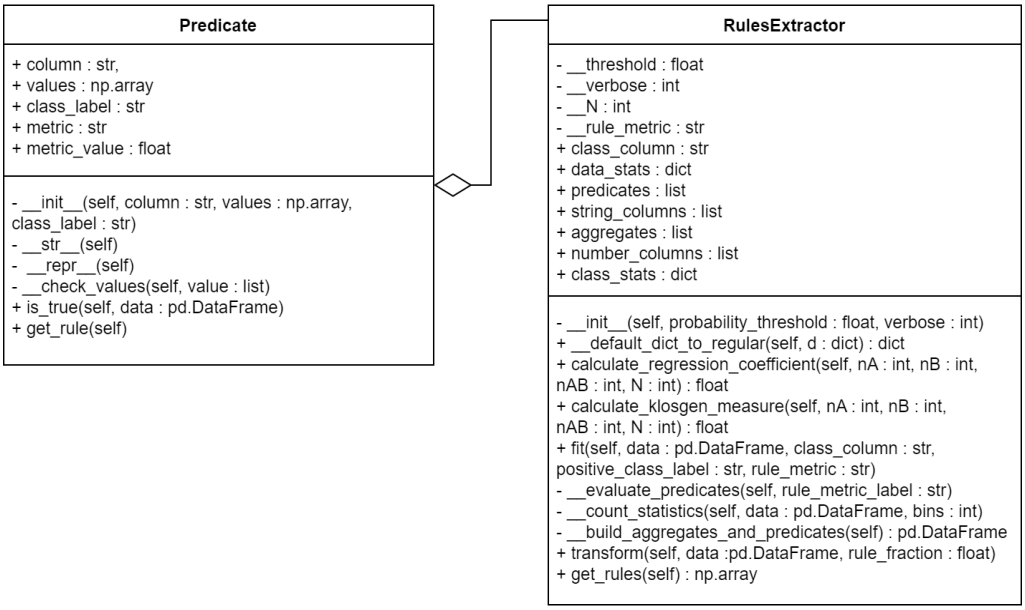

Рисунок 3.3.2 – Диаграмма классов модуля ИЗСНД

3) Модуль АОССОП состоит из следующих программных элементов (реализованных с помощью программных классов), обладающих собственным целевым предназначением (см. рис. 3.3.3 и 3.3.4).

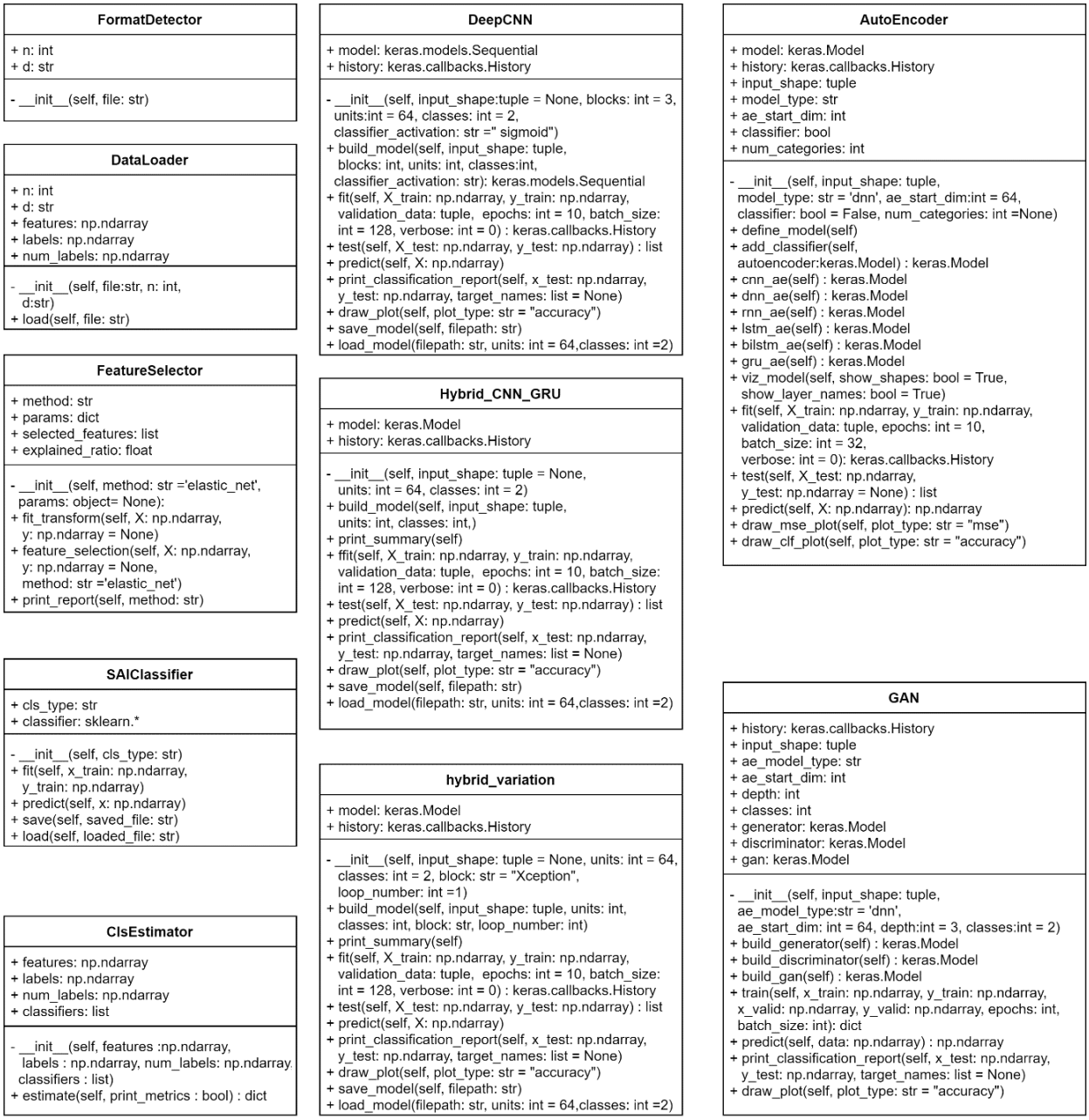

Рисунок 3.3.3 – Диаграмма классов модуля АОССОП

— *SAIClassifier* – элемент для селективной классификации, позволяющий выбрать один из следующих их типов: дерево решений, наивный Байес, логистическая регрессия и искусственная нейронная сеть. Атрибуты класса: *cls_type* – тип классификатора, *classifier* – модель классификации. Методы класса: (1) *fit* – обучение модели; (2) *predict* – прогнозирование класса состояния; (3) *save* – сохранение модели во внешний файл; (4) *load* – загрузка модели из внешнего файла; (5)*__init__* – конструктор класса.

— *FormatDetector* – элемент для определения формата входных данных. Атрибуты класса*: n* – число полей данных, *d* – символ разделителя полей в данных. Метод класса: *__init__* – конструктор класса.

— *DataLoader* – элемент для загрузки набора входных данных. Атрибуты класса: *n* – число полей данных, *d* – символ разделителя полей в данных, *features* – признаки данных, *labels* – метки данных, *num_labels* – количество уникальных меток. Методы класса: (1) *load* – загрузка данных, (2)*__init__* – конструктор класса.

— *ClsEstimator* – элемент для оценивания эффективности классификаторов. Атрибуты класса*: features* – признаки данных, *labels* – метки данных, *num_labels* – количество уникальных меток, *classifiers* – список моделей классификации для оценки. Методы класса: (1) *estimate* – оценка качества классификаторов; (2)*__init__* – конструктор класса.

— *FeatureSelector* – класс для выбора значимых признаков данных. Атрибуты класса: *method* – метод выбора признака («elastic_net» или «pca»), *params* – дополнительные параметры для выбранного метода, *selected_features* – список индексов выбранных признаков, *explained_ratio* – общее объясненное соотношение для выбранных функций (для PCA). Методы класса: (1) *fit_transform* – подобрать метод выбора признаков к входным данным и преобразовать их; (2) *feature_selection* – выполнить выбор признаков на основе указанного метода; (3) *print_report* – вывод отчета на основе выбранного метода выбора объекта.; (4)*__init__* – конструктор класса.

— *DeepCNN* – класс модели оценивания состояний, использующий в качестве классификатора сверточную нейронную сеть. Атрибуты класса: *model* – объект keras.Sequential, представляющий собой классификатор, *history* – информация о процессе обучения модели. Методы класса: (1) *build_model* – построение модели классификации; (2) *fit* – обучение модели; (3) *test* – оценка качества модели на тестовых данных; (4) *predict* – прогнозирование меток состояний объектов; (5) *print_classification_report* – отображение показателей качества классификации; (6) *draw_plot* – отображение графика процесса обучения с точностью или с потерями; (7) *save_model* – сохранение модели во внешний файл; (8) *load_model* – загрузка модели из внешнего файла; (9)*__init__* – конструктор класса.

— *Hybrid_CNN_GRU* – класс модели оценивания состояний, использующий в качестве классификатора гибридную модель, объединяющую слои CNN и GRU. Атрибуты класса: *model* – объект *keras.Model*, представляющий собой классификатор, *history* – информация о процессе обучения модели. Методы класса: (1) *build_model* – построение модели классификации; (2) *print_summary* – отображение сводной информации об

архитектуре модели; (3) *fit* – обучение модели; (4) *test* – оценка качества модели на тестовых данных; (5) *predict* – прогнозирование меток состояний объектов; (6) *print_classification_report* – отображение показателей качества классификации; (7) *draw_plot* – отображение графика процесса обучения с точностью или с потерями; (8) *save_model* – сохранение модели во внешний файл; (9) *load_model* – загрузка модели из внешнего файла; (10)*__init__* – конструктор класса.

— *hybrid_variation* – класс модели оценивания состояний, использующий в качестве классификатора гибридную модель вариаций сверточных слоев и GRU. Атрибуты класса: *model* – объект keras.Model, представляющий собой классификатор, *history* – информация о процессе обучения модели. Методы класса: (1) *build_model* – построение модели классификации; (2) *print_summary* – отображение сводной информации об архитектуре модели; (3) *fit* – обучение модели; (4) *test* – оценка качества модели на тестовых данных; (5) *predict* – прогнозирование меток состояний объектов; (6) *print_classification_report* – отображение показателей качества классификации; (7) *draw_plot* – отображение графика процесса обучения с точностью или с потерями; (8) *save_model* – сохранение модели во внешний файл; (9) *load_model* – загрузка модели из внешнего файла; (10)*__init__* – конструктор класса.

— *AutoEncoder* – класс модели оценивания состояний, использующий в качестве классификатора автокодировщик, в котором в качестве кодера и декодера могут выступать различные глубокие нейронные сети. Атрибуты класса: *model* – объект keras.Model, представляющий собой классификатор, *history* – информация о процессе обучения модели, *input_shape* – размер входных данных, *model_type* – тип глубокой нейронной сети, использующейся в качестве кодера и декодера, *ae_start_dim* – начальный размер автокодировщика, *classifier* – необходимо ли включить слой классификации, *num_categories* – количество уникальных меток. Методы класса: (1) *define_model* – определить структуру модели автокодировщика на основе указанного типа модели; (2) *add_classifier* – добавить слой классификации; (3) *cnn_ae* – установить CNN в качестве кодера и декодера; (4) *dnn_ae* – установить DNN в качестве кодера и декодера; (5) *rnn_ae* – установить RNN в качестве кодера и декодера; (6) *lstm_ae* – установить LSTM в качестве кодера и декодера; (7) *bilstm_ae* – установить Bi-LSTM в качестве кодера и декодера; (8) *gru_ae* – установить GRU в качестве кодера и декодера; (9) *viz_model* – визуализация архитектуры модели и сохранение в файл; (10) *fit* – обучение модели; (11) *test* – оценка качества модели на тестовых данных; (12) *predict* – прогнозирование меток состояний объектов; (13) *draw_mse_plot* – отображение графика процесса обучения с потерями; (14)

draw_clf_plot – отображение графика процесса обучения с точностью классификации; (15) *__init__* – конструктор класса.

GAN – класс модели оценивания состояний, использующий в качестве классификатора генеративно-состязательную сеть (GAN), основанную на автокодировщиике. Атрибуты класса: *gan* – объект keras.Model, представляющий собой классификатор, *generator* – объект keras.Model, представляющий собой генератор GAN, *discriminator*– объект keras.Model, представляющий собой дискриминатор GAN, *history* – информация о процессе обучения модели, *input_shape* – размер входных данных, *ae_model_type* – тип глубокой нейронной сети, использующейся в качестве кодера и декодера, *ae_start_dim* – начальный размер автокодировщика, *depth* – глубина для светочной нейронной сети, *classifier* – необходимо ли включить слой классификации, *classes* – количество уникальных меток. Методы класса: (1) *build_generator* – создать генератор с помощью указанной модели автокодировщика; (2) *build_discriminator* – создать дискриминатор для классификации; (3) *build_gan* – построить модель GAN, объединив генератор и дискриминатор; (4) *train* – обучение модели; (5) *predict* – прогнозирование меток состояний объектов; (6) *print_classification_report* – отображение показателей качества классификации; (7) *draw_plot* – отображение графика процесса обучения с точностью или с потерями; (8) *__init__* – конструктор класса.

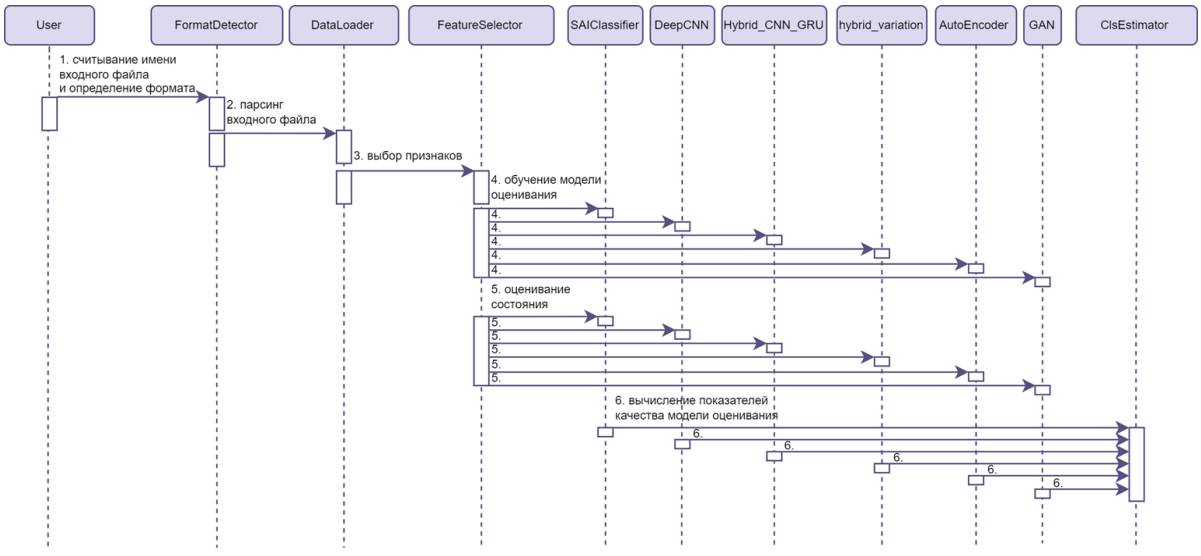

Рисунок 3.3.4 – Диаграмма последовательностей для алгоритма оценивания

Пользователь осуществляет ввод наименования обрабатываемого файла, в котором содержатся записи, подлежащие классификации (оценке состояния сложного объекта). Объект класса *FormatDetector* в свою очередь определяет формата входного файла, что

подразумевает вычисление числа полей и нахождение типа разделителя в csv-файле. Полученные сведения передаются объекту класса *DataLoader* для парсинга входного файла (выделения признаков и построения меток классов). Объект класса *FeatureSelector* используется для выбора значимых признаков данных. Обучение модели оценивания и непосредственно оценивание СЛоП осуществляется при помощи вызова объектов классов *SAIClassifier*, *DeepCNN*, *Hybrid_CNN_GRUm hybrid_variation*, *AutoEncoder* и *GAN*. Наконец, объект класса *ClsEstimator* позволяет вычислить показатели модели оценивания как на обучающей, так и на тестовой выборках.

4) Модуль АПССОП состоит из следующих программных элементов (реализованных с помощью программных классов), обладающих собственным целевым предназначением (см. рис. 3.3.5 и 3.3.6).

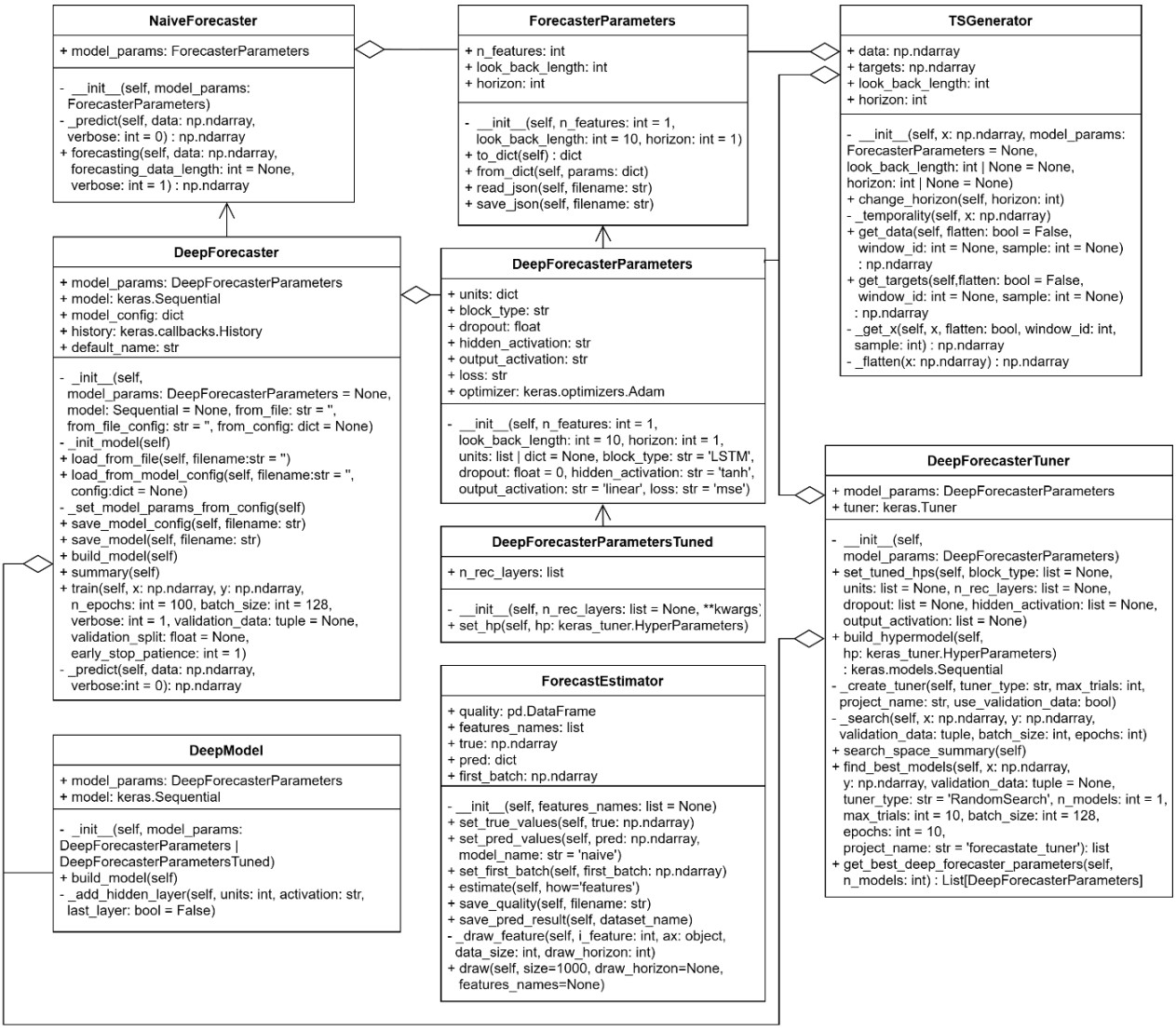

Рисунок 3.3.5 – Диаграмма классов модуля АПССОП

— *ForecasterParameters* – класс основных параметров модели прогнозирования. Атрибуты класса: *n_features* – количество признаков, *look_back_length* – число временных шагов до прогноза, *horizon* – горизонт прогнозирования во временных шагах. Методы класса: (1) *to_dict* – преобразование в словарь; (2) *from_dict* – преобразование из словаря; (3) *read_json* – загрузка параметров из JSON файла; (4) *save_json* – сохранение параметров в JSON файл; (5) *__init__* – конструктор класса.

— *DeepForecasterParameters* – класс параметров конфигурации глубокого модели прогнозирования. Наследует атрибуты и методы класса *ForecasterParameters.* Дополнительные атрибуты класса: *units* – число юнитов на слоях нейронной сети, *block_type* – тип блока RNN (SimpleRNN, LSTM, GRU), *dropout* – доля отбрасывания нейронов, *hidden_activation* – функция активации на скрытом слое, *output_activation* – функция активации на выходном слое, *optimizer* – функция оптимизации, *loss* – функция потерь.

— *DeepForecasterParametersTuned* – класс гиперпараметров для оптимизации конфигурации глубокой модели прогнозирования. Наследует атрибуты и методы класса *DeepForecasterParameters.* Дополнительный атрибут класса: *n_rec_layers* – число слоев модели. Дополнительный метод класса: *set_hp* – установить значения гиперпараметров.

— *TSGenerator* – класс генерации временных рядов. Атрибуты класса: *data* – входные данные временных рядов, *targets* – выходные данные временных рядов (целевые значения), *look_back_length* – число временных шагов до прогноза, *horizon* – горизонт прогнозирования во временных шагах. Методы класса: (1) *change_horizon* – изменить горизонт прогнозирования; (2) *_temporality* – преобразовать последовательность данных во временные окна; (3) *get_data* – вернуть входные данные временных рядов; (4) *get_targets* – вернуть выходные данные временных рядов; (5) *_get_x* – вернуть данные временных рядов; (6) *_flatten* – преобразовать временные окна обратно в последовательность; (7) *__init__* – конструктор класса.

— *NaiveForecaster* – класс базовой модели прогнозирования, использующей наивный метод. Атрибут класса: *model_params* – параметры модели прогнозирования. Методы класса: (1) *_predict* – функция прогнозирования наивной модели; (2) *forecasting* – прогнозирование заданной длины; (3) *__init__* – конструктор класса.

— *DeepModel* – класс генерации модели глубокой нейронной сети. Атрибуты класса: *model_params* – параметры модели, *model* – объект keras.models.Sequential. Методы класса: (1) *build_model* – формирование модели; (2) *_add_hidden_layer* – добавить новый скрытый слой в нейронную сети;(3) *__init__* – конструктор класса.

— *DeepForecaster* – класс глубокой модели прогнозирования. Наследует метод *forecasting* класса *NaiveForecaster*. Дополнительные атрибуты класса: *model_params* – параметры глубокой модели прогнозирования, *model* – объект keras.models.Sequential, представляющий собой нейронную сеть, *model_config* – конфигурация модели в формате библиотеки keras, *history* – информация о процессе обучения модели, *default_name* – имя модели. Дополнительные методы класса: (1) *_init_model* – инициализация атрибутов класса; (2) *load_from_file* – загрузить модель из файла; (3) *load_from_model_config* – определить модель через конфигурацию; (4) *_set_model_params_from_config* – установить параметры модели из атрибута конфигурации; (5) *save_model_config* – сохранить конфигурацию модели в файл; (6) *save_model* – сохранить модель в файл; (7) *build_model* – формирование модели; (8) *summary* – описание модели; (9) *train* – обучение модели; (10) *_predict* – прогнозирование.

— *DeepForecasterTuner* – класс оптимизации глубокой модели прогнозирования. Атрибуты класса: *model_params* – параметры глубокой модели прогнозирования, *tuner* – тюнер для оптимизации гиперпараметров. Методы класса: (1) *set_tuned_hps* – установить множество возможных значений гиперпараметров модели прогнозирования; (2) *build_hypermodel* – простроить гипермодель для оптимизации; (3) *_create_tuner* – создать тюнер по заданным настройкам; (4) *_search* – поиск лучшей(их) конфигурации(ий) модели прогнозирования; (5) *search_space_summary* – вывести сводку пространства поиска; (6) *find_best_models* – определить ряд лучших моделей прогнозирования по заданному множеству конфигураций гиперпараметров; (7) *get_best_deep_forecaster_parameters* – определить ряд лучших параметров моделей; (8) *__init__* – конструктор класса.

— *ForecastEstimator* – класс для оценки качества модели прогнозирования. Атрибут класса: *quality* – матрица результатов оценки качества, *features_names* – наименования признаков данных, *true* – истинные значения выходных данных временных рядов, *pred* – прогнозируемые значения выходных данных временных рядов, *first_batch* – первое входное временное окно. Методы класса: (1) *set_true_values* – установить истинные значения выходных данных временных рядов; (2) *set_pred_values* – установить прогнозируемые значения выходных данных временных рядов для заданной модели прогнозирования; (3) *set_first_batch* – установить значения первого входного временного окна; (4) *estimate* – оценить качество моделей прогнозирования; (5) *save_quality* – сохранить результаты оценки в файл; (6) *save_pred_result* – сохранить прогнозируемые значения выходных данных временных рядов; (7) *_draw_feature* – отобразить результаты

прогнозирования для одного признака данных; (8) *draw* – отобразить результаты прогнозирования для ряда признаков данных; (9) *__init__* – конструктор класса.

В сравнении с предыдущей версией библиотеки для программного модуля АПССОП осуществлены следующие изменения:

1) в класс *ForecasterParameters* добавлены методы *to_dict* и *from_dict;*

2) из класса *DeepForecasterParameters* исключены методы: (1) *n_rec_layers* – определить число слоев, (2) *save_json* – сохранение параметров в JSON файл, переопределение родительского метода; (3) *_units_to_dict* – преобразование списка юнитов в словарь;

3) в конструктор класса *DeepForecasterParameters* добавлен аргумент *optimizer*, позволяющий передавать оптимизатор нейронной сети;

4) добавлен класс *DeepForecasterParametersTuned,* характеризующий гиперпараметры для оптимизации конфигурации глубокой модели прогнозирования;

5) из класса *TSGenerator* исключен атрибут *model_params*;

6) в класс *TSGenerator* добавлены атрибуты *look_back_length* и *horizon*;

7) добавлен класс *DeepModel,* отвечающий за генерацию модели глубокой нейронной сети;

8) из класса *DeepForecaster* исключен метод *_add_hidden_layer*;

9) в класс *DeepForecaster* добавлен метод *summary*;

10) из класса *DeepForecasterTuner* исключены атрибут *hp_choices* и метод *_add_hidden_layer*;

11) в класс *DeepForecasterTuner* добавлены методы *search_space_summary* и *get_best_deep_forecaster_parameters.*

На рис. 3.3.6 приведена диаграмма последовательностей для компонента, реализующего алгоритм прогнозирования состояний сложных объектов и процессов.

Пользователь (*User*) определяет фиксированные параметры модели прогнозирования в *DeepForecasterParameters*, множество возможных значений гиперпараметров для оптимизации, а также входные данные параметров состояния в виде обучающей и тестовой выборок. Объект класса *DeepForecasterTuner* осуществляет оптимизацию гиперпараметров, которая заключается в поиске лучшей конфигурации глубокой модели прогнозирования на заданном множестве. Для установки параметров модели для оптимизации используется объект *DeepForecasterParametersTuned*, который определяется из класса *DeepForecasterTuner*. Для создания модели глубокой нейронной сети создается объект класса *DeepModel*. Объект класса *DeepForecaster* осуществляет

обучение модели прогнозирования, а также загрузку обученной модели и прогнозирование параметров состояния на основе входных данных. Качество прогнозирования рекуррентной модели оценивается объектом класса *ForecastEstimator*, которая вычисляет разницу между входными и предсказанными значениями.

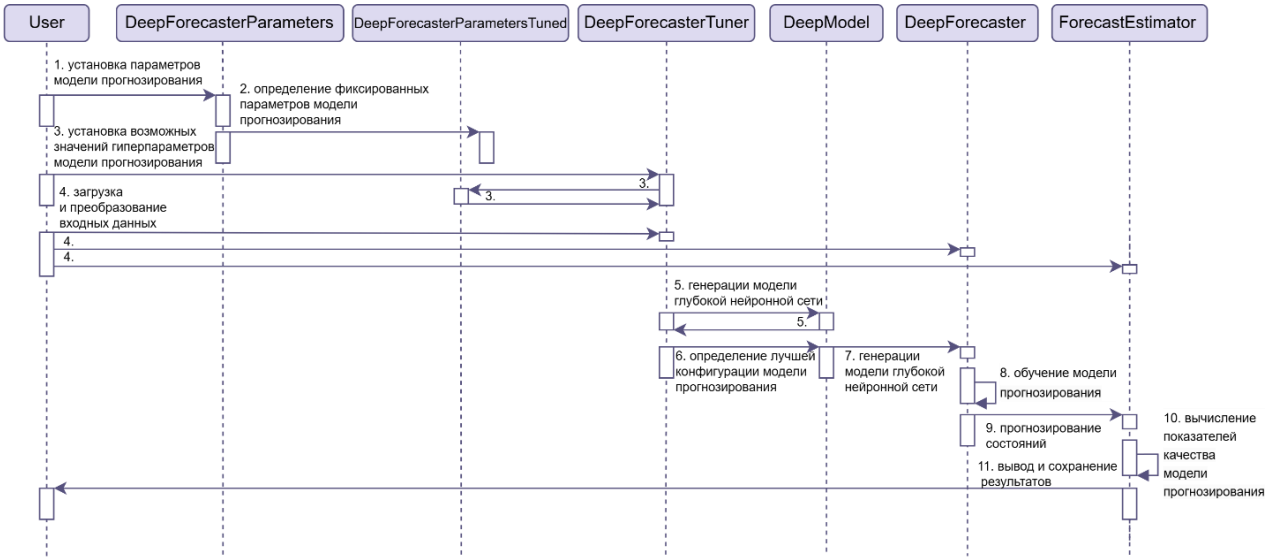

Рисунок 3.3.6 – Диаграмма последовательностей для алгоритма прогнозирования

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Техническими средствами являются электронно-вычислительные машины и устройства, которые используются при работе программы, должны иметь минимально необходимые характеристики, представленные в табл. 4.1. В качестве возможных операционных систем подходят любые из семейства Linux и Windows, совместимые с Python версии 3.11 и выше.

Таблица 4.1 – 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

5. ВЫЗОВ И ЗАГРУЗКА

5.1 Базовые функции компонента

1) Модуль ПОСНД

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)
```

2) Модуль ИЗСНД

Создание объекта *IzdapAlgo*:

```
# 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)
```

3) Модуль АОССОП

Модель инициализируется путем создания объекта *SAIClassifier*, которому передается тип классификатора, количество нейронов на распределительном 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 *ClsEstimator* создается путем вызова соответствующего конструктора с набором данных (векторы признаков, метки объектов) и типом классификатора:

```
classifier_types = ['decision_tree', 'naive_bayes', 'logistic_regression']
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 кодера и декодера (*ae_model_type*), начальная размерность (*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)
```

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):*

```
# Установка значений параметров
# n_features: number of features
# look_back_length: the width (number of time steps) of the input time windows
# horizon: output time window length
model_params = ForecasterParameters(n_features=4, look_back_length 100, horizon=10)
 # Изменения значений параметров
model_params.look_back_length = 200
# Загрузка параметров из внешнего файла
model_params.read_json('params.json')
# Загрузка параметров из словаря
model_params.from_dict({look_back_length=150, horizon=15})
```

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*) и функция оптимизации (*optimizer*)*:*

```
# Установка значений параметров
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')
 # Изменения значений параметров
model_params.block_type = 'GRU'
# Загрузка параметров из внешнего файла
model_params.read_json('params.json')
# Загрузка параметров из словаря
model_params.from_dict({block_type = 'GRU', dropout = 0.01})
```

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)
# или
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_params = ForecasterParameters(4, 100, 10)
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, 128])
model = DeepForecaster(model_params)
# Передача существующей модели
model = DeepForecaster(model=my_model)
# Загрузка модели из файла keras
model = DeepForecaster(from_file='my_model.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 = 1000)
```

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
```

6. ВХОДНЫЕ ДАННЫЕ

6.1 Состав и структура входных данных

1) Входные данные модуля ПОСНД сведены в единую табл. 6.1.1.

Таблица – 6.1.1. Входные данные модуля ПОСНД

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

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 табл. 6.1.2.

Таблица 6.1.2 – 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. 6.1.3

Таблица 6.1.3 – 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. 6.1.4

Таблица 6.1.4 – Входные данные модуля АПССОП

Наименование данных	**Обозна-чение**	**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	Нет

			функции через оперативную память	
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

6.2 Подготовка входных данных

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

1) Модуль ПОСНД: 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. ВЫХОДНЫЕ ДАННЫЕ

7.1 Состав и структура выходных данных

1) Выходные данные модуля ПОСНД сведены в единую табл. 7.1.1.

Таблица 7.1.1 – Выходные данные модуля ПОСНД

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. 7.1.2.

Таблица 7.1.2 – Выходные данные модуля ИЗСНД

Наименование данных	**Обозначение**	**Структура данных**	**Способ вывода данных**	**Ограничения**
Множество построенных ассоциативных правил класса	$U_k^{(j)}\left(d_k^l \in A_k^{(j)}\right) \to Y_j$	Массив объектов класса Rules	Вывод в консоль	Нет
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. 7.1.3.

Таблица 7.1.3 – 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. 7.1.4.

Таблица 7.1.4 – 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 Интерпретация выходных данных

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

1) Модуль ПОСНД 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.

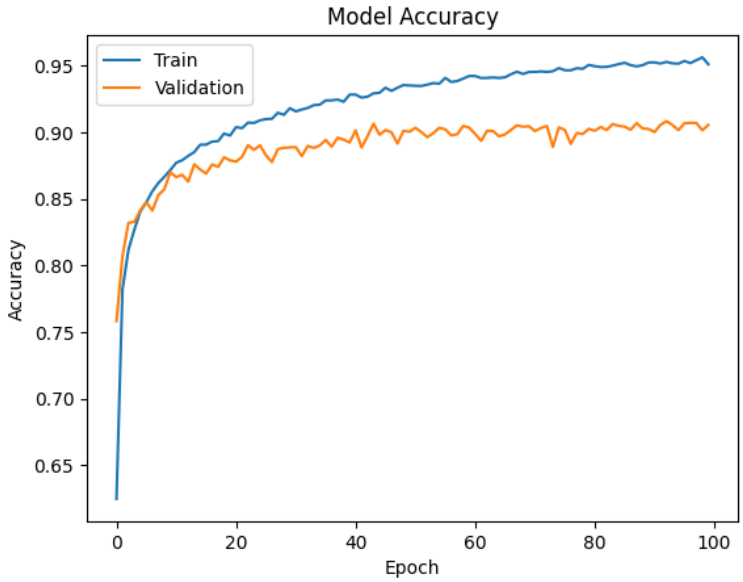

Рисунок 7.2.1 – Пример графика параметров обучения модели оценивания

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».

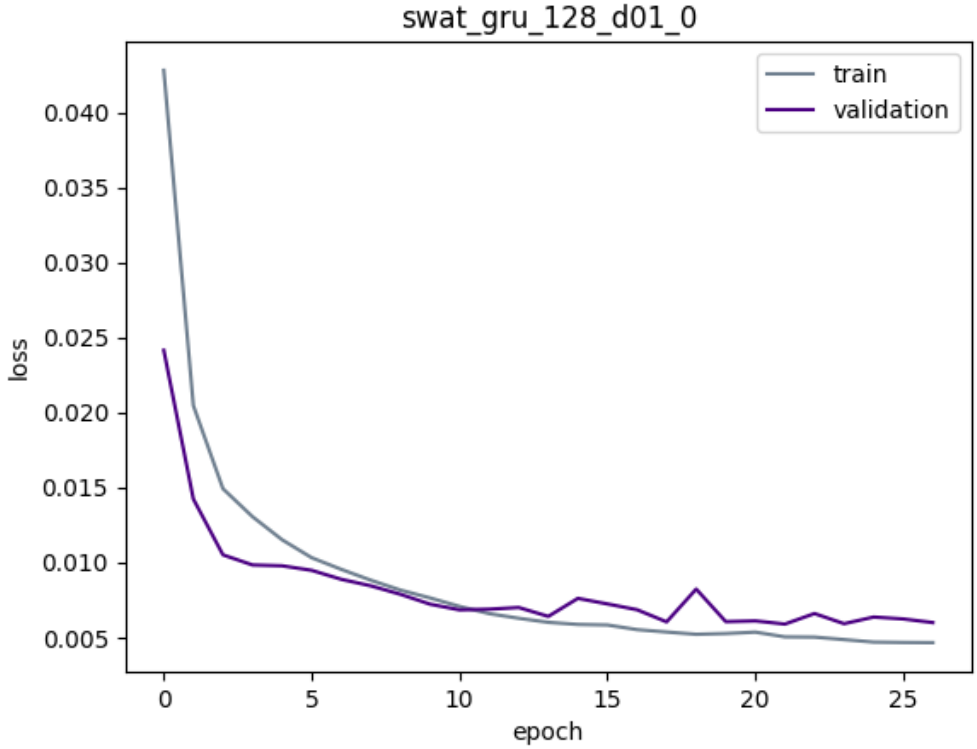

Рисунок 7.2.3 – Пример графика параметров обучения модели прогнозирования

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
P502	0.001445	0.038007	0.027568	0.000000	0.000007	0.002778
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.

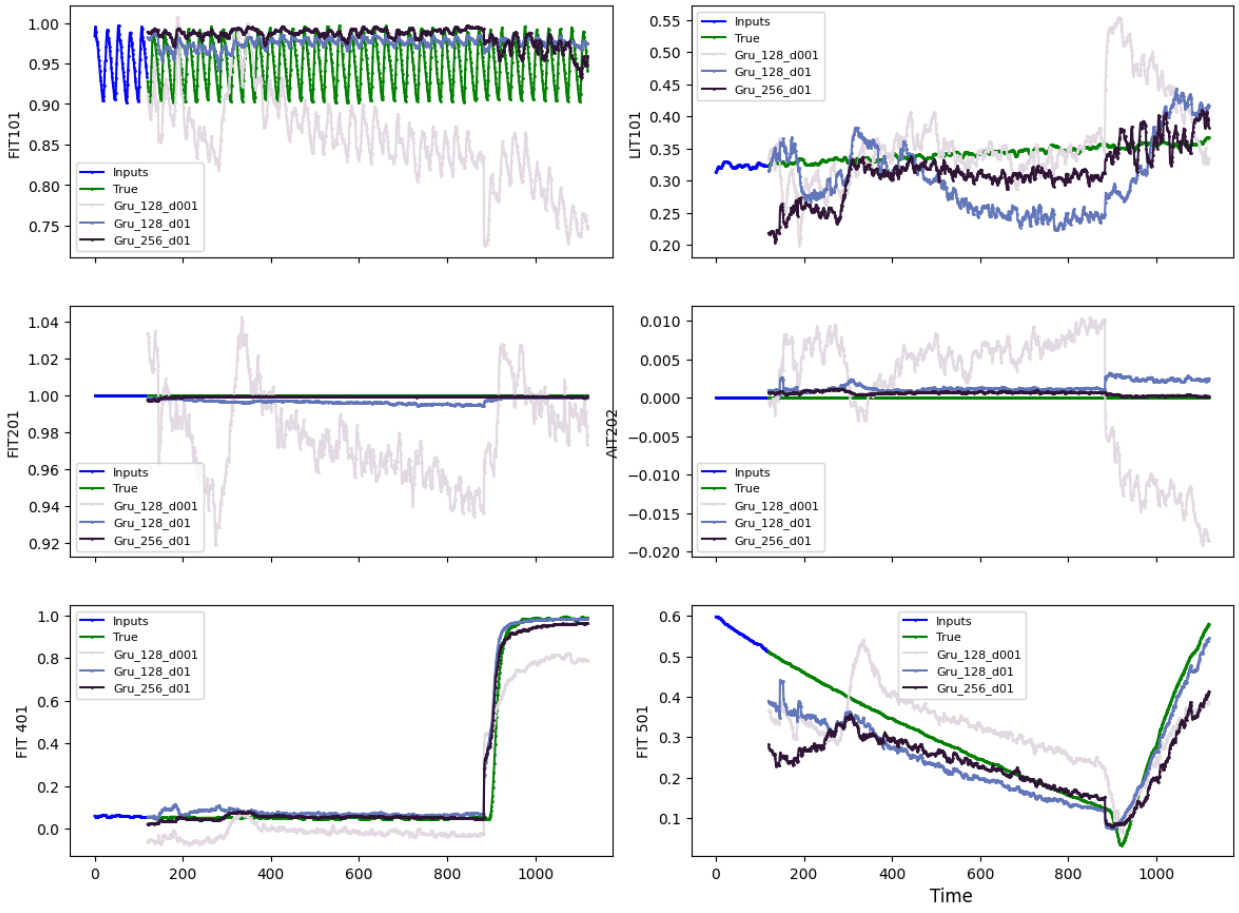

Рисунок 7.2.4 – Пример вывода таблицы показателей качества прогнозирования

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					