Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Высшая школа киберфизических систем и управления ИКНТ

Работа допущена к защите

Директор ВШ КФСУ

\_\_\_\_\_\_\_\_\_\_\_ В.П. Шкодырев

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**РАБОТА БАКАЛАВРА**

**ПРОГНОЗИРОВАНИЕ ТЕХНОЛОГИЧЕСКИХ ПАРАМЕТРОВ АВТОМАТИЗИРОВАННОГО ТЕПЛОВОГО ПУНКТА МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ**

направление 27.03.04 - Управление в технических системах

профиль 27.03.04\_05 - Интеллектуальные системы обработки информации и

управления

Выполнил

студент гр.3532704/80501 Е.А. Федотовских

Руководитель

старший преподаватель Л.А. Киселёва

Консультант

по нормоконтролю Л.А. Киселёва

Санкт-Петербург

2022

РЕФЕРАТ

На 49 с., 22 рисунка, 4 приложения.

КЛЮЧЕВЫЕ СЛОВА: ПРЕДИКТИВНОЕ УПРАВЛЕНИЕ, ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ, LSTM, ИСКУСТВЕННАЯ НЕЙРОННАЯ СЕТЬ, ГРАДИЕНТНЫЙ БУСТИНГ НАД РЕШАЮЩИМИ ДЕРЕВЬЯМИ, СЛУЧАЙНЫЙ ЛЕС.

Тема выпускной квалификационной работы: «Прогнозирование технологических параметров автоматизированного теплового пункта методами машинного обучения».

Данная работа посвящена исследованию в области прогнозирования временных рядов для создания модуля предиктивного управления. Задачи, которые решались в ходе исследования:

1. Изучить работу автоматизированного теплового пункта;
2. Изучить методы машинного обучения для прогнозирования временных рядов;
3. Проанализировать предоставленные данные для обучения, предобработать их;
4. Реализовать программные модули для прогнозирования одномерных временных рядов, используя изученные методы машинного обучения;
5. Сравнить качество получившихся моделей машинного обучения для прогнозирования одномерных временных рядов, оптимизировать модели, выбрать оптимальную;
6. Реализовать программный модуль прогнозирования технологических параметров.

Работа выполнена на базе автоматизированного теплового пункта. Были проведены исследования в области прогнозирования временных рядов и применены различные методы машинного обучения. Все модели методы реализованы на языке Python 3. Выбрана модель с наилучшим качеством предсказаний – LSTM. Эта модель встроена в модуль предиктивного управления.

ABSCTRACT

49 pages, 22 figures, 4 appendices.

KEYWORDS: PREDICTIVE CONTROL, TIME SERIES FORECASTING, LSTM, ARTIFICIAL NEURAL NETWORK, GRADIENT BOOSTING OVER DECISION TREES, RANDOM FOREST.

Topic of the final qualification work: "Forecasting the technological parameters of an automated heat supply unit using machine learning methods."

This work is devoted to research in the field of time series forecasting to create a predictive control module. Tasks that were solved during the study:

1. Explore the work of an automated heat supply unit;
2. Study machine learning methods for time series forecasting;
3. Analyze the provided data for training, pre-process it;
4. Implement software modules for predicting one-dimensional time series using the studied methods of machine learning;
5. Compare the quality of the resulting machine learning models for predicting one-dimensional time series, optimize the models, choose the optimal one;
6. Implement a software module for predicting technological parameters.

The work was carried out of an automated heat supply unit. Research has been done in the field of time series forecasting and various machine learning methods have been applied. All model methods are implemented in Python 3. The model with the best quality of predictions, LSTM, is selected. This model is built into the predictive control module.

**Содержание**

[**Введение** 3](#_Toc104909104)

[**1.** **Общая постановка задачи** 6](#_Toc104909105)

[**1.1** **Описание функциональной схемы теплового пункта** 6](#_Toc104909106)

[**1.2** **Описание и предобработка измеряемых данных** 7](#_Toc104909107)

[1.2.1. Одномерный временной ряд 7](#_Toc104909108)

[**2.** **Методы прогнозирования временных рядов** 10](#_Toc104909109)

[**2.1 Случайный лес** 10](#_Toc104909110)

[2.1.1 Решающие деревья 10](#_Toc104909111)

[2.1.2 Бэггинг над решающими деревьями 14](#_Toc104909112)

[**2.2 Градиентный бустинг над решающими деревьями** 15](#_Toc104909113)

[**2.3** **Искусственные глубокие нейронные сети** 20](#_Toc104909114)

[2.3.1 Общие сведения 20](#_Toc104909115)

[2.3.2 Обучение нейронных сетей 22](#_Toc104909116)

[**2.4 Искусственные рекуррентные нейронные сети LSTM** 25](#_Toc104909117)

[2.4.1 Общие положения о рекуррентных нейронных сетях 25](#_Toc104909118)

[2.4.2. Сети LSTM 26](#_Toc104909119)

[**3.** **Сравнение методов прогнозирования** 29](#_Toc104909120)

[**3.1.** **Выбор метрики и оптимизация моделей** 29](#_Toc104909121)

[**3.2.** **Сравнение моделей** 30](#_Toc104909122)

[**Заключение** 34](#_Toc104909123)

[**Список литературы** 35](#_Toc104909124)

[**Приложение А.** 37](#_Toc104909125)

[**Модель случайного леса** 37](#_Toc104909126)

[**Приложение Б.** 40](#_Toc104909127)

[**Модель градиентного бустинга** 40](#_Toc104909128)

[**Приложение В.** 43](#_Toc104909129)

[**Модель искусственной нейронной сети** 43](#_Toc104909130)

[**Приложение Г.** 46](#_Toc104909131)

[**Модель LSTM** 46](#_Toc104909132)

# **Введение**

В XXI веке началось бурное развитие различного рода методов машин-ного обучения, в том числе, нейронных сетей. Это было обусловлено ростом вычислительных мощностей, а также увеличением объема накопленных человечеством в цифровом виде данных. За счет этого появилась возможность использования моделей машинного обучения без чрезмерных временных затрат для решения самых разнообразных задач. Также свою роль сыграла разработка новых типов нейронных сетей, более подходящих для решения некоторых специфических задач. Данные разработки стали применяться в сфере АСУ ТП. Сегодня с их помощью решаются такие задачи, как детекти-рование аномалий, предиктивная аналитика, прогнозирование временных рядов, техническая диагностика и др. В последнее время нейронные сети используются для управления объектами коммунального хозяйства такими как, например, тепловые пункты.

Тепловой пункт ‒ это важный узел в теплосистеме, обеспечивающий передачу тепловой энергии из центральной магистрали к потребителю. Тепловые пункты обслуживают многоквартирные дома (индивидуальный тепловой пункт) или целые микрорайоны, поселки и группы объектов (центральный тепловой пункт). Главная задача тепловых пунктов ‒ прием теплоэнергии из центральной сети и распределение тепла потребителям таким образом, чтобы каждый объект получал теплоноситель заданной температуры и под должным давлением. Современной задачей теплового пункта является регулирование температуры в здании в зависимости от условий окружающей среды для заблаговременного изменения расхода теплоносителя, чтобы ком-пенсировать инерцию здания. Более оптимальным решением является предик-тивное регулирование на основе прогноза температуры окружающей среды.

За несколько лет работы измерялись и сохранялись в базу данных значения внешней температуры в разные времена года. С помощью методов машинного обучения, используя эти данные, можно предсказывать внешнюю температуру за небольшой предыдущий период времени. Это поможет заранее подать сигнал на уменьшение или увеличение интенсивности отопления, и поддерживать нужный температурный режим в здании, т.к. тепловой пункт обладает большой инерционностью.

Для решения этой задачи потребуется рассмотреть следующие методы машинного обучения для предсказания временных рядов разных размерностей:

1. Случайный лес;
2. Градиентный бустинг над решающими деревьями;
3. Классические искусственные нейронные сети;
4. Искусственные рекуррентные нейронные сети Long-Short Term Memory (LSTM) - долгая краткосрочная память.

С помощью приведенных выше методов можно выполнить прогнозирование параметра по предыдущим значениям самого же себя (одномерный временной ряд).

*Целью данной работы* является создание программного модуля на основе моделей машинного обучения для возможности предиктивного управления интенсивностью отопления и детектирования аномалий на автоматизированном тепловом пункте. Это поможет более стабильно поддерживать уставку температуры внутри здания.

Для реализации цели необходимо выполнить следующие задачи:

1. Изучить работу автоматизированного теплового пункта;
2. Изучить методы машинного обучения для прогнозирования временных рядов;
3. Проанализировать предоставленные данные для обучения, предобработать их;
4. Реализовать программные модули для прогнозирования одномерных временных рядов, используя изученные методы машинного обучения;
5. Сравнить качество получившихся моделей машинного обучения для прогнозирования одномерных временных рядов, оптимизировать модели, выбрать оптимальную;
6. Реализовать программный модуль прогнозирования технологических параметров.

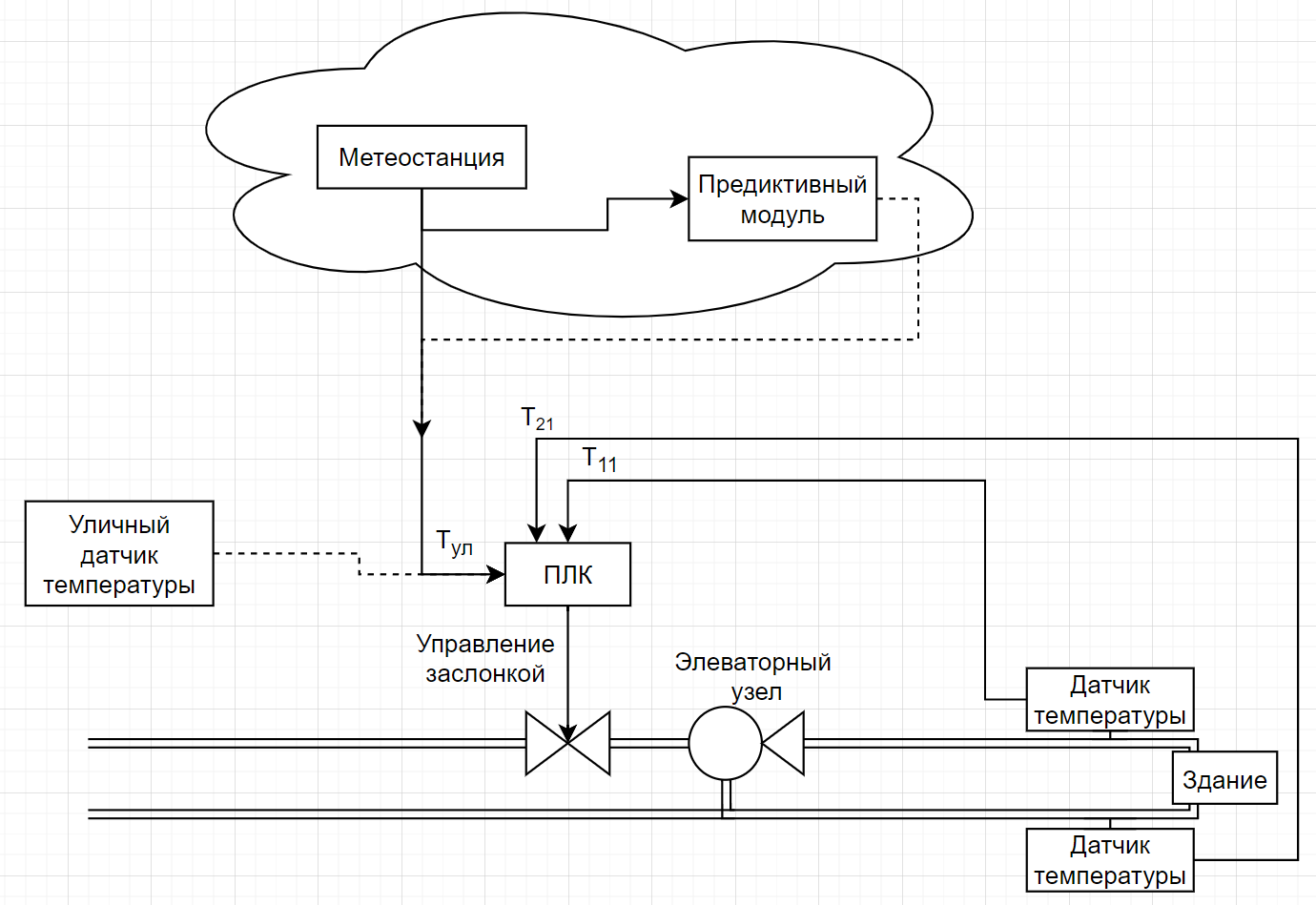
В России и странах СНГ практически не существует подобных систем, работающих в масштабах хотя бы нескольких десятков объектов.

Актуальность бакалаврской работы заключается в том, что можно выполнять предиктивное погодное регулирование не только для систем отопления зданий, но и в других сферах, таких как: агропромышленный комплекс, теплогенерирующие предприятия и т.д.

1. **Общая постановка задачи** 
   1. **Описание функциональной схемы теплового пункта**

В качестве объекта управления, задающие параметры которого требуется спрогнозировать для обеспечения его качественной работы, выступает автома-тизированный тепловой пункт, осуществляющий регулирование интенсив-ности отопления жилого дома или объекта инфраструктуры на основе измерения температуры наружного воздуха.

Функциональная схема теплового пункта представлена на рисунке 1.



𝑇ул − температура наружного воздуха; 𝑇11 − температура носителя к потребителю;

𝑇21 − температура носителя от потребителя; 𝑇21, 𝑇11 − внутренние параметры системы,

𝑇ул − внешние параметры системы

Рисунок 1 – Функциональная схема автоматизированного теплового пункта

Объект представляет собой отопительный контур, в который из город-ской теплосети поступает горячая вода. В элеваторном узле горячая вода смешивается с водой, прошедшей через здание, и отдает ему часть своего тепла, из-за чего охлаждается сама. Чтобы не подавать в систему отопления здания слишком горячую воду используется элеваторный узел. Управление этим узлом осуществляет заслонка, которая может находиться в разных положениях, тем самым контролируя подачу из городской теплосети горячей воды. Это позволяет поддерживать температуру внутри здания, что является основной задачей этого теплового пункта. Сигнал управления заслонкой формируется программой ПЛК, на который подаются данные с датчиков уличной температуры 𝑇ул или метеостанции и датчиков температуры носителя от- 𝑇21 и к потребителю 𝑇11. Степень открытия заслонки зависит от этих трех параметров: 𝑇21, 𝑇11, 𝑇ул. Датчики температуры для измерения первых двух параметров стоят на входе и выходе системы отопления здания. Датчик измерения уличной температуры висит на фасаде здания, но используется только в случае, если данные с метеостанции перестают поступать. В обычном режиме значение уличной температуры приходит с метеостанции, так как она проводит измерения точнее, данные о температуре заносятся в один из регистров ПЛК.

В ходе работы теплового пункта необходимо решить следующую задачу: предсказать наружную температуру на основе ее предыдущих значений, поступающих с метеостанции и записываемых в регистр ПЛК, тем самым будет осуществляться управление заслонкой в предиктивном режиме (англ. predictive ‒ предсказывающий). Это позволит подавать управляющий сигнал на заслонку заранее, на основе прогноза обученной модели, что при большой инерционности работы теплового пункта (от 30 мин) позволит прогревать /охлаждать здание в соответствии с внешними климатическими условиями (мороз, жара и т.д.).

Эта задача называется **прогнозированием одномерных** **временных рядов** и является задачей регрессии, то есть восстановления целевых значений непрерывных параметров при определенных значениях признаков.

* 1. **Описание и предобработка измеряемых данных**

### 1.2.1. Одномерный временной ряд

Типами входных данных при машинном обучении являются временные ряды или сигналы, которые содержат значения наружной температуры, измеренные каждые 4 минуты, всего около 21000 измерений, что составляет информацию за два месяца. Перед созданием и обучением моделей заполняются имеющиеся пропущенные значения, это делается методом интерполяции или взятием среднего значения правого и левого значений относительно пустого места. Также удаляются выбросы, т.к. их наличие в наборе данных приводит к некорректным результатам. Выбросы (outliers) – это данные, резко отличающиеся от основного числа данных. Будем считать выб-росом изменение параметра температуры более чем на 1 градус по модулю между соседними измерениями.

Если требуется знать значение температуры на 30 минут вперед (инерционность теплового пункта), то для этого необходимо иметь предсказа-ние на 7-8 элементов временного ряда вперед, что представляет собой гипер-параметр, названный *forward\_number*. В качестве элементов, на основе которых делаем предсказание, выступает гиперпараметр размер окна – *window\_size*.

Во всех случаях будем реализовывать методы машинного обучения с учителем, где в качестве признаков взяты *window\_size* элементов, а в качестве целевого значения - элемент, который находится через *forward\_number* элемен-тов от последнего элемента из признаков. Если необходимо предсказать следующее значение, то *forward\_number=0*.

Датасет (некоторое количество исходных данных) для прогнозирования одномерных временных рядов представлен на рисунке 2, где зеленым цветом обозначены элементы признаков, оранжевым – целевого значения. То есть каждому массиву из *window\_size* зеленых элементов соответствует одно оранжевое целевое значение. Создание датасетов происходит со сдвигом, пока не закончатся все значения временного ряда.

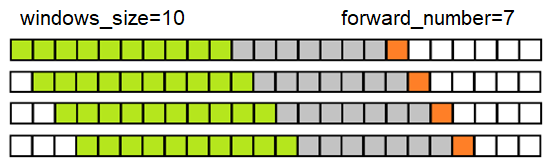


Рисунок 2 – Датасет для прогнозирования одномерных временных рядов

На основе этого правила и формируются датасеты для обучения, проверки качества модели и тестирования (разбиты в процентном отношении 70-20-10). На тренировочном наборе данных происходит обучение моделей, при этом во время обучения ошибка также считается и на валидационном датасете, который не принимает участие в изменении параметров моделей, а служит для избегания переобучения. Тестовый датасет используется для проверки качества модели на данных, которые никак не участвовали в обучении.



Рисунок 3 – Разделение на датасеты одномерного временного ряда

1. **Методы прогнозирования временных рядов**

Машинное обучение – подраздел компьютерных наук, в частности искус-ственного интеллекта, изучающий набор вычислительных и статистических методов, для построения моделей, улавливающих закономерности в данных. Задача машинного обучения — это тип прогноза или вывода, основанный на возникшей проблеме или на вопросе, а также доступных данных. Основные задачи машинного обучения – регрессия, классификация, кластеризация данных и др. Задача классификации назначает данные категориям, задача клас-теризации группирует данные в соответствии со сходством. Задача регрессии – это прогноз на основе выборки объектов с различными признаками. На выходе должно получиться вещественное число (5, 77, 78.454 и др.), например, значение температуры.

В рассматриваемом случае стоит задача регрессии ‒ восстановления целевых значений непрерывных параметров при определенных значениях признаков. Рассмотрим методы регрессии, которые были реализованы в бакалаврской работе для прогнозирования временных рядов.

## **2.1 Случайный лес**

### 2.1.1 Решающие деревья

Для задачи регрессии используются регрессионные решающие деревья, которые предсказывают значения целевого значения прогнозируемого пара-метра с помощью применения последовательности предикатов. Предикат – это простое решающее правило, а именно взятие порога по значению какого-либо признака.

где *xj* – значение признака; *t* – порог значения

В вершинах решающих деревьев находятся предикаты, в листьях – прог-нозы целевого параметра. Если предикат верен для текущего признака из набо-ра признаков, то происходит переход в правое поддерево потомок, если нет, то в левое поддерево потомок. При выполнении предсказания происходит проход по дереву сверху вниз до какого-то определенного листа. Этот проход выпол-няется с корневой вершины. Решающее дерево глубины 2, использующее два признака (с номерами 2 и 6) для определения искомого целевого параметра изображено на рисунке 4.

Каждый предикат порождает разделение текущего подмножества пространства признаков на две части, рисунок 5. Первый, он же корневой, предикат разделяет пространство на две части по признаку номер 2: все что больше -2.58, и все, что меньше.

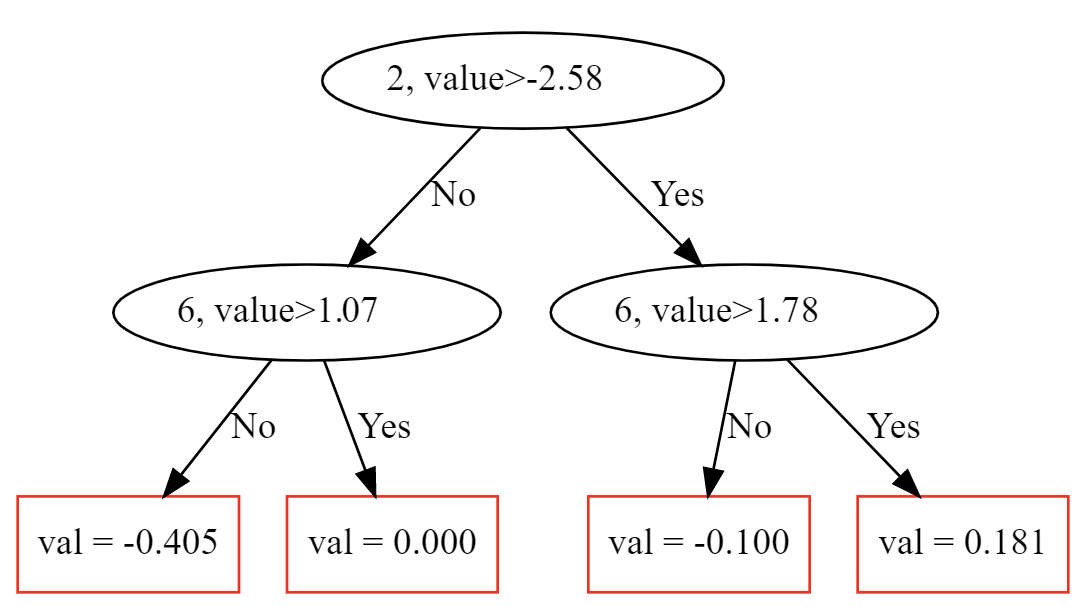


Рисунок 4 – Пример решающего дерева

Аналогичным образом предикаты на следующем уровне решающего дерева разделяют пространства на две части по признаку номер 6. В итоге получается несколько областей плоскости, для каждой из которых, определено значение целевого параметра, и любому объекту с определенными значениями признаков соответствует какая-то из этих областей.

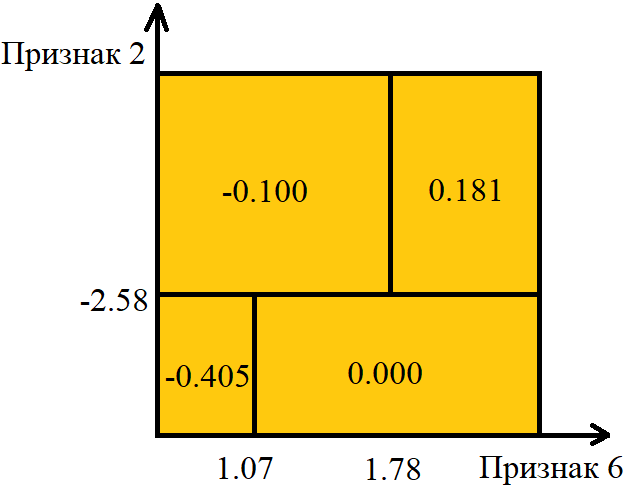


Рисунок 5 – Разделение пространства признаков предикатами решающего дерева

Алгоритм построения решающего дерева состоит в следующем:

1. Создание вершины. *Xm*– множество объектов, попавших в текущую вершину.
2. При выполнении критерия остановки *Stop*(*Xm*) вершина становится листом и ей в соответствие ставится какой-то ответ *Answer*(*Xm*).
3. Если критерий остановки не выполнен, находим предикат для вершины, который определяет наилучшее разбиение множества объектов *Xm*на два подмножества: *Xl*и *Xr*. При этом должен максимизироваться критерий ветвления *Branch*(*Xm, t*)*.*
4. Рекурсивное повторение для *Xl*и *Xr*.

Целью всех функций является построение итогового решающего дерева, которое бы минимизировало функцию потерь *L*(*y,с*),где *y* - столбец ответов (целевых значений) соответствующих матрице признаков *x*; *с* - ответы, кото-рые дает решающее дерево для *x*.

Функция *Answer*(*Xm*)вычисляет ответ для текущего листа в зависимости от попавших в него объектов выборки. Обычно берется среднее значение или медианное.

Функция *Stop*(*Xm*) определяет, необходимо ли продолжить построение дерева или остановиться. Чаще всего используются простые правила: достижение однородности объектов в листе или достижение определенного количества элементов в листе.

Функция *Branch*(*Xm, t*)измеряет насколько оптимально предикат разделяет текущую выборку *Xm*. Это называется критерий ветвления. Он дает оценку насколько произойдет улучшение финальной метрики качества решающего дерева для двух вариантов при разделении.

1. Если получившиеся листы будут конечными;
2. Если исходная вершина останется конечным листом, т.е. не произойдет разделения.

Пусть задана функция потерь *L*(*yi,c*), где *c* – ответ, который дает решающее дерево для текущего набора *Xm* перед разделением предикатом выборки на *Xl* и *Xr* . Ответ *c* должен минимизировать усредненное значение функции потерь.– количество элементов в выборке

Эта величина называется информативностью, и чем она меньше, тем лучше выполняется предсказание для текущей выборки. После разделения формула информативности преобразуется в:

Для минимизации информативности необходим правильный выбор и . Чтобы принять решение о разделении выборки, нужно сравнить значение информативности для изначального листа и для получившихся листьев после разделения.

Критерий ветвления *Branch*(*Xm,t*)*=*

Для задачи регрессии обычно используется среднеквадратичная ошибка:

*2*

Для ее минимизации берется среднее значение, а именно:

В итоге формула информативности выглядит следующим образом:

У решающих деревьев присутствует проблема переобучения, когда ошибка на обучающей выборке продолжает падать, а на тестовой, которая не участвует в обучении, с какого-то момента начинает расти. То есть модель идеально адаптируется к обучающим элементам, но плохо к общему набору элементов. Для предотвращения этого необходимо в какой-то момент останав-ливать процесс ветвления на листья.

Существует несколько вариантов ограничений:

1. По максимальной глубине дерева.
2. По минимальному числу объектов в листе.
3. По максимальному числу листьев во всем дереве.
4. По улучшению критерия ветвления при разделении выборки не менее чем на s процентов.

Чаще всего используются сразу все виды ограничений, это происходит во время построения дерева и называется early stopping.

## 2.1.2 Бэггинг над решающими деревьями

Основная идея использования нескольких моделей, а в данном случае решающих деревьев, обученных на имеющихся данных – получить более высокую точность предсказания на тестовых данных. Ансамбль из моделей решающих деревьев называется случайным лесом. Также один элемент из ансамбля называют базовым алгоритмом.

Пусть имеется обучающая выборка объектов *X*. Для обучения каждой из *k* моделей решающего дерева берется *n* элементов из этой выборки, причем выбор объектов происходит равновероятно и с возможным повторением. Получаются *k* выборок (), в каждой из которых часть элементов отсутствует, а часть входит несколько раз. В итоге после обучения *k* моделей, выполняющих задачу регрессии, происходит усреднение всех *k* предсказаний этих моделей. Модель – ансамбль моделей решающих деревьев.

Построение такого ансамбля выполняется следующим образом:

1. Формируется подвыборка ;
2. Случайным образом происходит выбор n < N признаков, где N – полное количество признаков.
3. На основе выбранных признаков подвыборки строится i-ое решающее дерево.

Затем шаги 1-3 повторяются, до выполнения критерия остановки построения леса.

При построении случайного леса необходимо учитывать несколько гиперпараметров:

1. Глубину деревьев. При малом количестве предикатов в дереве, проис-ходит запоминание только основных закономерностей обучающей выборки. Для тестового набора точность будет не очень высокой, но предсказания будут стабильными. При большой глубине в учет идут и низкоуровневые закономерности, поэтому предсказания на тестовой выборке будут больше зависеть от ее репрезентативности. Если она высокая, то и качество модели будет выше. Из этого следует вывод, что стоит применять относительно глубокие деревья.
2. Количество признаков, используемых для обучения одного дерева. Чем больше признаков берется для построения деревьев, тем сильнее они будут похожи друг на друга, то есть тем больше будет корреляция между ними. Тогда слабо работает идея ансамблирования. Если же брать слишком мало признаков, то сами деревья будут слабыми. Оптимальным для задач регрессии считается брать одну треть всех признаков для построения каждого из деревьев.
3. Количество деревьев в случайном лесу. Увеличение числа деревьев приводит к уменьшению дисперсии предсказаний алгоритма. Но это не проис-ходит до бесконечности, так как обучающая выборка имеет конечный размер. Поэтому целесообразно строить график зависимости ошибки на валидацион-ном датасете от количества деревьев в лесу и прекратить увеличение леса, когда график будет выходить на плато.

## **2.2 Градиентный бустинг над решающими деревьями**

Этот метод в какой-то степени является доработкой случайного леса. В случайном лесе решающие деревья строятся независимо друг от друга. Данный же метод строит композицию из деревьев, такую, что каждое новое дерево старается устранить ошибки предыдущих. Происходит движение в сторону уменьшения функции потерь. Это главное отличие от случайного леса.

Строится композиция из K деревьев фиксированной глубины:

Первое дерево строится так, чтобы уменьшить функцию ошибки Ошибка предсказания целевой переменной первым деревом равна:

Идея заключается в том, чтобы с помощью дерева скорректировать ошибку дерева , а общая ошибка ансамбля двух деревьев была равна 0 для предсказания целевой величины .

Высока вероятность, что не удастся с первого раза получить такое второе дерево, которое бы полностью корректировало ошибку первого, но можно выбрать из семейства деревьев такое, которое справляется с этой задачей лучше всего.

Далее описанные выше действия повторяются до построения композиции из K деревьев. Разность между ответом целевой переменной и предсказанием композиции равна:

Обучение композиции можно представить как перемещение в простран-стве из точки () в точку (), которая по ожиданиям будет находиться как можно более близко к точке истинных значений (*y1, y2,…, yN*). где *N* – количество элементов в обучающей выборке.

Данную задачу можно рассмотреть и со стороны уменьшения функции потерь, а не уменьшения расстояния между предсказаниями композиции и реальными значениями.

Рассмотрим квадратичную функцию потерь для задачи регрессии:

где – вектор признаков объекта; – ответ, соответствующий этому вектору. - дифференцируема.

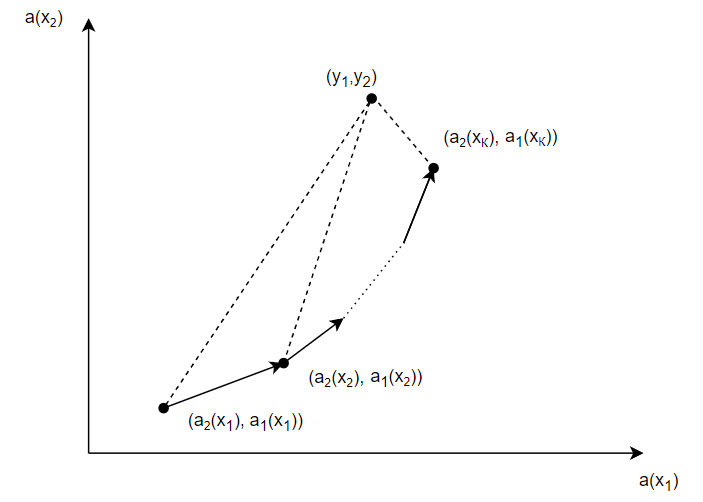
**

Рисунок 6– Движение в пространстве предсказаний композиции для объектов выборки к минимизации разницы предсказаний и реальными значениями

Теперь посчитаем производную по заданной функции потерь по предсказанию композиции для *i*-го объекта:

Отсюда видно, что разность ответа и предсказания композиции из К деревьев выражается через производную следующим образом:

Получается, что для каждого объекта из всей выборки очередное дерево в алгоритме градиентного бустинга обучается предсказывать антиградиент функции потерь по предсказаниям модели в точке , где - предсказания текущей части композиции на объекте .

На каждой итерации вновь добавляемое дерево обучается так, чтобы уменьшить значение функции потерь для текущей композиции:

Разложим функцию в ряд Тейлора до первого члена в окрестности точки для построения следующих деревьев:

Убрав постоянный член, получаем задачу оптимизации:

Видно, что эта сумма является скалярным произведением векторов, и ее значение минимизируется , пропорциональные значениям . Поэтому на каждой итерации новое дерево обучается предсказывать значения антиградиента функции потерь по предсказаниям текущей композиции и делает шаг в сторону вектора .

На рисунке 7 изображено два потенциальных движения из текущего предсказания (): в сторону уменьшения разницы предсказаний и реальных значений или в сторону антиградиента текущего предсказания. Эти направления не всегда могут совпадать, как это показано на рисунке (линиями обозначены уровни значений функции потерь), шаг в направлении антиградиента более выгоден.

При построении очередного решающего дерева решается задача регрессии с целевым значением, равным антиградиенту функции потерь для предсказания .

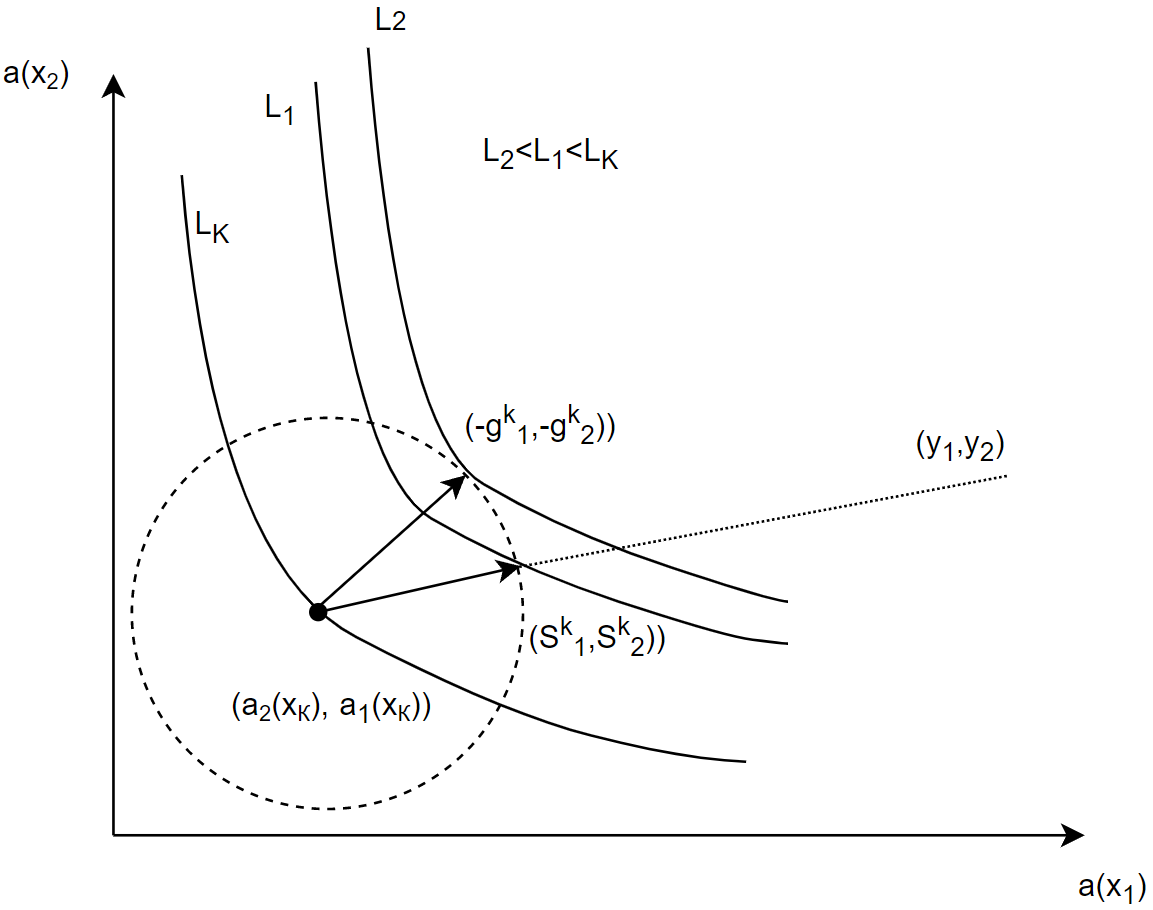


Рисунок 7 – движение в пространстве предсказаний композиции для объектов выборки в направлении антиградиента

Вводится оценочная функция *S*. Она показывает, насколько текущее дерево хорошо приближает антиградиент и используется для критерия ветвле-ния:

, ,– значения функции *S* в вершине, в левом потомке, правом потомке соответственно. – количество элементов в этих вершинах.

Обычно используют одну из следующих функций:

где – предсказание дерева на объекте,

, – антиградиент функции потерь.

Для того, чтобы построить следующее дерево используется следующий алгоритм:

1. Считается производная функции потерь по текущему предсказанию композиции:
2. Строится регрессионное дерево на обучающей выборке , которое минимизирует оценочную функцию.

Если глубина деревьев слишком большая, то как и в случайном лесу, при градиентном бустинге высока вероятность переобучения.

Существует два решения данной проблемы:

1. Уменьшение глубины деревьев
2. Ввод параметра learning rate (темп обучения), . Его значение задается вручную или определяется по выборке данных: количеству объектов в ней и признаков.
   1. **Искусственные глубокие нейронные сети**

### 2.3.1 Общие сведения

Искусственная нейронная сеть – сложная дифференцируемая функция, задающая отображение из исходного пространства признаков в пространство истинных значений (ответов), все параметры которой могут настраиваться одновременно и взаимосвязанно, то есть сеть может обучаться от одного конца до другого. Чаще всего представляет собой последовательность дифференцируемых параметрических преобразований. Такие сложные функ-ции представляют в виде суперпозиции более простых:

1. Линейных слоев. По сути, это линейное преобразование над данны-ми: . - матрица изменяющихся (обучающихся) весов слоя, - вектор сдвига (тоже обучаемые веса), - вектор входных данных. Происходит преобразование d-мерных векторов в k-мерные.
2. Функции активации. Это нелинейные преобразования, которые последовательно применяются к входным данным.

На рисунке 8 изображена нейросеть, в которой есть только линейные слои и функции активации, она называется полносвязной нейронной сетью или многослойным перцептроном.

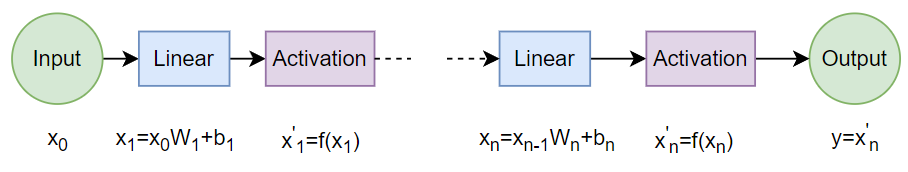


Рисунок 8 – Структура простейшей искусственной нейронной сети

Функции активации добавляются между линейными слоями для того, чтобы ввести нелинейность и улавливать более сложные закономерности. Так как линейная комбинация линейностей также линейна, нет смысла ставить две линейности подряд. Это легко показывается:

Наиболее популярные функции активации:

1. ReLU: . Является кусочно-линейной функцией. Производная в нуле доопределяется нулем.

Минусы: смещена относительно нуля; для отрицательных значений производная равна 0, что может привести к затуханию градиента.

Плюсы: простота вычисления функции и ее производной.

1. SeLU:, . Улучшение ReLU, параметр дает небольшой наклон в области отрицательных значений, что обеспечивает большую симметричность относительно нуля, а также меньше провоцирует затухание градиента.
2. Sigmoid:

Плюсы: ограниченная область значений.

Минусы: смещена относительно нуля; сложное вычисление из-за наличия экспоненты; очень маленькая производная на концах приводит к затуханию градиента; максимальное значение производной 0.25, что также приводит к затуханию градиента.

1. tanh(x):

Плюсы: ограниченная область значений; симметричная.

Минусы: сложные вычисления из-за наличия экспонент; очень маленькая производная на концах приводит к затуханию градиента

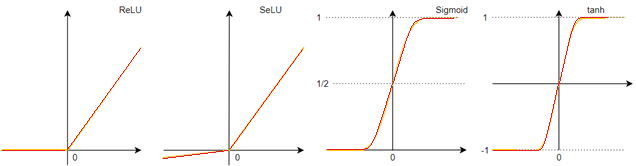


Рисунок 9 – Основные функции активации

### 2.3.2 Обучение нейронных сетей

Прямым проходом (forward propagation) называют применение нейрон-ной сети к исходным данным, то есть вычисление выхода по входу. Обучение нейронных сетей происходит при обратном проходе (backward propagation), от заданной функции потерь , назад через все преобразования, с измене-нием весов слоев нейронной сети. При этом применяется метод градиентного спуска. *y* – истинное значение, – значение на выходе нейросети.

Суть метода обратного распространения ошибки заключается в последо-вательном вычислении градиента при обратном проходе, начиная с последнего слоя и умножая на производные предыдущих слоев последовательно. Все осно-вано на правиле дифференцирования сложной функции:

По сути, градиент – это направление, в котором функция растет наиболее быстро. Поэтому необходимы антиградиенты, на основе которых происходит изменение обучаемых параметров каждого слоя для скорейшего убывания функции потерь.

Алгоритм backward propagation для выборки *X*, и текущем значении весов :

1. Произвести вычисления forward pass:
2. Вычислить все градиенты с помощью backward pass
3. Изменить веса с помощью полученных градиентов:

,

где – величина градиентного шага.

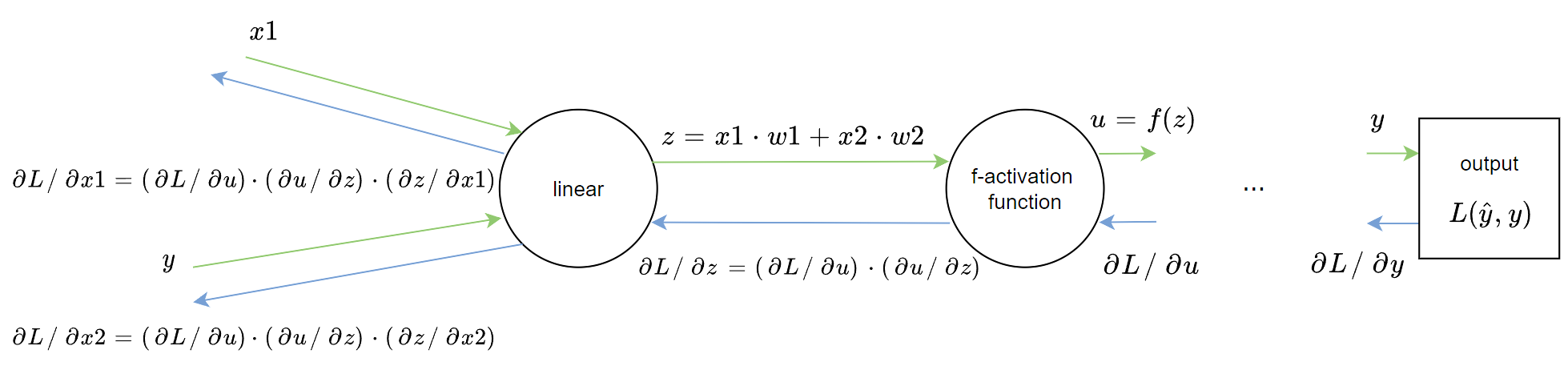


Рисунок 10 – Вычисление градиентов на промежуточных слоях

По сути, находим, как каждый из обучающихся параметров влияет на заданную функцию потерь и делаем шаг в сторону уменьшения этой ошибки.

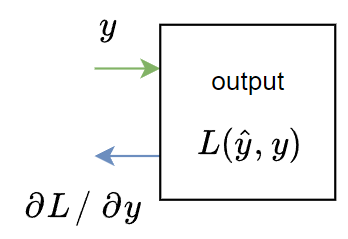


Рисунок 11 – Вычисление градиента на выходном слое

От гиперапараметра lr сильно зависят результаты обучения моделей. В зависимости от шага есть следующие возможные варианты обучения:

1. Шаг будет слишком большой, тогда произойдет расхождение, вместо схождения ошибки.
2. Шаг будет слишком большой, но не оптимальный, тогда ошибка будет уменьшаться, но не будет достигнут минимум
3. Шаг будет слишком маленький, нейросеть будет очень долго обучать-ся. И не будет достигнут глобальный минимум ошибки, а только локальный.
4. Шаг будет оптимальным, тогда получится достигнуть глобального минимума.

На рисунке 12 слева изображен график ошибки от одного из изменяю-щихся параметров при различных lr. Справа график ошибки от количества эпох при разных lr.

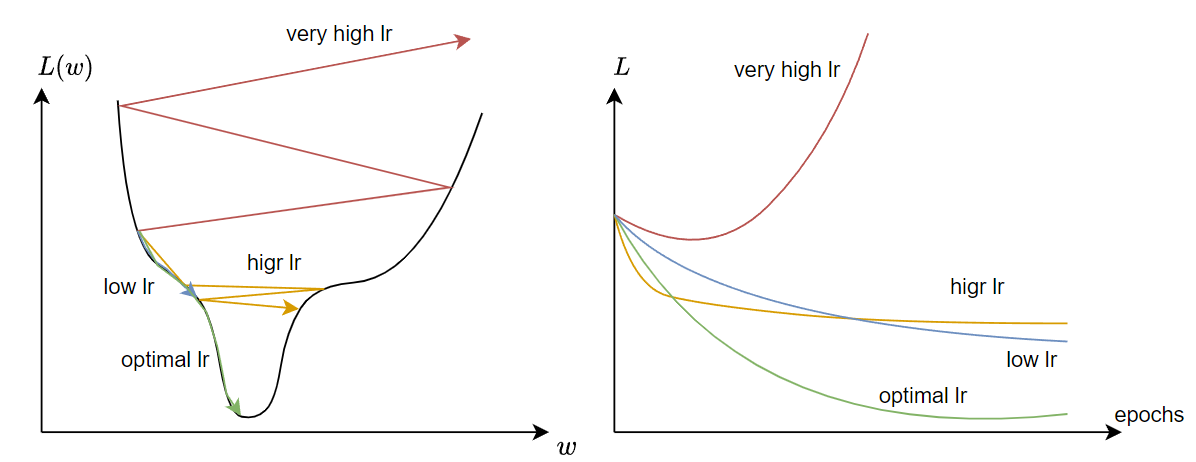


Рисунок 12 – Графики зависимостей ошибок при различных гиперпараметрах lr

При обучении необходимо разделять всю обучающую выборку на батчи – мини-выборки. Batch\_size – еще один важный гиперпараметр. Если выборка объектов для обучения очень большая, то считать градиенты для всех объектов за раз очень дорого в вычислительном плане. Если же брать только один объект, то он может оказаться выбросом, тогда обучение модели произойдет не в нужную сторону. Поэтому берется случайная подвыборка из Batch\_size объектов, вычисляются градиенты, обновляются веса нейросети.

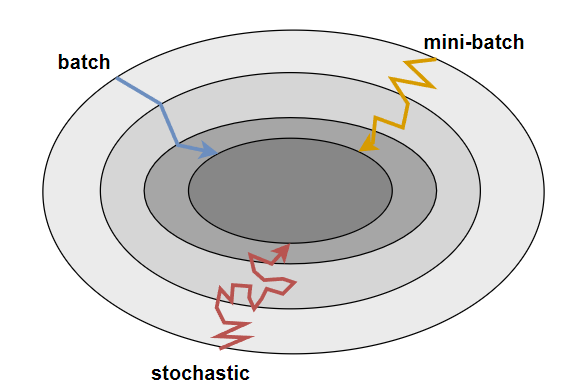


Рисунок 13 – Движения к уменьшению общей ошибки на всей обучающей выборке при различных значениях batch\_size

Симметричность функций активации является полезным свойством. Допустим изначально все веса проинициализировались, как положительные. Такая несимметричная функция относительно 0, как сигмоида, для положи-тельных чисел дает результат больше, чем 0.5. С каждым следующим слоем после сигмоид, значения будут все ближе к 1, и при расчете градиентов, они начнут затухать. Пусть нейронная сеть состоит из линейных слоев с функцией активации сигмоида. Тогда и если элементы будут достаточно большими, то градиент функции потерь будет близок к нулю и веса не обновятся, то есть не произойдет обучение. График производной сигмоиды изображен на рисунке 14.

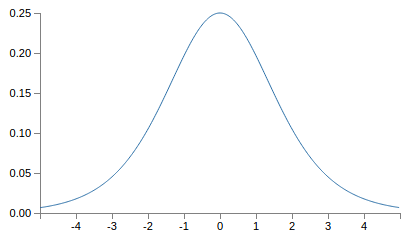


Рисунок 14 – График производной сигмоиды

**2.4 Искусственные рекуррентные нейронные сети LSTM**

### 2.4.1 Общие положения о рекуррентных нейронных сетях

Основной минус описанных выше методов – потеря связи между последовательностью элементов временного ряда. На вход модели они поступают как обычные признаки, независящие друг от друга. Рекуррентные нейронные сети используют обратные связи, которые позволяют улавливать информацию, содержащуюся в последовательности элементов. Такую сеть можно представить как копии самой себя (иногда копии называют блоками), передающие информацию друг другу. На ее вход последовательно поступают элементы и какая-то информация с прошлого блока (кроме первого блока), на выходе получается результат . В итоге последовательности соответ-ствует последовательность . Это изображено на рисунке 15.

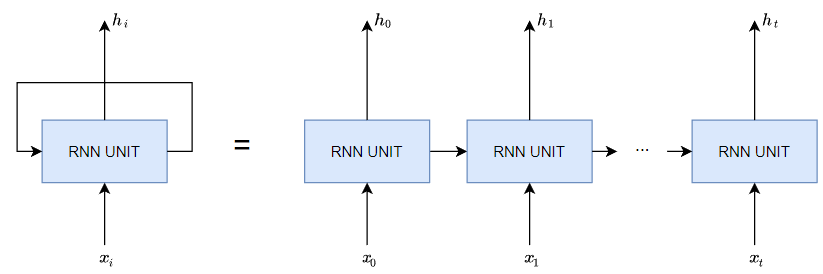


Рисунок 15 – Принцип работы рекуррентных нейронных сетей

Существует несколько вариантов использования рекуррентных нейрон-ных сетей: один элемент к одному, один ко многим, многие к одному, многие ко многим. Как на входе, так и на выходе может быть как один входной вектор, так и последовательность векторов.

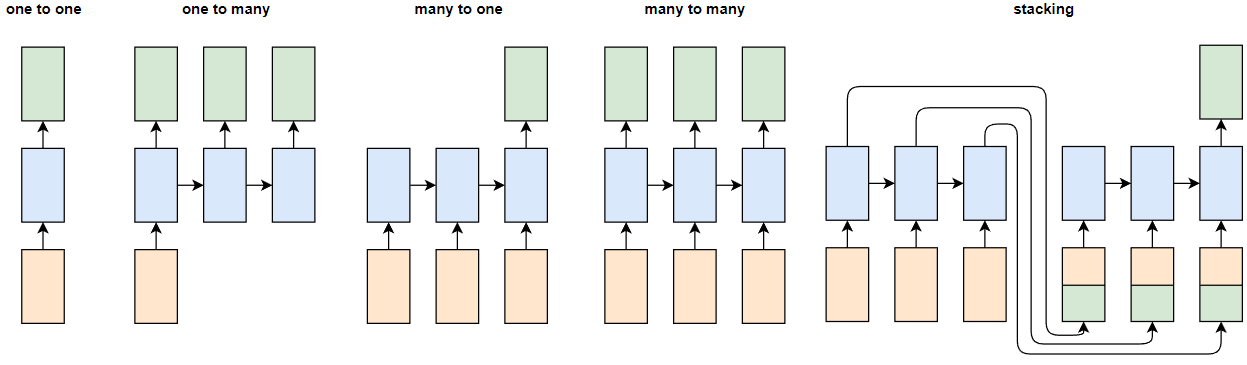


Рисунок 16 – варианты использования рекуррентных нейронных сетей

Для прогнозирования временных рядов имеет смысл использовать концепцию многие к одному, так как последовательности значений соответствует одно целевое значение. Но есть вариант стэкинга, то есть добавления нескольких слоев, при чем все кроме последнего работают по концепции многие ко многим, то есть выдают на выходе последовательность, которая является входной для следующего слоя. Последний же слой работает по концепции многие к одному.

### Сети LSTM

LSTM (Long-Short Term Memory) – долгая краткосрочная память. Это наиболее удачная модификация рекуррентных нейронных сетей, так как она позволяет улавливать не только недавнюю информацию, но и долговременную. На рисунке 18 изображена структура блока LSTM.

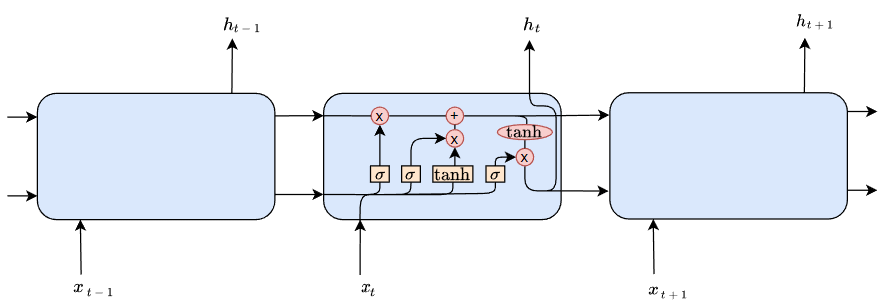


Рисунок 17 – Внутренняя структура LSTM

У него есть два входа с предыдущего блока, вход текущего значения входной последовательности данных . Также имеется два выхода, которые идут в следующий блок, при этом значении одного из выходов является значением выходной последовательности . Стрелками указаны направления передачи векторов; соединение двух стрелок – операция конкатенации; разъе-динение – копирование; красным цветом обозначены поэлементные операции; оранжевым – идущие друг за другом линейный и активационный слои нейронной сети.

На рисунке 18 рассмотрены происходящие с данными процессы внутри блока LSTM более детально. Главной особенностью LSTM является состояние ячейки , в котором и хранится основная информация о последовательности. Ее можно как добавлять в ячейку, так и удалять оттуда. Эти действия производятся с помощью так называемых вентилей, которые состоят из sigmoid слоя и последующей операции поэлементного умножения.

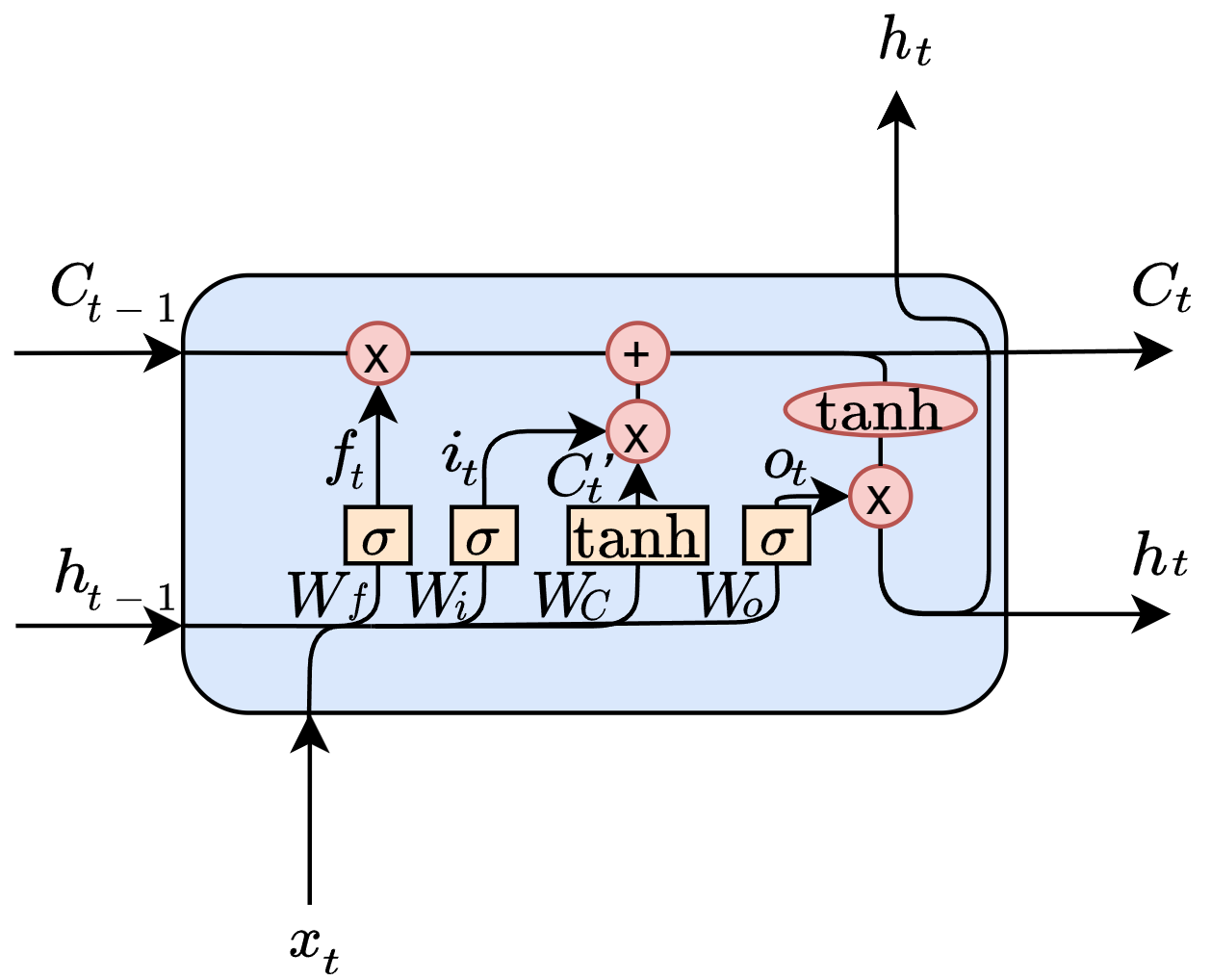


Рисунок 18 – движение и действия над векторами в LSTM

Первым стоит вентиль забывания, который определяет какую информацию удалять из ячейки. На его вход подается конкатенация и , прошедшая через линейный слой с обучающимися параметрами , а затем через сигмоидальную функцию активации. На выходе она выдает число от 0 до 1 для каждого элемента ячейки . 0 эквивалентно полному забыванию, а 1 – сохранению информации.

Следующий вентиль – запоминания, он отвечает за запись новой информации в ячейку и состоит из двух частей. Аналогично вентилю забывания стоит слой с параметрами и сигмоидой, а также слой с параметрами и гиперболическим тангенсом. Первая часть отвечает за то, какие значения необходимо обновить, а вторая за формирование вектора новых значений , которые будут добавлены в состояние ячейки.

Теперь состояние обновляется до последовательным умножением на вектор (выхода вентиля забывания) и сложением с поэлементным произведением векторов и . Тем самым произошло забывание ненужной информации и запоминание нужной новой.

Выходом из блока является вектор , являющимся поэлементным произведением и состояния ячейки, после применения к ее значениям гиперболического тангенса – . – результат выхода слоя с обучаемыми параметрами и последующим слоем сигмоиды.

Формулы, описывающие работу блока LSTM:

Обучение рекуррентных нейронных сетей производится методом обратного распространения ошибки.

1. **Сравнение методов прогнозирования**
   1. **Выбор метрики и оптимизация моделей**

После создания моделей машинного обучения, работающих по разным методам, необходимо сравнить качество их прогнозов, то есть насколько хорошо они выполняют задачу регрессии. Это выполняется на тестовом дата-сете, который никак не участвовал в обучении. Для оценки качества моделей необходимо выбрать метрику – критерий качества, который будет зависеть от предсказательных меток. Иногда может совпадать с функцией потерь. Во всех методах при обучении использовалась функция потерь MSE (среднеквад-ратичная ошибка), так как она дифференцируема. В качестве метрики возьмем абсолютную ошибку:

У моделей есть большое количество гиперпараметров, которые влияют на качество моделей. Их можно менять вручную, пытаясь добиться наилуч-шего качества модели на валидационном датасете. Но можно это автоматизи-ровать и использовать метод Grid search – создать сетку из различных значений гиперпараметров и обучить большое количество моделей. После этого выбрать ту модель, у которой наименьшая ошибка на валидационной выборке.

Гиперпараметры для которых использовался Grid search в каждом из методов:

1. Случайный лес: количество деревьев; максимальная глубина дерева; минимальное число объектов для разделения на листы; минимальное число объектов в листе; максимальное количество признаков, используемых деревом
2. Градиентный бустинг: количество деревьев; максимальная глубина дерева; максимальное количество листьев; темп обучения
3. Нейронные сети: размер батча, количество эпох, темп обучения
4. LSTM: размер батча, количество эпох, темп обучения, количество блоков LSTM.

Кроме этого, Grid search был применен для следующих вариантов использования LSTM: многие ко многим и стэкинг.

* 1. **Сравнение моделей**

После нахождения оптимальных гиперпараметров можно делать прогнозы на тестовых данных и сравнивать с реальными значениями.

Также стоит отметить, что при формировании датасетов можно брать разный размер окна, то есть использовать разное количество предыдущих значений для предсказаний. Рассмотрим графики разностей предсказаний моделей и реальных значений при размере окна в 15 (то есть используются данные о температуре за последние 60 минут) и предсказании на 7 элементов (на 28 минут) вперед. Каждая точка на таком графике значение MAE для одного экземпляра тестовой выборки.

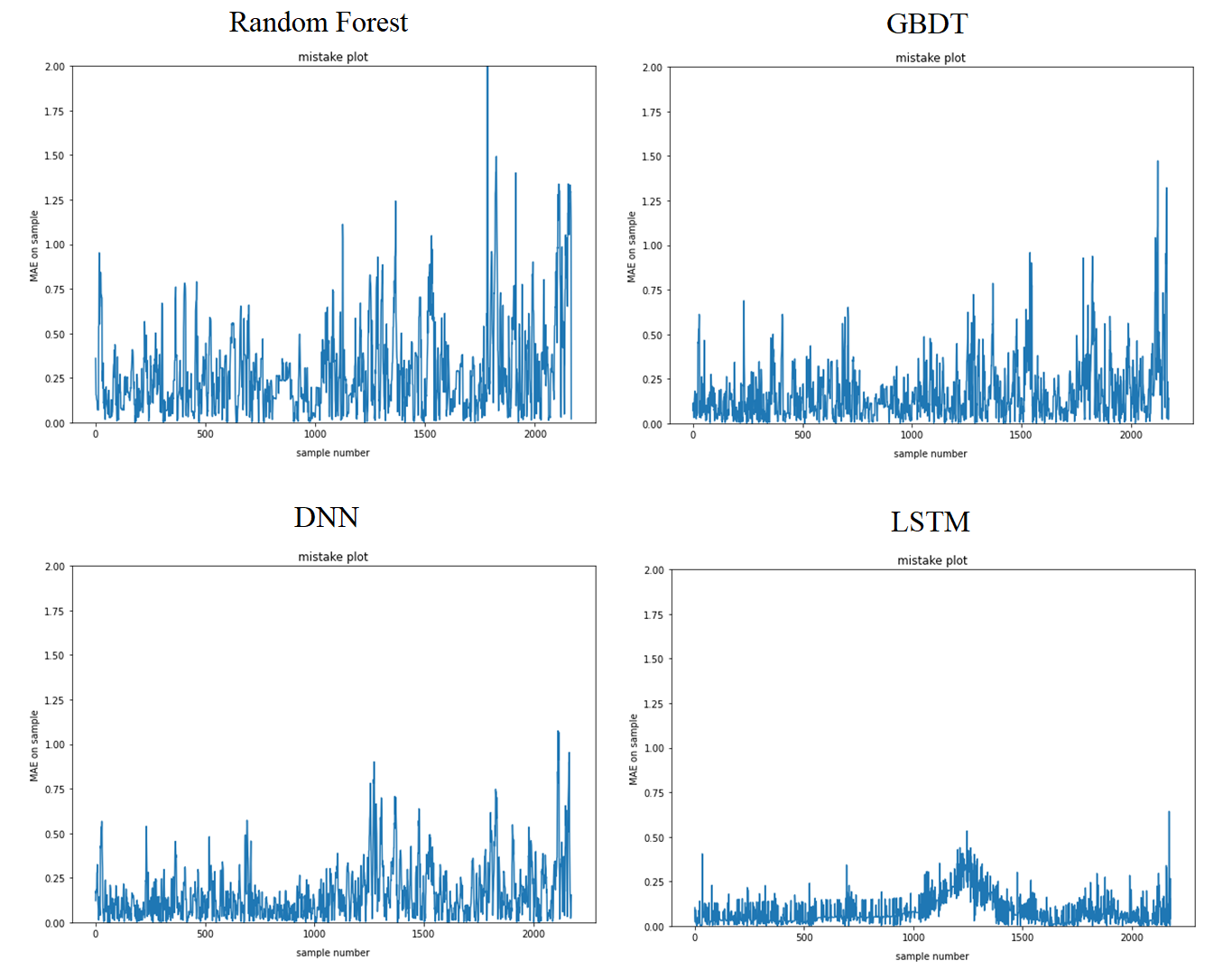


Рисунок 19 – графики MAE для разных моделей с использованием данных за 1 час и предсказанием на 28 минут вперед

Средняя абсолютная ошибка MAE на методах составила:

1. Случайный лес: 0.285
2. Градиентный бустинг над решающими деревьями: 0.164
3. Глубокие нейронные сети: 0.152
4. LSTM: 0.082

Теперь проведем похожий эксперимент, но при формировании датасета будем брать каждое седьмое значение, то есть одно показание раз в 28 минут. Предсказывать будем по 15 предыдущим показаниям (данные за 7 часов) на один элемент вперед (на 28 минут), то есть глубина предсказания такая же, как в прошлом эксперименте.

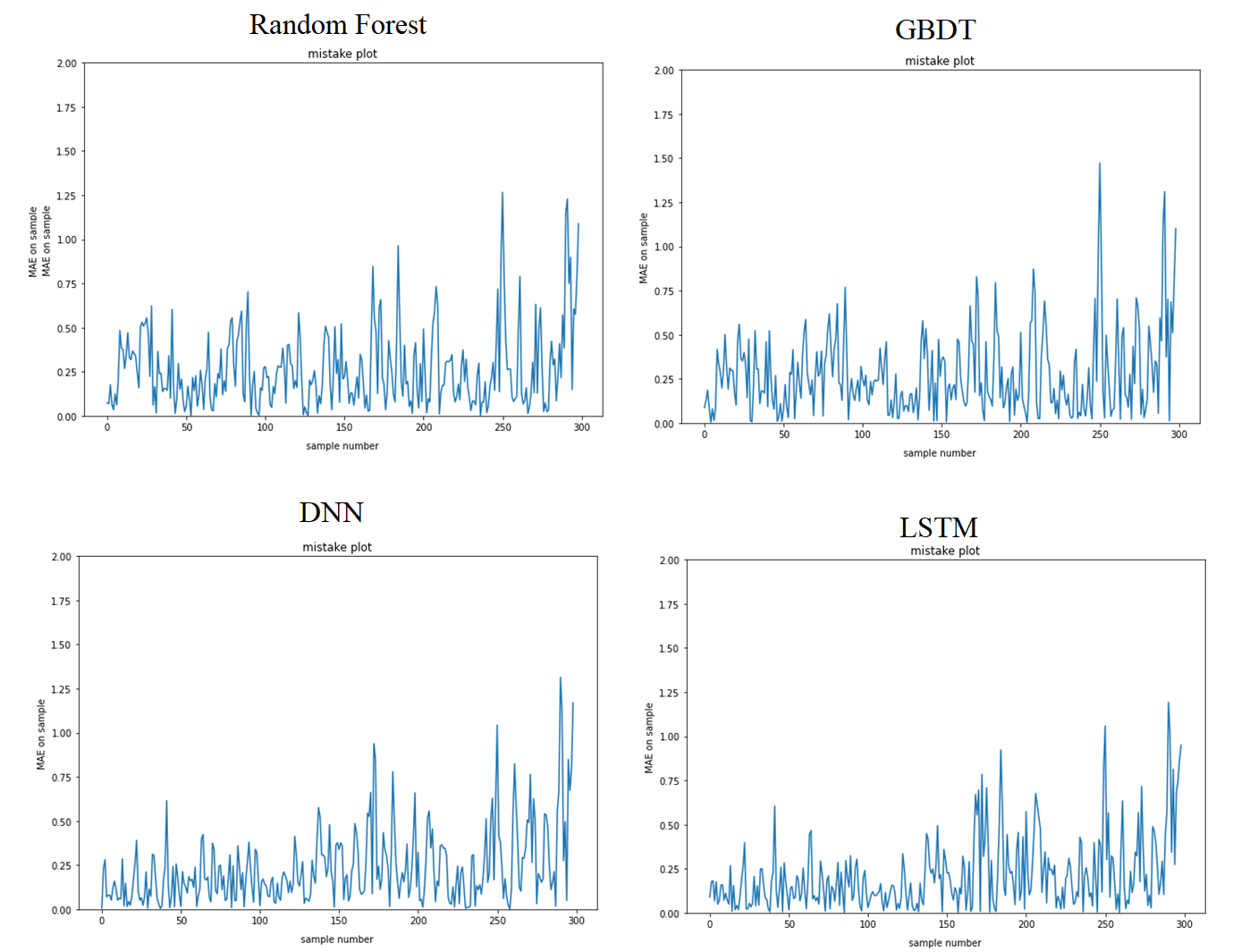


Рисунок 20 – графики MAE для разных моделей с использованием прореженных данных за 7 часов и предсказанием на 28 минут вперед

Средняя абсолютная ошибка MAE на методах составила:

1. Случайный лес: 0.264
2. Градиентный бустинг над решающими деревьями: 0.241
3. Глубокие нейронные сети: 0.232
4. LSTM: 0.18

Как можно видеть по результатам эксперимента, более качественные предсказания модели дают, используя все данные за ближайшее время, чем используя прореженные данные, но за больший временной интервал. Это связанно с тем, что следующее значение температуры больше зависит от последних предшествующих и их изменений.

Также можно сделать гипотезу, что более важны разности между температурами, чем сами значения. Для проведения такого эксперимента был сформирован датасет разностей, в каждом элементе которого разность между текущей температурой и температурой в момент прошлого измерения.

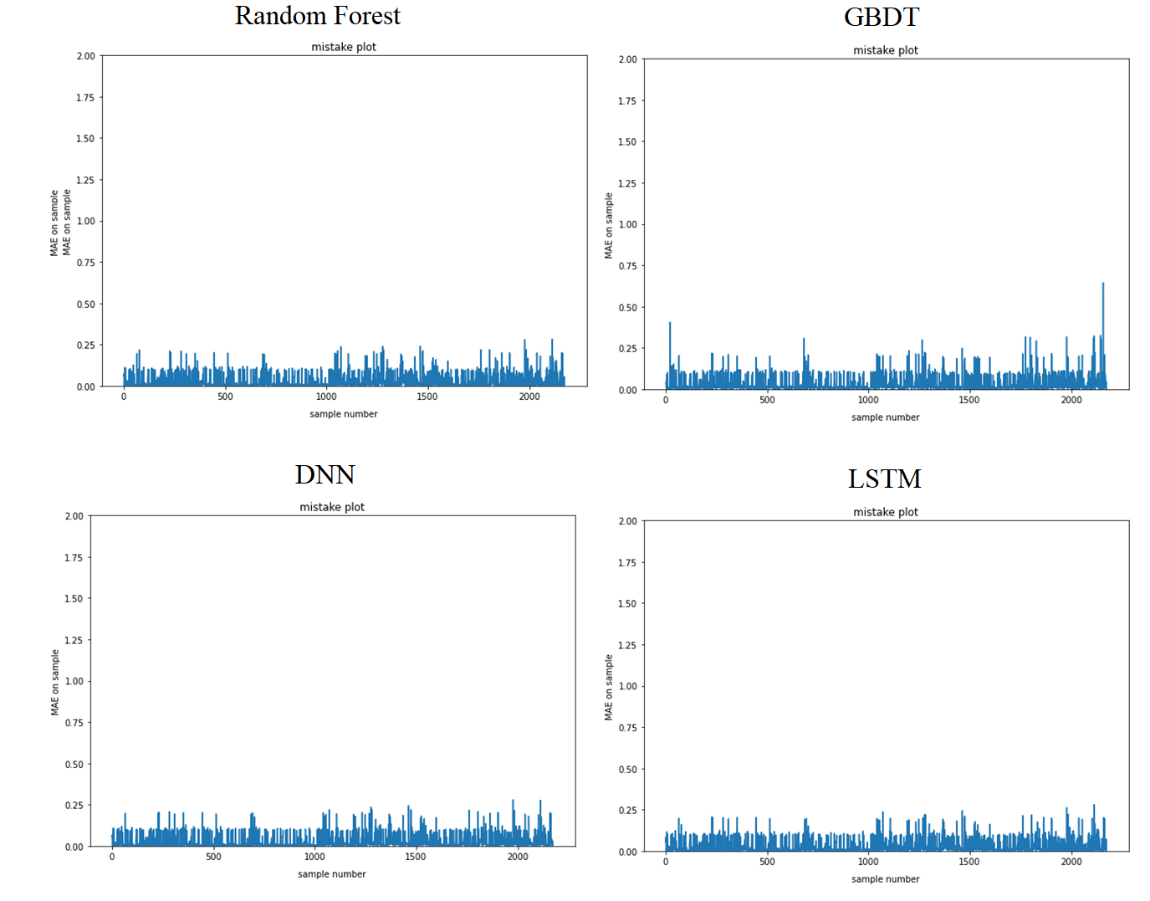
****

Рисунок 21 – графики MAE для разных моделей с использованием данных за 1 час и предсказанием на 28 минут вперед для разностей температур

Средняя абсолютная ошибка MAE на методах составила:

1. Случайный лес: 0.034
2. Градиентный бустинг над решающими деревьями: 0.028
3. Глубокие нейронные сети: 0.03
4. LSTM: 0.023

Гипотеза о том, что при работе с разностями качество возрастет подтвердилась. Также это позволяет использовать модель при любых температурах, так как важны только изменения, а не абсолютные значения. Это очень важно при условиях недостатка данных для обучения.

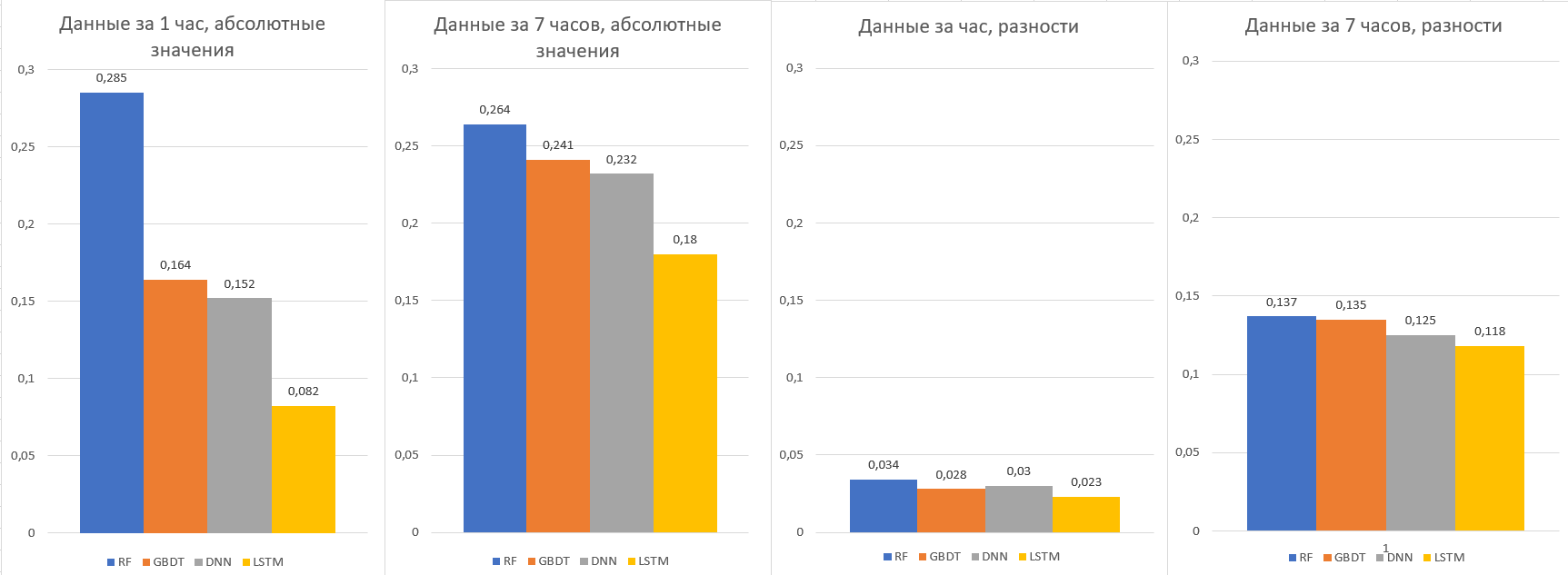


Рисунок 22 – сравнение МАЕ моделей при разных методах прогнозирования

Оказалось, что для всех экспериментов самую низкую точность предсказаний показал случайный лес. Более высокую точность дал градиентный бустинг, что закономерно, так как этот метод – существенная доработка случайного леса. Глубокие нейронные сети дали примерно тот же результат. Как и ожидалось, рекуррентные нейронные сети LSTM дали наилучшие прогнозы, это связано с тем, что они предназначены для работы с последовательностями данных.

Также были протестированы варианты ансамблей разных методов, то есть усреднения предсказаний разных моделей. Но не один из ансамблей не дал точность выше, чем LSTM отдельно.

После сравнения качества предсказаний моделей делаем вывод, что LSTM лучше всего справляется с задачей прогнозирования. Теперь для реализации предиктивного модуля можем использовать одну или несколько обученных моделей, каждая из которых может предсказывать значение температуры на определенное время вперед. Для этого обученная модель загружается в оперативную память, считываются приходящие с метеостанции данные за определенный промежуток времени (столько последних значений, сколько нужно модели) и моделью выполняется предсказание, которое записывается в нужный регистр ПЛК. Он, используя эту информацию выполняет регулирование заслонкой.

# **Заключение**

В рамках выпускной квалификационно работы были рассмотрены четыре метода прогнозирования временных рядов: случайный лес, градиентный бустинг над решающими деревьями, искусственные глубокие нейронные сети, искусственные рекуррентные нейронные сети LSTM.

Было проведено несколько экспериментов: модели разных методов обучались на абсолютных значениях температуры, на разницах между текущим и прошлым значением; датасеты для обучения составлялись с использованием разного количества предыдущих измерений.

В результате экспериментов выяснилось, что хуже всего с задачей справляется случайный лес, далее, примерно на одном уровне стоят градиентный бустинг и глубокие нейронные сети, лучше всего же для прогнозирования подходит LSTM. Также оказалось, что лучше использовать разности температур, чем абсолютные значения, и брать данные за ближайшее время, чем использовать прореженные данные, но за больший временной интервал.

В качестве лучшей модели выбрана искусственная нейронная сеть LSTM, так как она показала наилучшие результаты и наиболее подходит для работы с такими последовательностями, как временные ряды. На ее основе можно производить прогнозирование температуры и соответственно выполнять погодное предиктивное управление тепловым пунктом.

Разработка всех моделей производилась на языке Python в среде Jupiter Notebook.

# **Список литературы**

1. Будыльский, Д. В. GRU и LSTM: современные рекуррентные нейронные сети / Д. В. Будыльский, - Текст : непосредственный // Молодой ученый. – 2015. – № 15(95). – С. 51-54.
2. Boas, Matheus. Optimal Decision Trees for the Algorithm Selection Problem: Integer Programming Based Approaches / Matheus Boas, G. Vilas [et al]. - Int. Trans. Oper. Res  28 (2021): 2759-2781. – Текст непосредственный.
3. McQuistan Adam. Using Machine Learning to Predict the Weather: Part 1. / Adam McQuistan. – URL : <https://stackabuse.com/using-machine-learning-to-predict-the-weather-part-1> (дата обращения : 4.05.2022). – Текст : электронный.
4. Pialunga Piero. Weather forecasting with Machine Learning, using Python / Piero Pialung/ - URL: <https://towardsdatascience.com/weather-forecasting-with-machine-learning-using-python-55e90c346647> (Дата обращения 30/05/2022). - Текст : электронный.
5. Воронцов К. В. Глубокие нейронные сети / Воронцов К. В./ – URL: <http://www.machinelearning.ru/wiki/images/7/71/Voron-ML-DeepLearning-slides.pdf> (Дата обращения 3/05/2022). - Текст : электронный.
6. Joseph Rocca. Ensemble methods: bagging, boosting and stacking / Joseph Rocca / - URL: <https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205> (Дата обращения 10/05/2022). - Текст : электронный.
7. Steven Yu. Stacking and Blending — An Intuitive Explanation / Steven Yu / - URL: <https://medium.com/@stevenyu530_73989/stacking-and-blending-intuitive-explanation-of-advanced-ensemble-methods-46b295da413c> (Дата обращения 25/05/2022). - Текст : электронный.
8. Документация к библиотеке машинного обучения XGBoost. / - URL: <https://xgboost.readthedocs.io/en/stable/> (Дата обращения 23/05/2022). - Текст : электронный.
9. Документация к библиотеке машинного обучения CatBoost. / - URL: <https://catboost.ai/en/docs/references/training-parameters/> (Дата обращения 14/05/2022). - Текст : электронный.
10. Документация к библиотеке машинного обучения Keras. / - URL: <https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM> (Дата обращения 15/05/2022). - Текст : электронный.
11. Документация к библиотеке машинного обучения PyTorch / - URL: <https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html> (Дата обращения 13/05/2022). - Текст : электронный.
12. Документация к библиотеке машинного обучения scikit learn / - URL: [https://scikit-learn.org/stable/modules/ensemble.html#](https://scikit-learn.org/stable/modules/ensemble.html) (Дата обращения 25/05/2022). - Текст : электронный.
13. Учебник по машинному обучению от ШАД / - URL: <https://ml-handbook.ru/> (Дата обращения 25/05/2022). - Текст : электронный.

**Приложение А.**

**Модель случайного леса**

from datetime import timedelta

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot as plt

import plotly.express as px

from typing import List

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.graphics.tsaplots import plot\_pacf

import pandas as pd

data = pd.read\_csv('dataTnv.csv',index\_col=False)

dataTemp = pd.read\_csv('dataTnv.csv',index\_col=False, usecols=["avg\_value"])

datainter=dataTemp.fillna(data.interpolate())

dataParse\_time=pd.to\_datetime(data.btime, format="%Y-%m-%d %X+03")

df=pd.concat([dataParse\_time,datainter],axis=1)

train\_size = int(df.shape[0] \* 0.7)

val\_size = int(df.shape[0] \* 0.2)

test\_size = int(df.shape[0] \* 0.1)

train\_df = df.iloc[:train\_size]

val\_df = df.iloc[train\_size:-test\_size]

test\_df = df.iloc[train\_size+val\_size:]

train\_df.shape, val\_df.shape, test\_df.shape

df.loc[:train\_size, 'partition'] = 'train'

df.loc[train\_size:train\_size+val\_size, 'partition'] = 'validation'

df.loc[train\_size+val\_size:, 'partition'] = 'test'

fig = px.line(

    data\_frame = df.reset\_index(),

    x      = 'btime',

    y      = 'avg\_value',

    color  = 'partition',

    width  = 900,

    height = 500)

fig.update\_xaxes(

    rangeslider\_visible=True,

    rangeselector=dict(

        buttons=list([

            dict(count=1, label="h", step="hour", stepmode="backward"),

   Продолжение приложения А

         dict(count=24, label="24h", step="hour", stepmode="backward"),

            dict(count=24\*7, label="w", step="hour", stepmode="backward"),

            dict(count=1, label="m", step="month", stepmode="backward"),

            dict(step="all")

        ])

    )

)

fig.show()

df=df.drop(columns='partition')

def make\_dataset\_for\_rf(df, window\_size,front\_number, use\_scaler=False):

    features\_arr = df[["avg\_value"]].iloc[:-(front\_number)]

    features\_arr = np.array(features\_arr, dtype=np.float32)

    if use\_scaler:

        features\_arr = scaler.transform(features\_arr)

    target = df[["avg\_value"]].iloc[(window\_size+front\_number):]

    target = np.array(target, dtype=np.float32)

    features = []

    for i in range(len(target)):

        features.append(features\_arr[i:window\_size+i])

    features = np.asarray(features)

    return features,target

window\_size = 15

front\_number = 7

x\_train,y\_train = make\_dataset\_for\_rf(df=train\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_val,y\_val = make\_dataset\_for\_rf(df=val\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_test,y\_test = make\_dataset\_for\_rf(df=test\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_train = x\_train.reshape(len(y\_train),window\_size)

x\_val = x\_val.reshape(len(y\_val),window\_size)

x\_test = x\_test.reshape(len(y\_test),window\_size)

y\_train = y\_train.reshape(len(y\_train))

y\_test = y\_test.reshape(len(y\_test))

from sklearn.ensemble import RandomForestRegressor

from sklearn.datasets import make\_regression

Окончание приложения А

regr = RandomForestRegressor(n\_estimators=1000, max\_depth=10, min\_samples\_split=2, min\_samples\_leaf=1,verbose=1)

regr.fit(x\_train, y\_train)

prediction = regr.predict(x\_test)

val\_time = val\_df[:len(-y\_test)].drop(columns=["avg\_value"])

data\_plot = pd.DataFrame({

                'test': np.reshape(y\_test,len(y\_test)),

                'prediction': prediction,

                 })

data\_plot.index.name = 'date\_time'

fig = px.line(

    data\_frame = data\_plot.reset\_index(),

    x      = val\_time['btime'],

    y      = data\_plot.columns,

    width  = 900,

    height = 500

    )

fig.update\_xaxes(

    rangeslider\_visible=True,

    rangeselector=dict(

        buttons=list([

            dict(count=1, label="h", step="hour", stepmode="backward"),

            dict(count=4, label="4h", step="hour", stepmode="backward"),

            dict(count=24, label="24h", step="hour", stepmode="backward")

            dict(step="all")

        ])

    )

)

fig.show()

import sys

np.set\_printoptions(threshold=sys.maxsize)

dif\_xgb = np.array(prediction-y\_test.reshape(len(y\_test)))

plt.figure(figsize=(10, 7))

plt.plot(val\_time,dif\_xgb)

plt.title('mistake plot')

plt.ylabel('dif\_temperature')

plt.xlabel('date,time',labelpad=10)

plt.xticks( rotation='vertical')

dif\_xgb = np.abs(dif\_xgb)

print(dif\_xgb.mean())

Рисунок А.1 – Листинг программы ПЛК метода Случайный лес

**Приложение Б.**

**Модель градиентного бустинга**

from datetime import timedelta

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot as plt

import plotly.express as px

from typing import List

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.graphics.tsaplots import plot\_pacf

import pandas as pd

data = pd.read\_csv('dataTnv.csv',index\_col=False)

dataTemp = pd.read\_csv('dataTnv.csv',index\_col=False, usecols=["avg\_value"])

datainter=dataTemp.fillna(data.interpolate())

dataParse\_time=pd.to\_datetime(data.btime, format="%Y-%m-%d %X+03")

df=pd.concat([dataParse\_time,datainter],axis=1)

train\_size = int(df.shape[0] \* 0.7)

val\_size = int(df.shape[0] \* 0.2)

test\_size = int(df.shape[0] \* 0.1)

train\_df = df.iloc[:train\_size]

val\_df = df.iloc[train\_size:-test\_size]

test\_df = df.iloc[train\_size+val\_size:]

train\_df.shape, val\_df.shape, test\_df.shape

df.loc[:train\_size, 'partition'] = 'train'

df.loc[train\_size:train\_size+val\_size, 'partition'] = 'validation'

df.loc[train\_size+val\_size:, 'partition'] = 'test'

fig = px.line(

    data\_frame = df.reset\_index(),

    x      = 'btime',

    y      = 'avg\_value',

    color  = 'partition',

    width  = 900,

    height = 500)

fig.update\_xaxes(

    rangeslider\_visible=True,

    rangeselector=dict(

        buttons=list([

            dict(count=1, label="h", step="hour", stepmode="backward"),

            dict(count=24, label="24h", step="hour", stepmode="backward"),

            dict(count=24\*7, label="w", step="hour", stepmode="backward"),

            dict(count=1, label="m", step="month", stepmode="backward"),

Продолжение приложения Б

            dict(step="all") ])))

fig.show()

df=df.drop(columns='partition')

def make\_dataset\_for\_gbdt(df, window\_size,front\_number, use\_scaler=False):

    features\_arr = df[["avg\_value"]].iloc[:-(front\_number)]

    features\_arr = np.array(features\_arr, dtype=np.float32)

    if use\_scaler:

        features\_arr = scaler.transform(features\_arr)

    target = df[["avg\_value"]].iloc[(window\_size+front\_number):]

    target = np.array(target, dtype=np.float32)

    features = []

    for i in range(len(target)):

        features.append(features\_arr[i:window\_size+i])

    features = np.asarray(features)

    return features,target

window\_size = 15

front\_number = 7

x\_train,y\_train = make\_dataset\_for\_gbdt(df=train\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_val,y\_val = make\_dataset\_for\_gbdt(df=val\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_test,y\_test = make\_dataset\_for\_gbdt(df=test\_df, window\_size=window\_size, front\_number=front\_number, use\_scaler=False)

x\_train = x\_train.reshape(len(y\_train),window\_size)

x\_val = x\_val.reshape(len(y\_val),window\_size)

x\_test = x\_test.reshape(len(y\_test),window\_size)

import xgboost

from xgboost import XGBRegressor

model = XGBRegressor(n\_estimators=1000, max\_depth=7, learning\_rate=0.01)

model.fit(x\_train,y\_train,

         eval\_set=[(x\_train,y\_train),(x\_val,y\_val)],

         eval\_metric='mae',

         verbose=True, callbacks=[xgboost.callback.EarlyStopping(rounds=15,save\_best=True)])

model.best\_ntree\_limit

results = model.evals\_result()

plt.figure(figsize=(10,7))

plt.plot(results["validation\_0"]["mae"], label="Training loss")

Окончание приложения Б

plt.plot(results["validation\_1"]["mae"], label="Validation loss")

plt.axvline(model.best\_ntree\_limit, color="gray", label="Optimal tree number")

plt.xlabel("Number of trees")

plt.ylabel("Loss")

plt.legend()

prediction = model.predict(x\_test)

val\_time = val\_df[:len(-y\_test)].drop(columns=["avg\_value"])

data\_plot = pd.DataFrame({

                'test': np.reshape(y\_test,len(y\_test)),

                'prediction': prediction, })

data\_plot.index.name = 'date\_time'

fig = px.line(

    data\_frame = data\_plot.reset\_index(),

    x      = val\_time['btime'],

    y      = data\_plot.columns,

    width  = 900,

    height = 500

    )

fig.update\_xaxes(

    rangeslider\_visible=True,

    rangeselector=dict(

        buttons=list([

            dict(count=1, label="h", step="hour", stepmode="backward"),

            dict(count=4, label="4h", step="hour", stepmode="backward"),

            dict(count=24, label="24h", step="hour", stepmode="backward"),

            dict(step="all")])))

fig.show()

import sys

np.set\_printoptions(threshold=sys.maxsize)

dif\_xgb = np.array(prediction-y\_test.reshape(len(y\_test)))

plt.figure(figsize=(10, 7))

plt.plot(val\_time,dif\_xgb)

plt.title('mistake plot')

plt.ylabel('dif\_temperature')

plt.xlabel('date,time',labelpad=10)

plt.xticks( rotation='vertical')

dif\_xgb = np.abs(dif\_xgb)

print(dif\_xgb.mean())

Рисунок Б.1 – Листинг программы для модели градиентного бустинга

**Приложение В.**

**Модель искусственной нейронной сети**

from datetime import timedelta

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot as plt

import plotly.express as px

from typing import List

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.graphics.tsaplots import plot\_pacf

import pandas as pd

data = pd.read\_csv('dataTnv.csv',index\_col=False)

dataTemp = pd.read\_csv('dataTnv.csv',index\_col=False, usecols=["avg\_value"])

datainter=dataTemp.fillna(data.interpolate())

dataParse\_time=pd.to\_datetime(data.btime, format="%Y-%m-%d %X+03")

df=pd.concat([dataParse\_time,datainter],axis=1)

df = df.iloc[::7, :]

df['dif\_avg\_value'] = np.append(0,(np.array(df['avg\_value'][1:]) - np.array(df['avg\_value'][:-1])))

df.loc[df.dif\_avg\_value > 0.3,'dif\_avg\_value'] = 0

df.loc[df.dif\_avg\_value < -0.3,'dif\_avg\_value'] = 0

train\_size = int(df.shape[0] \* 0.7)

val\_size = int(df.shape[0] \* 0.2)

test\_size = int(df.shape[0] \* 0.1)

train\_df = df.iloc[:train\_size]

val\_df = df.iloc[train\_size:-test\_size]

test\_df = df.iloc[train\_size+val\_size:]

df.loc[:train\_size, 'partition'] = 'train'

df.loc[train\_size:train\_size+val\_size, 'partition'] = 'validation'

df.loc[train\_size+val\_size:, 'partition'] = 'test'

fig, ax = plt.subplots(figsize=(11, 4))

train\_df['dif\_avg\_value'].plot(ax=ax, label='train')

val\_df['dif\_avg\_value'].plot(ax=ax, label='validation')

test\_df['dif\_avg\_value'].plot(ax=ax, label='test')

ax.legend()

def make\_dataset\_for\_NN(df, window\_size,ahead\_number, use\_scaler=False):

    features\_arr = df[["dif\_avg\_value"]].iloc[:-(ahead\_number)]

    features\_arr = np.array(features\_arr, dtype=np.float32)

    if use\_scaler:

      Продолжение приложения В

  features\_arr = scaler.transform(features\_arr)

    target = df[["dif\_avg\_value"]].iloc[(window\_size+ahead\_number):]

    target = np.array(target, dtype=np.float32)

    features = []

    for i in range(len(target)):

        features.append(features\_arr[i:window\_size+i])

    features = np.asarray(features)

    return features,target

window\_size = 15

ahead\_number = 1

x\_train,y\_train = make\_dataset\_for\_NN(df=train\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_val,y\_val = make\_dataset\_for\_NN(df=val\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_test,y\_test = make\_dataset\_for\_NN(df=test\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_train = x\_train.reshape(len(y\_train),window\_size)

x\_val = x\_val.reshape(len(y\_val),window\_size)

x\_test = x\_test.reshape(len(y\_test),window\_size)

from keras.models import Sequential

from keras.layers import Activation, Dense,Dropout

model = Sequential([

    Dense(128, activation='selu', input\_shape=(window\_size,)),

    Dropout(0.5),

    Dense(256, activation='selu'),

    Dense(256, activation='selu'),

    Dense(256, activation='selu'),

    Dropout(0.5),

    Dense(256, activation='selu'),

    Dense(256, activation='selu'),

    Dropout(0.5),

     Dense(256, activation='selu'),

   Dropout(0.5),

    Dense(1)

])

def compile\_and\_fit(model,x\_train,y\_train, x\_val,y\_val, batch\_size, num\_epochs):

    model.compile(optimizer='adam', loss='mae', metrics=['mse'])

    return model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val),

                    batch\_size=batch\_size, epochs=num\_epochs, verbose=1,

                    callbacks=

Окончание приложения B

         [tf.keras.callbacks.ModelCheckpoint('best\_models\_NN/', monitor='val\_loss', verbose=1,save\_best\_only=True, save\_weights\_only=False,

                            mode='min', save\_freq='epoch')])

batch\_size = 8

num\_epochs = 25

history =  compile\_and\_fit(model, x\_train, y\_train, x\_val, y\_val, batch\_size,num\_epochs)

np.round(model.evaluate(x\_val,y\_val),4)

np.round(model.evaluate(x\_test,y\_test),4)

from keras.models import load\_model

model = load\_model('best\_models\_NN/')

prediction = model.predict(x\_test)

val\_time = val\_df[:len(-y\_test)].drop(columns=["dif\_avg\_value","avg\_value"])

plt.figure(figsize=(10, 7))

plt.plot(val\_time,y\_test)

plt.plot(val\_time,prediction)

plt.title('prediction')

plt.ylabel('temperature')

plt.xlabel('date,time')

plt.legend(['real', 'prediction'], loc='lower left')

plt.xticks( rotation='vertical')

dif = np.array(prediction-y\_test)

plt.figure(figsize=(10, 7))

plt.plot(val\_time,dif)

plt.title('mistake plot')

plt.ylabel('diftemperature')

plt.xlabel('date,time',labelpad=10)

plt.xticks( rotation='vertical')

dif = np.abs(dif)

print(dif.mean())

Рисунок В.1 – Листинг программы для модели искусственной нейронной сети

**Приложение Г.**

**Модель LSTM**

from datetime import timedelta

import numpy as np

import tensorflow as tf

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_absolute\_error

from matplotlib import pyplot as plt

import plotly.express as px

from typing import List

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.graphics.tsaplots import plot\_pacf

import pandas as pd

data = pd.read\_csv('dataTnv.csv',index\_col=False)

dataTemp = pd.read\_csv('dataTnv.csv',index\_col=False, usecols=["avg\_value"])

datainter=dataTemp.fillna(data.interpolate())

dataParse\_time=pd.to\_datetime(data.btime, format="%Y-%m-%d %X+03")

df=pd.concat([dataParse\_time,datainter],axis=1)

df = df.iloc[::7, :]

df['dif\_avg\_value'] = np.append(0,(np.array(df['avg\_value'][1:]) - np.array(df['avg\_value'][:-1])))

df.loc[df.dif\_avg\_value > 0.3,'dif\_avg\_value'] = 0

df.loc[df.dif\_avg\_value < -0.3,'dif\_avg\_value'] = 0

train\_size = int(df.shape[0] \* 0.7)

val\_size = int(df.shape[0] \* 0.2)

test\_size = int(df.shape[0] \* 0.1)

train\_df = df.iloc[:train\_size]

val\_df = df.iloc[train\_size:-test\_size]

test\_df = df.iloc[train\_size+val\_size:]

df.loc[:train\_size, 'partition'] = 'train'

df.loc[train\_size:train\_size+val\_size, 'partition'] = 'validation'

df.loc[train\_size+val\_size:, 'partition'] = 'test'

fig, ax = plt.subplots(figsize=(11, 4))

train\_df['dif\_avg\_value'].plot(ax=ax, label='train')

val\_df['dif\_avg\_value'].plot(ax=ax, label='validation')

test\_df['dif\_avg\_value'].plot(ax=ax, label='test')

ax.legend()

def make\_dataset\_for\_LSTM(df, window\_size,ahead\_number, use\_scaler=False):

    features\_arr = df[["dif\_avg\_value"]].iloc[:-(ahead\_number)]

    features\_arr = np.array(features\_arr, dtype=np.float32)

    if use\_scaler:

      Продолжение приложения Г

  features\_arr = scaler.transform(features\_arr)

    target = df[["dif\_avg\_value"]].iloc[(window\_size+ahead\_number):]

    target = np.array(target, dtype=np.float32)

    features = []

    for i in range(len(target)):

        features.append(features\_arr[i:window\_size+i])

    features = np.asarray(features)

    return features,target

window\_size = 15

ahead\_number = 1

x\_train,y\_train = make\_dataset\_for\_LSTM(df=train\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_val,y\_val = make\_dataset\_for\_LSTM(df=val\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_test,y\_test = make\_dataset\_for\_LSTM(df=test\_df, window\_size=window\_size, ahead\_number=ahead\_number, use\_scaler=False)

x\_train = x\_train.reshape(len(y\_train),window\_size)

x\_val = x\_val.reshape(len(y\_val),window\_size)

x\_test = x\_test.reshape(len(y\_test),window\_size)

lstm\_model = tf.keras.models.Sequential([

    tf.keras.Input((window\_size,1)),

    #tf.keras.layers.LSTM(32, return\_sequences=True),

    tf.keras.layers.LSTM(32, return\_sequences=False),

    tf.keras.layers.Dense(1)

])

def compile\_and\_fit(model,x\_train,y\_train, x\_val,y\_val, batch\_size, num\_epochs):

    model.compile(loss=tf.losses.MeanSquaredError(), optimizer=tf.optimizers.Adam(), metrics=[tf.metrics.MeanAbsoluteError()])

    return model.fit(x=x\_train, y=y\_train, batch\_size=batch\_size, epochs=num\_epochs, verbose=1,

                        callbacks =

                        tf.keras.callbacks.ModelCheckpoint(

                        'best\_models\_lstm/', monitor='val\_loss', verbose=1,

                        save\_best\_only=True, save\_weights\_only=False,

                        mode='min', save\_freq='epoch'), validation\_data=(x\_val, y\_val), shuffle=True,

                       )

batch\_size = 8

num\_epochs = 25

history =  compile\_and\_fit(lstm\_model, x\_train, y\_train, x\_val, y\_val, batch\_size, num\_epochs)

Окончание приложения Г

plt.figure(figsize=(10, 7))

plt.plot(history.history['val\_loss'])

plt.plot(history.history['loss'])

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['val', 'train'], loc='upper left')

plt.figure(figsize=(10, 7))

plt.plot(history.history['val\_mean\_absolute\_error'])

plt.plot(history.history['mean\_absolute\_error'])

plt.ylabel('mae')

plt.xlabel('epoch')

plt.legend(['val', 'train'], loc='upper left')

np.round(lstm\_model.evaluate(x\_val,y\_val),4)

np.round(lstm\_model.evaluate(x\_test,y\_test),4)

from keras.models import load\_model

lstm\_model = load\_model('best\_models\_lstm/')

prediction = lstm\_model.predict(x\_test)

val\_time = val\_df[:len(-y\_test)].drop(columns=["dif\_avg\_value",'avg\_value'])

plt.figure(figsize=(10, 7))

plt.plot(val\_time,y\_test)

plt.plot(val\_time,prediction)

plt.title('prediction')

plt.ylabel('temperature')

plt.xlabel('date,time')

plt.legend(['real', 'prediction'], loc='lower left')

plt.xticks( rotation='vertical')

dif = np.array(prediction-y\_test)

plt.figure(figsize=(10, 7))

plt.plot(dif)

plt.ylim(0, 2)

plt.title('mistake plot')

plt.ylabel('MAE on sample')

plt.xlabel('sample number',labelpad=10)

dif = np.abs(dif)

print(dif.mean(

Рисунок Г.1. – Листинг программы для модели LSTM