МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное
автономное образовательное
учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

«Системы искусственного интеллекта»

Направление подготовки 09.03.04 «Программная инженерия»
Направленность (профиль) «Разработка и сопровождение программного обеспечения»
Квалификация выпускника: бакалавр

Ставрополь

2026

Введение

Цель изучения дисциплины «Системы искусственного интеллекта» является формирование у студентов теоретических знаний и практических навыков в области разработки и применения систем искусственного интеллекта, включая методы машинного обучения, обработки естественного языка, компьютерного зрения, а также интеграцию интеллектуальных компонентов в программные продукты различного назначения.

Задачами обучения являются:

- изучение основных paradigms и подходов в области искусственного интеллекта: символьный ИИ, машинное обучение, глубокое обучение;
- освоение методов предобработки данных, выбора признаков и оценки качества моделей машинного обучения;
- формирование навыков реализации алгоритмов машинного обучения на современных языках программирования (Python) с использованием специализированных библиотек (scikit-learn, TensorFlow, PyTorch);
- изучение архитектур нейронных сетей и методов их обучения для решения задач классификации, регрессии, кластеризации;
- освоение методов обработки естественного языка (NLP) и компьютерного зрения (CV) для создания интеллектуальных приложений;
- формирование навыков интеграции готовых моделей ИИ в программные продукты (веб-приложения, мобильные приложения, desktop-приложения);
- изучение этических и правовых аспектов применения систем искусственного интеллекта.

Лабораторная работа № 1

Настройка среды разработки. Введение в Python для Data Science

Цель: Подготовить инструментарий для реализации алгоритмов машинного обучения.

Теоретическое обоснование

Экосистема Data Science в Python. Успешная работа с данными и машинным обучением невозможна без правильно организованного инструментария. **Anaconda** — дистрибутив Python, включающий более 1500 научных пакетов и менеджер окружений conda. Виртуальные окружения изолируют зависимости проектов, предотвращая конфликты версий библиотек. **Jupyter Notebook** — интерактивная веб-среда, где код, визуализация и пояснительный текст (Markdown) могут сосуществовать в одной ячейке. Это идеальный формат для исследовательского анализа (EDA) и воспроизводимых экспериментов. **VS Code** — альтернатива с мощной поддержкой Python и встроенным просмотрщиком Jupyter.

NumPy — фундамент всех численных вычислений. Его главный объект — ndarray (N-мерный массив), который обеспечивает:

- Векторизованные операции (без явных циклов, на языке C) → высокая производительность.
- Единый тип данных (dtype) для всех элементов → экономия памяти.
- Широкий набор функций: линейная алгебра, статистика, генерация случайных чисел.

Pandas построен поверх NumPy и предоставляет табличные структуры:

- **Series** — одномерный массив с метками (индексом).
- **DataFrame** — двумерная таблица с разнотипными столбцами. Основные возможности: чтение данных из файлов (CSV, Excel, JSON, SQL), фильтрация, группировка (groupby), объединение (merge), работа с пропусками. Pandas стал стандартом для манипуляции табличными данными.

Визуализация: Matplotlib — низкоуровневая библиотека, позволяющая создавать любые типы графиков с тонкой настройкой каждого элемента. **Seaborn** — высокоуровневая надстройка, упрощающая построение статистических графиков (pairplot, heatmap, boxplot, violinplot) и использующая приятные темы оформления по умолчанию.

Цель работы — освоить базовый инструментарий и создать первый ноутбук, демонстрирующий полный цикл: загрузка → описание → визуализация → простые вычисления

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Установить Anaconda, создать виртуальное окружение ds_lab1 с Python 3.9+. 2. В Jupyter Notebook импортировать NumPy как np. Создать массив 5×5 случайных чисел от 0 до 1. Вычислить среднее и стандартное отклонение. 3. Создать Pandas DataFrame из словаря: {'Имя': ['Анна','Борис','Виктор'], 'Возраст': [23,45,31], 'Город': ['Москва','СПб','Казань']}. 4. Добавить столбец Возраст_кв = возраст в квадрате. 5. Загрузить датасет tips из Seaborn (sns.load_dataset('tips')). 6. Вывести первые 5 строк и общую информацию (info()). 7. Построить гистограмму столбца total_bill (Matplotlib, 20 бинов, оранжевый цвет, заголовок, подписи осей). 8. Построить scatter-график total_bill vs tip с раскраской по полу (hue='sex'). 9. Найти средний счёт по дням недели (groupby). 10. Сохранить ноутбук в HTML и график scatter в PNG.
2	1. Установить VSCode, расширение Python. Создать виртуальное окружение через python -m venv ds_env. 2. В Jupyter (или в .ipynb внутри VSCode) импортировать

	NumPy, создать массив 20 случайных целых от 10 до 50. Найти min, max, median. 3. Загрузить датасет iris (например, sns.load_dataset('iris')). 4. Вывести размерность и типы столбцов. 5. Вывести описательную статистику describe(). 6. Построить boxplot для sepal_length. 7. Построить pairplot для всех числовых признаков, раскрасить по species. 8. Сгруппировать по species и вывести средние значения. 9. Создать признак sepal_area = sepal_length * sepal_width. 10. Сохранить pairplot в файл pairplot_iris.png (dpi=150).
3	1. Установить PyCharm, настроить интерпретатор с виртуальным окружением. 2. Создать массив NumPy 100 равномерных чисел от 0 до 10, reshape в 10×10. 3. Создать DataFrame из Excel (5 строк, столбцы: «Товар», «Цена», «Кол-во»). 4. Загрузить датасет mpg из Seaborn. Вывести строки с 10 по 20 (.iloc). 5. Построить гистограмму horsepower (15 бинов, зелёный). 6. Построить jointplot (mpg vs weight). 7. Вычислить корреляционную матрицу числовых признаков. 8. Создать столбец power_to_weight = horsepower / weight. 9. Отсортировать по mpg по убыванию, показать первые 5. 10. Сохранить DataFrame в CSV без индекса.
4	1. Использовать Google Colab, установить библиотеки. 2. NumPy: массив 30 нормальных случайных чисел (μ=0, σ=1). Найти 10-й и 20-й процентили. 3. Загрузить датасет diamonds из Seaborn. Вывести количество строк и столбцов, пропуски. 4. Построить heatmap корреляции числовых признаков. 5. Построить гистограмму price с 30 бинами и линией среднего. 6. Сгруппировать по cut, вывести среднюю цену и

	средний карат. 7. Отфильтровать алмазы с ценой >5000 и каратом >1.0. 8. Сохранить отфильтрованный DataFrame в Excel. 9. Построить boxplot цены по огранке. 10. Вывести максимальную и минимальную цену для каждого типа огранки.
5	1. Установить Spyder (из Anaconda). 2. NumPy: массив 3×4×5 из единиц, заменить каждый элемент на его квадрат. 3. Загрузить flights из Seaborn. Построить сводную таблицу (pivot) «пассажиры» по годам и месяцам. 4. Построить линейный график средних пассажиров по годам. 5. Создать столбец year_month как строку. 6. Построить heatmap числа пассажиров (год – вертикаль, месяц – горизонталь). 7. Найти месяц с максимальным средним числом пассажиров. 8. Отсортировать датасет по убыванию пассажиров, топ-10. 9. Сохранить heatmap в PNG (300 dpi). 10. Вывести статистику по годам (среднее, медиана, максимум).
6	1. Настроить VS Code с Jupyter. 2. NumPy: массив 50 случайных целых от 0 до 100. Найти моду с помощью np.bincount. 3. Загрузить planets из Seaborn. Вывести уникальные значения method. 4. Построить гистограмму массы планет для метода «Radial Velocity». 5. Построить scatter: масса vs orbital period, цвет по методу. 6. Сгруппировать по методу, вывести среднюю массу и медианный период. 7. Найти планету с максимальной массой. 8. Создать столбец mass_class: <0.1 → «малая», иначе «большая». 9. Построить столбчатую диаграмму количества планет по методам. 10. Сохранить график столбчатой диаграммы в SVG.

7	1. Установка через командную строку: pip install numpy pandas matplotlib seaborn jupyter. 2. NumPy: создать массив 8×8 случайных чисел, нормализовать (вычесть среднее, разделить на std). 3. Загрузить titanic из Seaborn. Вывести процент выживших по классу (pclass). 4. Построить countplot выживших по полу. 5. Построить гистограмму возраста с разделением на выживших/погибших (использовать hue). 6. Создать новый признак family_size = sibsp + parch. 7. Сгруппировать по embarked, вывести среднюю стоимость билета. 8. Отфильтровать пассажиров с возрастом > 60. 9. Построить boxplot стоимости билета по классам. 10. Сохранить DataFrame с новым признаком в CSV.
8	1. Работа в JupyterLab (в составе Anaconda). 2. NumPy: создать массив 1000 случайных чисел по закону Пуассона (λ=5). Построить гистограмму. 3. Загрузить датасет exercise из Seaborn. Вывести информацию о пропусках. 4. Построить violinplot пульса (pulse) по типу упражнения (kind). 5. Построить scatterplot времени (time) vs пульса с цветом по диете. 6. Сгруппировать по kind и diet, вывести средний пульс. 7. Создать столбец pulse_diff = разница между максимальным и минимальным пульсом в группе. 8. Отфильтровать строки с пульсом > 100. 9. Построить линейный график среднего пульса по времени для каждой диеты. 10. Сохранить все графики в один PDF-файл.
9	1. Установить Miniconda, создать окружение с Python 3.10. 2. NumPy: создать два массива 5×3 со случайными числами, вычислить их поэлементное произведение. 3. Загрузить датасет car_crashes из Seaborn. Вывести названия столбцов. 4. Построить гистограмму числа погибших (total). 5. Построить pairplot первых 4 признаков. 6. Вычислить корреляцию между speeding и alcohol.

	7. Создать новый признак danger_index = speeding + alcohol. 8. Отсортировать штаты по danger_index и вывести топ-5. 9. Построить barplot опасности по штатам (топ-10). 10. Сохранить DataFrame в JSON.
10	1. Использовать Deepnote или Kaggle Notebook. 2. NumPy: создать массив 10×10 с числами от 1 до 100, затем получить транспонированный массив. 3. Загрузить датасет penguins из Seaborn. Удалить строки с пропусками. 4. Построить boxplot длины клюва (bill_length_mm) по видам. 5. Построить scatterplot длины и глубины клюва, цвет по полу. 6. Сгруппировать по виду и полу, вывести среднюю длину клюва. 7. Создать столбец bill_ratio = bill_length / bill_depth. 8. Отфильтровать пингвинов с body_mass_g > 4000. 9. Построить гистограмму массы тела с разбивкой по видам (с наложением). 10. Сохранить ноутбук в PDF через jupyter nbconvert.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Чем отличается Anaconda от стандартной установки Python? В каких случаях предпочтительнее использовать Miniconda?
2. Что такое Jupyter Notebook? Перечислите основные режимы ячеек (Markdown, Code, Raw) и их назначение.
3. В чём главное отличие массива NumPy от списка Python (производительность, типы данных, методы)?
4. Что такое Pandas Series и DataFrame? Как они связаны с массивами NumPy?
5. Как загрузить CSV-файл с разделителем «;» и пропущенными значениями «NA»? Приведите код.
6. Что выведет df.info()? A df.describe(include='object')?
7. Какая функция в Matplotlib строит гистограмму? Назовите 4 основных параметра.
8. В чём преимущество Seaborn перед Matplotlib для статистической визуализации? Приведите два примера.
9. Как добавить новый столбец в DataFrame на основе существующих? Напишите пример с вычислением отношения двух столбцов.
10. Как сохранить график в файл с высоким разрешением (300 dpi) и прозрачным фоном? Как сохранить весь ноутбук в HTML?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов)	40
Качество кода. Читаемость, наличие комментариев, использование осмысленных имён переменных, воспроизводимость.	10
Визуализации. Наличие заголовков, подписей осей, легенд, выбор адекватного типа графика, чистота изображения.	10
Ответы на контрольные вопросы (10 вопросов)	10

Структура и оформление отчёта. Наличие всех разделов (титул, цель, теория, ход, выводы, вопросы), логичность, отсутствие лишних выводов.	15
Выводы и интерпретация. Анализ полученных результатов, сравнение с ожиданиями, указание на возможные ошибки.	10
Сохранение результатов. Наличие HTML-экспорта, сохранённых графиков (PNG/PDF) в соответствии с заданием.	5

Лабораторная работа № 2

Загрузка и первичный анализ данных

Цель: Освоить методы загрузки данных из различных источников и проведения первичного анализа.

Теоретическое обоснование

Источники данных. В реальных проектах данные редко лежат в одном CSV-файле. Pandas поддерживает:

- **CSV** — pd.read_csv() с параметрами: sep (разделитель), encoding, na_values (какие строки считать пропусками), parse_dates, chunksize (чтение по частям).
- **Excel** — pd.read_excel() с указанием листа (sheet_name) и диапазона (usecols).
- **JSON** — pd.read_json() или pd.json_normalize() для вложенных структур.
- **API** — через библиотеку requests получаем JSON, затем преобразуем в DataFrame.
- **SQL** — pd.read_sql() с SQL-запросом и подключением к БД.

Первичный анализ (EDA). После загрузки необходимо понять структуру:

- df.head(), df.tail(), df.sample() — просмотр строк.

- df.info() — количество строк, типы столбцов, память, число ненулевых значений (косвенно пропуски).
- df.describe() — основные статистики для числовых столбцов: среднее, стандартное отклонение, минимум, максимум, квартили.
- df.describe(include='object') — для категориальных: уникальные значения, топ-мода.
- df.isnull().sum() — точное число пропусков.
- df.duplicated().sum() — дубликаты.

Группировка и агрегация. groupby() разбивает данные по значениям одного или нескольких столбцов, затем к каждой группе применяется агрегирующая функция (mean, sum, count, agg с несколькими функциями). pivot_table() создаёт сводную таблицу, аналогичную Excel.

Выбросы (outliers) можно предварительно выявить с помощью boxplot или статистических критериев (IQR, Z-score). Этот этап готовит данные к очистке.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет titanic из библиотеки Seaborn (sns.load_dataset('titanic')). 2. Вывести количество строк и столбцов, типы данных. 3. Используя info(), определить столбцы с пропущенными значениями и количество пропусков в каждом. 4. Вывести статистику числовых признаков (describe()). 5. Для категориальных признаков sex и embarked вывести частоты (value_counts()). 6. Найти выбросы в признаке age методом IQR: рассчитать Q1, Q3, IQR, границы. 7. Построить boxplot для fare с разделением по классу (pclass). 8. Сгруппировать данные по pclass и вычислить средний

	возраст и медианную стоимость билета. 9. Создать сводную таблицу (pivot table): средний fare по sex и survived. 10. Сохранить исходный DataFrame в CSV-файл titanic_initial.csv без индекса.
2	1. Загрузить датасет California housing (через sklearn.datasets.fetch_california_housing) в DataFrame. 2. Вывести первые 3 и последние 3 строки. 3. Определить наличие дубликатов (duplicated().sum()). 4. Вывести корреляционную матрицу всех числовых признаков. 5. Найти максимальное и минимальное значение MedHouseVal. 6. Определить, сколько районов имеют MedInc > 5. 7. Построить гистограмму HouseAge (20 бинов). 8. Сгруппировать по AveBedrms (округлив до целого) и вывести среднюю стоимость. 9. Найти 5% самых дорогих районов (по MedHouseVal). 10. Сохранить статистику (min, max, mean, std) по каждому столбцу в JSON-файл.
3	1. Загрузить датасет wine из sklearn (load_wine), преобразовать в DataFrame с именами признаков. 2. Вывести имена признаков и целевой переменной. 3. Проверить баланс классов (value_counts() целевой переменной). 4. Вычислить среднее и стандартное отклонение для каждого признака. 5. Построить pairplot для первых четырёх признаков, раскрасив по классам. 6. Найти признаки с наибольшей положительной корреляцией (с помощью corr() и .unstack()). 7. Проверить наличие пропусков (должно быть 0, но показать метод). 8. Создать новый признак – сумму всех химических компонентов (по строкам). 9. Разделить данные на две части: объекты класса 1 и объекты классов 2+3. 10. Экспортировать обе части в один Excel-файл на разные листы.
4	1. Загрузить CSV-файл с кодировкой cp1251 (скачать

	датасет «Отток клиентов» с Kaggle или синтетический). 2. Вывести размерность, типы, количество уникальных значений в категориальных столбцах. 3. Вывести процент пропусков в каждом столбце (в виде таблицы). 4. Построить гистограммы всех числовых признаков (2×2 subplot). 5. Найти выбросы в признаке Tenure (метод z-score: >3 или <-3). 6. Сгруппировать по Churn и вывести средние значения всех числовых признаков. 7. Построить boxplot MonthlyCharges в разрезе Churn. 8. Создать сводную таблицу: средний TotalCharges по PaymentMethod и Churn. 9. Отфильтровать клиентов с MonthlyCharges > 100 и Tenure < 12. 10. Сохранить отфильтрованные данные в CSV.
5	1. Загрузить JSON-файл через API (например, данные о погоде за последние 7 дней с open-meteo или готовый файл из репозитория). 2. Нормализовать JSON в плоский DataFrame с помощью pd.json_normalize(). 3. Вывести информацию о типах и пропусках. 4. Преобразовать столбец с датой в тип datetime. 5. Вычислить среднюю температуру, влажность, скорость ветра. 6. Построить линейный график температуры по датам. 7. Найти день с максимальной и минимальной температурой. 8. Сгруппировать по дням недели и вычислить среднюю температуру. 9. Определить наличие выбросов в скорости ветра (визуально через boxplot). 10. Сохранить обработанные данные в Parquet-файл.
6	1. Загрузить Excel-файл с несколькими листами (например, sales.xlsx с листами 2022, 2023). 2. Загрузить каждый лист в отдельный DataFrame, вывести названия листов. 3. Объединить данные из двух листов вертикально

	(pd.concat). 4. Вывести основную статистику по столбцу Revenue. 5. Проверить наличие дубликатов по TransactionID. 6. Построить гистограмму Revenue с логарифмической шкалой. 7. Сгруппировать по ProductCategory и вывести сумму и средний Revenue. 8. Найти топ-5 продуктов по выручке. 9. Создать новый столбец Quarter из даты. 10. Сохранить объединённый DataFrame в CSV.
7	1. Загрузить датасет adult (перепись населения) из открытого источника (или fetch_openml). 2. Вывести количество строк, столбцов, названия столбцов. 3. Определить, сколько категориальных признаков (тип object). 4. Для каждого категориального признака вывести топ-5 частот. 5. Вычислить среднее значение hours-per-week для разных education. 6. Построить boxplot age по income (<=50K, >50K). 7. Найти выбросы в capital-gain (люди с очень большим доходом). 8. Сгруппировать по workclass и income, вывести количество людей. 9. Создать сводную таблицу: средний age по marital-status и sex. 10. Сохранить DataFrame в CSV с изменёнными названиями столбцов (латиница → транслит).
8	1. Загрузить датасет credit_card_fraud (с дисбалансом классов) – синтетический или из Kaggle. 2. Вывести распределение классов (Class – 0/1). 3. Вычислить среднее, медиану, максимум для Amount отдельно для каждого класса. 4. Построить гистограмму Amount с разбивкой по классам (на одном графике с прозрачностью). 5. Определить количество пропусков в каждом столбце. 6. Найти корреляцию Amount с Time (интерпретировать). 7. Построить ящик с усами для Amount (масштаб

	логарифмический). 8. Сгруппировать по часу (из Time) и вычислить среднюю сумму транзакции. 9. Отфильтровать мошеннические транзакции с Amount > 1000. 10. Сохранить статистику по каждому признаку (среднее, медиана, Q1, Q3) в текстовый файл.
9	1. Загрузить данные из SQLite (создать БД с таблицей employees, заполнить 20 строками). 2. Выполнить SQL-запрос через pd.read_sql для выборки сотрудников с зарплатой > 50000. 3. Вывести первые 5 строк результата. 4. Вычислить среднюю зарплату по отделам (используя группировку после загрузки). 5. Найти максимальную и минимальную зарплату. 6. Определить наличие пропусков в столбце Bonus. 7. Построить гистограмму Salary. 8. Создать новый столбец Salary_category (низкая/средняя/высокая) на основе квантилей. 9. Сгруппировать по категории зарплаты и отделу, вывести количество сотрудников. 10. Сохранить результат группировки в Excel.
10	1. Загрузить текстовый файл с фиксированной шириной столбцов (.txt или .fwf) – например, данные переписи. 2. Использовать pd.read_fwf с заданными спецификациями ширин. 3. Вывести размерность и типы. 4. Переименовать столбцы в осмысленные имена. 5. Вывести статистику для числовых столбцов. 6. Определить количество пропусков в каждом столбце (если есть). 7. Построить scatter plot двух выбранных признаков. 8. Сгруппировать по одному категориальному признаку, вывести средние. 9. Найти выбросы в одном из числовых признаков визуально. 10. Сохранить очищенный DataFrame в CSV с разделителем «;».

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Какие параметры pd.read_csv() помогают при проблемах с кодировкой (например, cp1251), нестандартным разделителем, пропущенными значениями?
2. Как загрузить данные из Excel, указав конкретный лист и диапазон ячеек (например, sheet_name='Лист1', skiprows=2, usecols='A:C')?
3. Что такое API-ключ и как его передать в GET-запросе с помощью библиотеки requests для получения данных в JSON?
4. В чём разница между методами df.head(), df.tail(), df.sample(5)?
5. Что показывает df.shape, df.ndim, df.size?
6. Как определить количество пропущенных значений в каждом столбце одной командой? Как отобразить процент пропусков?
7. Какой метод Pandas выводит основные статистики для всех числовых столбцов? Что означают 25%, 50%, 75%?
8. Как найти выбросы с помощью межквартильного размаха (IQR)? Приведите формулу верхней и нижней границы.
9. Для чего нужны методы groupby() и agg()? Приведите пример вычисления одновременно среднего и максимума по группе.
10. Что такое сводная таблица (pivot_table) и чем она отличается от groupby?

Как задать значения, строки, столбцы и агрегирующую функцию?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов)	40
Качество кода. Читаемость, наличие комментариев, использование осмысленных имён переменных, воспроизводимость.	10
Визуализации. Наличие подписей, заголовков, легенд, выбор адекватного типа графика.	10
Ответы на контрольные вопросы (10 вопросов)	10
Структура и оформление отчёта. Наличие всех разделов (титул, цель, теория, ход, выводы, вопросы), логичность, отсутствие лишних выводов.	10
Выводы и интерпретация. Анализ полученных статистик, описание выявленных пропусков и выбросов, интерпретация группировок.	10
Дополнительные требования (сохранение файлов, работа с разными источниками). Правильное сохранение в указанных форматах, загрузка из JSON/Excel/API в соответствии с вариантом.	10

Лабораторная работа № 3

Визуализация данных с Matplotlib и Seaborn

Цель и содержание работы: Освоить инструменты визуализации для исследовательского анализа данных.

Теоретическое обоснование

Роль визуализации. Визуализация — ключевой инструмент EDA. Она

позволяет обнаружить: распределение переменных (гистограмма, ящик с усами), взаимосвязи (диаграмма рассеяния, pairplot), выбросы (boxplot), корреляции (heatmap).

Основные типы графиков:

- Гистограмма (plt.hist, sns.histplot) — показывает частоту попадания значений в интервалы. Параметры: bins (число интервалов), density (нормировка), cumulative.
- Boxplot (ящик с усами) — отображает медиану (центральная линия), первый и третий квартили (границы ящика), «усы» (обычно 1.5·IQR) и выбросы (точки за усами). Позволяет быстро сравнить распределения по категориям.
- Scatter plot (plt.scatter, sns.scatterplot) — точки на плоскости для двух числовых признаков. Параметр hue раскрашивает точки по категориальному признаку, size — размер по числовому.
- Pairplot (sns.pairplot) — матрица всех попарных scatter-графиков + гистограммы на диагонали. Неоценим при исследовании многомерных данных.
- Heatmap (sns.heatmap) — цветовая матрица, чаще всего для корреляционной матрицы (df.corr()). Параметр annot=True показывает числовые значения.

Настройка стилей. Seaborn имеет встроенные темы: darkgrid, whitegrid, dark, white, ticks. Палитры: deep, muted, pastel, bright, dark, colorblind. Matplotlib позволяет управлять размером фигуры (figsize), шрифтами, цветами через plt.rcParams.

Сохранение графиков: plt.savefig('name.png', dpi=300, bbox_inches='tight', transparent=True).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	**Задание**

1	1. Загрузить датасет iris из Seaborn. 2. Построить гистограмму (hist) для признака sepal_length (20 бинов, синий цвет, прозрачность 0.7). 3. Построить boxplot для sepal_width по видам (species). 4. Построить scatter plot: petal_length vs petal_width, раскрасить точки по видам (разные цвета). 5. Построить pairplot для всех числовых признаков, раскрасив по species, с диагональю в виде гистограмм. 6. Построить тепловую карту (heatmap) корреляции числовых признаков (с подписями значений в ячейках). 7. Настроить стиль Seaborn на "whitegrid", размер фигуры для heatmap – 8×6. 8. Добавить на график scatter заголовок «Зависимость длины лепестка от ширины», подписи осей. 9. Сохранить heatmap в PNG (dpi=200). 10. Создать один общий рисунок с subplot 2×2, содержащий: гистограмму, boxplot, scatter, heatmap (уменьшенную).
2	1. Загрузить датасет tips из Seaborn. 2. Построить гистограмму total_bill с 30 бинами, добавить линию среднего значения (красную). 3. Построить violinplot (tip по дням недели day). 4. Построить scatter plot: total_bill vs tip с размером точек по size, цветом по sex. 5. Построить jointplot (total_bill, tip) с типом 'hex'. 6. Построить countplot количества чаевых по time (обед/ужин). 7. Построить boxplot total_bill по day с наложением точек (stripplot). 8. Настроить палитру цветов на 'coolwarm'. 9. Добавить легенду к scatter plot (объяснить цвета и размеры). 10. Сохранить jointplot в PDF.
3	1. Загрузить датасет diamonds из Seaborn. 2. Построить гистограмму price с логарифмической шкалой по оси Y. 3. Построить boxplot price по огранке (cut). 4. Построить scatter plot: carat vs price с прозрачностью 0.3 и цветом по color.

	5. Построить pairplot для признаков carat, depth, table, price, раскрасив по cut. 6. Построить тепловую карту корреляции всех числовых признаков. 7. Построить violinplot price по цвету (color). 8. Создать subplot 1×2: boxplot цены по огранке и boxplot цены по цвету. 9. Добавить подписи осей, заголовки, настроить размер шрифта. 10. Сохранить subplot в SVG.
4	1. Загрузить датасет mpg из Seaborn. 2. Построить гистограмму mpg (городской цикл) с 15 бинами, зелёный цвет. 3. Построить boxplot horsepower по количеству цилиндров (cylinders). 4. Построить scatter plot: weight vs mpg, цвет по origin, размер по horsepower. 5. Построить pairplot только для признаков mpg, weight, horsepower, acceleration. 6. Построить регрессионный график (lmplot) зависимости mpg от weight с разбивкой по origin. 7. Построить тепловую карту корреляции (все признаки). 8. Настроить стиль "darkgrid", изменить размер фигуры на 10×6. 9. Добавить сетку на scatter plot. 10. Сохранить lmplot в PNG с разрешением 150 dpi.
5	1. Загрузить датасет flights из Seaborn. 2. Построить линейный график (lineplot) числа пассажиров по годам (усреднив по месяцам). 3. Построить тепловую карту числа пассажиров (год – ось Y, месяц – ось X). 4. Построить столбчатую диаграмму (barplot) среднего числа пассажиров по месяцам. 5. Построить boxplot числа пассажиров по годам. 6. Построить swarmplot (или stripplot) пассажиров по месяцам. 7. Создать subplot 2×1: линейный график и тепловая карта. 8. Настроить цветовую палитру тепловой карты на

	'Blues'. 9. Добавить заголовок «Анализ авиапассажиров» для всего рисунка (fig.suptitle). 10. Сохранить subplot в PDF.
6	1. Загрузить датасет penguins из Seaborn. 2. Построить гистограмму длины клюва (bill_length_mm) с разделением по видам (hue). 3. Построить boxplot глубины клюва (bill_depth_mm) по видам и полу. 4. Построить scatter plot: длина vs глубина клюва, цвет по виду, форма по полу. 5. Построить pairplot с диагональю в виде KDE, раскрасив по виду. 6. Построить violinplot массы тела (body_mass_g) по видам. 7. Построить тепловую карту корреляции (числовые признаки). 8. Настроить палитру на 'Set2'. 9. Сохранить pairplot в файл с высоким разрешением (300 dpi). 10. Добавить подписи осей и легенду ко всем графикам.
7	1. Загрузить датасет titanic из Seaborn. 2. Построить countplot выживших (survived) с разделением по полу. 3. Построить гистограмму возраста (age) с 30 бинами, раскрасив по выжившим. 4. Построить boxplot стоимости билета (fare) по классу (pclass) с логарифмической шкалой. 5. Построить violinplot возраста по классу и выжившим (две категории). 6. Построить тепловую карту корреляции числовых признаков (age, fare, sibsp, parch). 7. Построить barplot средней стоимости билета по порту посадки (embarked). 8. Создать subplot 2×2: countplot, гистограмма, boxplot, barplot. 9. Настроить общий стиль "white", размер шрифта 12. 10. Сохранить subplot в PNG.
8	1. Загрузить датасет exercise из Seaborn. 2. Построить гистограмму пульса (pulse) с разделением

	по типу упражнения (kind). 3. Построить boxplot пульса по времени (time). 4. Построить scatter plot: время vs пульс, цвет по диете (diet). 5. Построить линейный график среднего пульса по времени для каждой диеты. 6. Построить violinplot пульса по типу упражнения и диете (двойное разбиение). 7. Построить тепловую карту корреляции (пульс, возраст, рост, вес). 8. Настроить стиль "ticks", размер фигуры 12×8. 9. Добавить заголовки ко всем графикам. 10. Сохранить violinplot в SVG.
9	1. Загрузить датасет car_crashes из Seaborn. 2. Построить гистограмму числа погибших (total). 3. Построить barplot штрафов за превышение (speeding) по штатам (топ-10). 4. Построить scatter plot: алкоголь vs превышение, размер по погибшим. 5. Построить pairplot для всех признаков (6 признаков). 6. Построить тепловую карту корреляции. 7. Построить boxplot для признака alcohol. 8. Создать subplot 2×2: гистограмма, barplot, scatter, boxplot. 9. Настроить палитру 'RdBu'. 10. Сохранить subplot в PDF с высоким качеством.
10	1. Загрузить синтетические данные: сгенерировать 200 точек с помощью make_blobs (3 центра). 2. Построить scatter plot полученных точек, раскрасив по кластерам. 3. Построить гистограмму по первому признаку. 4. Построить boxplot второго признака по кластерам. 5. Построить jointplot (первый vs второй признак) с типом 'kde'. 6. Построить тепловую карту корреляции. 7. Построить линейный график (отсортировать точки по первому признаку, второй признак – линия). 8. Настроить стиль "seaborn-v0_8-whitegrid". 9. Добавить заголовки, подписи, легенду. 10. Сохранить jointplot в PNG.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём разница между plt.plot() и plt.scatter()? Для каких данных каждый из них предпочтителен?
2. Как построить несколько графиков в одной фигуре с помощью subplot()? Приведите пример создания сетки 2×3.
3. Что такое «ящик с усами» (boxplot)? Какие элементы на нём отображаются (медиана, квартили, выбросы)?
4. Как интерпретировать тепловую карту (heatmap) корреляционной матрицы? Какие значения считаются сильной положительной/отрицательной корреляцией?
5. Чем отличается pairplot от scatter_matrix? Какую информацию можно извлечь из pairplot?
6. Как изменить цветовую палитру в Seaborn? Назовите 3 встроенные палитры.
7. Как добавить заголовок, подписи осей и легенду на график Matplotlib? Приведите код.
8. Что такое jointplot в Seaborn? Какие типы диаграмм можно отобразить на диагонали и в основной области?

9. Как сохранить график в файл с прозрачным фоном и разрешением 300 dpi?
10. В чём преимущество использования violinplot перед boxplot? Когда его следует применять?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов)	40
Качество визуализаций. Наличие заголовков, подписей осей, легенд, выбор адекватного типа графика, читаемость.	20
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, чистота кода, логичность расположения графиков и комментариев.	10
Выводы и интерпретация. Глубина анализа графиков, выявленные закономерности, сравнение разных типов визуализаций.	10
Сохранение графиков. Все требуемые графики сохранены в указанных форматах, файлы присутствуют, разрешение соответствует.	10

Лабораторная работа 4

Предобработка данных: очистка и обработка пропусков

Цель: Научиться обрабатывать пропущенные значения и выбросы.

Теоретическое обоснование

Природа пропусков. Пропуски (NaN, None) возникают по разным причинам: ошибки сбора, неполные записи, некорректные значения. По

механизму возникновения делятся на:

- MCAR (Missing Completely at Random) — отсутствие не зависит от данных.
- MAR (Missing at Random) — зависит от наблюдаемых признаков.
- MNAR (Missing Not at Random) — зависит от самой пропущенной величины.

Методы обработки пропусков:

1. **Удаление строк** (dropna(axis=0)) — эффективно, если доля пропусков мала (<5%). Удаление столбцов (axis=1) — при >50% пропусков.
2. **Заполнение константой** (fillna(0) или fillna('Unknown')) — для категориальных признаков.
3. **Заполнение статистиками** — средним (для нормальных распределений), медианой (устойчива к выбросам), модой (для категориальных). Часто используют групповое заполнение (например, средний возраст по классу).
4. **Интерполяция** (interpolate()) — для временных рядов (линейная, полиномиальная, сплайновая).
5. **Предсказание пропусков** — KNNImputer (заполняет средним по k ближайшим соседям) или IterativeImputer (модель регрессии для каждого признака)

Выбросы (outliers). Методы выявления:

- IQR (межквартильный размах): нижняя граница = Q1 - 1.5·IQR, верхняя = Q3 + 1.5·IQR.
- Z-score: $|z| > 3$, где $z = (x - \mu)/\sigma$.
- Визуальные: boxplot, scatter plot.

Обработка выбросов: удаление, каппинг (замена на границу), логарифмическое преобразование, winsorization.

Важно: любые преобразования, основанные на статистиках (среднее, медиана), должны вычисляться **только на обучающей выборке**, а затем применяться к тестовой, чтобы избежать утечки данных.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет titanic из Seaborn. 2. Определить столбцы с пропущенными значениями, вывести количество пропусков в каждом. 3. Удалить строки с пропусками в столбце embarked (т.к. их мало). 4. Заполнить пропуски в столбце age медианным значением возраста по классу (pclass). 5. Для столбца cabin удалить столбец целиком (более 70% пропусков). 6. Проверить, остались ли пропуски после обработки. 7. Выявить выбросы в столбце fare методом IQR (рассчитать границы). 8. Заменить выбросы в fare на верхнюю границу IQR (каппинг). 9. Построить boxplot fare до и после обработки (два графика рядом). 10. Сохранить очищенный DataFrame в CSV titanic_cleaned.csv.
2	1. Загрузить датасет mpg из Seaborn. 2. Вывести количество пропусков в каждом столбце. 3. Заполнить пропуски в horsepower средним значением по производителю (origin). 4. Для столбца name удалить его (не нужен для анализа). 5. Удалить строки, где weight имеет пропуски (если есть). 6. Проверить наличие выбросов в mpg с помощью z-score (порог 3). 7. Удалить строки с выбросами в mpg. 8. Заполнить пропуски в acceleration медианой (если остались).

	9. Построить гистограммы horsepower до и после заполнения. 10. Сохранить результат в Excel.
3	1. Загрузить датасет penguins из Seaborn. 2. Определить процент пропусков в каждом столбце. 3. Удалить строки, где sex имеет пропуски (удалить явно). 4. Заполнить пропуски в bill_length_mm средним значением по виду (species). 5. Заполнить пропуски в bill_depth_mm медианой по острову (island). 6. Заполнить пропуски в flipper_length_mm с помощью интерполяции (метод 'linear'). 7. Выявить выбросы в body_mass_g методом IQR, вывести их количество. 8. Заменить выбросы на медиану. 9. Построить scatter plot массы тела до и после обработки (с разными цветами). 10. Сохранить очищенный DataFrame в CSV.
4	1. Загрузить датасет diamonds из Seaborn (пропусков нет, поэтому искусственно добавить 5% случайных пропусков в столбце price и carat). 2. Вывести количество пропусков после добавления. 3. Удалить строки с пропусками в carat (если их < 5%). 4. Заполнить пропуски в price медианой цены для каждой огранки (cut). 5. Использовать KNNImputer (k=5) для восстановления пропусков в price (сравнить с медианой). 6. Оценить качество заполнения: MSE между восстановленными и исходными (до добавления пропусков). 7. Выявить выбросы в table методом IQR. 8. Удалить строки с выбросами в table. 9. Построить boxplot price до и после обработки. 10. Сохранить результат в Parquet.
5	1. Загрузить датасет wine из sklearn, преобразовать в DataFrame. Пропусков нет – добавить 10% пропусков в случайные столбцы. 2. Использовать SimpleImputer со стратегией 'most_frequent' для категориальных (если есть) или для

	числовых – 'mean'. 3. Для одного признака использовать заполнение константой (0). 4. Сравнить средние и дисперсии признаков до и после заполнения. 5. Выявить выбросы методом z-score в признаке alcohol. 6. Удалить строки с выбросами. 7. Построить гистограмму alcohol с наложением KDE до и после удаления выбросов. 8. Проверить, нет ли пропусков после всех операций. 9. Сохранить очищенный DataFrame в CSV. 10. Сделать вывод о влиянии заполнения на распределение.
6	1. Загрузить датасет flights из Seaborn (полный, без пропусков). Искусственно удалить 10% значений из столбца passengers. 2. Заполнить пропуски с помощью линейной интерполяции (interpolate(method='linear')). 3. Построить график исходного ряда и восстановленного (на одном поле). 4. Заполнить пропуски в том же столбце методом 'pad' (вперёд) и сравнить визуально. 5. Вычислить MAE между восстановленными и исходными (до удаления). 6. Выявить выбросы в passengers с помощью скользящего окна (3 сигмы). 7. Удалить выбросы (заменить на NaN и интерполировать). 8. Сохранить результат в Excel. 9. Построить boxplot исходных и обработанных данных. 10. Сделать вывод о лучшем методе интерполяции.
7	1. Загрузить датасет adult из openml (fetch_openml('adult', version=1)). 2. Вывести пропуски (в исходных данных обозначены как '?'). Заменить '?' на NaN. 3. Удалить строки с пропусками в workclass (если < 10%). 4. Заполнить пропуски в occupation модой по полу. 5. Заполнить пропуски в native-country константой 'United-States'.

	6. Для числового признака capital-gain заполнить пропуски (если есть) медианой. 7. Выявить выбросы в hours-per-week методом IQR, вывести их количество. 8. Заменить выбросы на медиану. 9. Построить violinplot hours-per-week до и после обработки. 10. Сохранить очищенный DataFrame в CSV.
8	1. Загрузить датасет credit_card_fraud (синтетический или малая выборка). Искусственно внести 3% пропусков в Amount и Time. 2. Заполнить пропуски в Amount с использованием IterativeImputer (прогнозирование по другим признакам). 3. Заполнить пропуски в Time медианой. 4. Сравнить распределение Amount до и после заполнения (гистограммы). 5. Выявить выбросы в Amount методом z-score, удалить строки с $\|z\|>3$. 6. Построить boxplot Amount после удаления выбросов. 7. Проверить, изменилось ли соотношение классов после удаления выбросов. 8. Сохранить результат в HDF5. 9. Сделать вывод о применимости IterativeImputer. 10. Визуализировать исходные и восстановленные значения Amount на scatter plot.
9	1. Загрузить датасет california_housing из sklearn. Пропусков нет – добавить 15% пропусков в MedInc и AveRooms. 2. Заполнить пропуски в MedInc медианой по соседям (группировка по HouseAge с округлением). 3. Заполнить пропуски в AveRooms с помощью SimpleImputer(strategy='median'). 4. Сравнить средние значения до и после заполнения. 5. Выявить выбросы в MedInc методом IQR, заменить их на верхнюю границу. 6. Построить гистограмму MedInc до и после обработки выбросов. 7. Удалить строки, где AveBedrms > 10 (аномальные значения).

	8. Проверить наличие пропусков после всех шагов. 9. Сохранить очищенный DataFrame в CSV. 10. Вычислить корреляцию MedInc с MedHouseVal до и после очистки.
10	1. Загрузить датасет titanic. 2. Применить комплексную обработку: пропуски в age заполнить предсказанием с помощью линейной регрессии (используя pclass, sex, fare). 3. Пропуски в embarked заполнить модой. 4. Удалить cabin (высокий процент пропусков). 5. Выявить выбросы в fare методом IQR, заменить на медиану. 6. Построить boxplot fare после обработки. 7. Сравнить средний возраст выживших и погибших до и после заполнения пропусков. 8. Сохранить результат в Excel с несколькими листами (исходный, очищенный, отчёт о пропусках). 9. Визуализировать распределение возраста до и после заполнения (две гистограммы). 10. Сделать вывод о влиянии способа заполнения на аналитические выводы.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Какие методы Pandas позволяют обнаружить пропущенные значения? Чем

отличается isnull() от isna()?

2. В каких случаях предпочтительнее удалять строки с пропусками, а не заполнять их? Назовите критерии.
3. Что такое медиана, и почему её чаще используют для заполнения пропусков, чем среднее арифметическое?
4. Как работает заполнение пропусков методом интерполяции? Приведите пример для временного ряда.
5. Что такое KNNImputer? Какие параметры нужно задать? В чём его преимущество и недостаток?
6. Какие методы выявления выбросов вы знаете? Опишите метод IQR (формула, интерпретация).
7. Чем отличается удаление выбросов от каппинга (clipping/winsorization)? Когда какой метод применим?
8. Как проверить, что после обработки пропусков распределение признака не исказилось? Назовите визуальные и статистические методы.
9. Что такое SimpleImputer в sklearn? Какие стратегии заполнения он поддерживает?
10. Почему перед заполнением пропусков (особенно средним/медианой) важно анализировать зависимость признака от других? Приведите пример ошибочного заполнения.

Критерии оценивания

Параметр	**Баллы**
Выполнение индивидуального задания (10 пунктов).	40
Качество обработки пропусков. Правильный выбор метода (удаление/заполнение) в зависимости от контекста, обоснование.	10
Обработка выбросов. Корректное применение IQR или z-score, выбор метода замены/удаления.	10
Ответы на контрольные вопросы (10 вопросов)	10

Структура и оформление отчёта. Наличие всех разделов, чистота кода, комментарии.	10
Выводы и интерпретация. Анализ изменений распределений, сравнение методов, обоснование выбора.	10
Сохранение результатов и визуализации. Сохранение очищенного DataFrame, наличие графиков «до/после».	10

Лабораторная работа № 5

Предобработка данных: кодирование и масштабирование

Цель работы: Освоить методы кодирования категориальных признаков и масштабирования числовых.

Теоретическое обоснование

Категориальные признаки. Большинство моделей работают только с числами. Способы кодирования:

- **Label Encoding** — каждой категории присваивается целое число (0,1,2,...). Недостаток: вводит ложный порядок. Подходит для порядковых (ordinal) признаков.
- **One-Hot Encoding** — создаёт бинарные столбцы для каждой категории. Длина вектора равна числу уникальных значений. Параметр drop='first' убирает один столбец, избегая мультиколлинеарности.
- **Ordinal Encoding** — задаётся явный порядок категорий (например, малый=1, средний=2, большой=3).
- **Target Encoding** (Mean Encoding) — замена категории на среднее значение целевой переменной для этой категории. Эффективно, но требует аккуратной кросс-валидации.

Масштабирование числовых признаков. Необходимо для методов, основанных на расстояниях (kNN, SVM, PCA, линейная регрессия с регуляризацией). Основные scaler'ы:

- **StandardScaler** — вычитает среднее и делит на стандартное отклонение: z = (x - μ) / σ. Результат: среднее 0, дисперсия 1.
- **MinMaxScaler** — сжимает в диапазон [0, 1] (или другой): x_scaled = (x - min) / (max - min). Чувствителен к выбросам.
- **RobustScaler** — использует медиану и IQR: x_scaled = (x - median) / IQR. Устойчив к выбросам.
- **MaxAbsScaler** — масштабирует на [-1, 1] для разреженных данных.

Pipeline — объект sklearn, объединяющий последовательность преобразований и финальную модель. Преимущества:

- Код становится короче и чище.
- Предотвращает утечку данных (трансформации обучаются только на train).
- Упрощает кросс-валидацию.

ColumnTransformer — позволяет применять разные преобразования к разным столбцам (например, OneHot для категориальных, StandardScaler для числовых).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет titanic из Seaborn. 2. Выбрать категориальные признаки: sex, embarked. Применить **One-Hot Encoding** с помощью pd.get_dummies (drop_first=False). 3. Для признака pclass выполнить **Ordinal Encoding** вручную (1→1, 2→2, 3→3) – он уже порядковый. 4. Применить **Label Encoding** к признаку sex (male=0, female=1) с использованием sklearn.preprocessing.LabelEncoder. 5. Выбрать числовые признаки: age, fare. Применить **StandardScaler** (среднее=0, std=1).

	6. Вывести среднее и стандартное отклонение масштабированных признаков (должны быть 0 и 1). 7. Применить **MinMaxScaler** к тем же признакам (диапазон [0,1]). 8. Сравнить результаты StandardScaler и MinMaxScaler на гистограммах (построить 2 графика). 9. Создать **Pipeline** из двух шагов: MinMaxScaler → OneHotEncoder для sex и embarked. 10. Применить Pipeline к исходным данным, вывести форму полученной матрицы.
2	1. Загрузить датасет adult из openml (fetch_openml('adult', version=1)). 2. Выбрать категориальные признаки: workclass, education, marital-status, occupation, relationship, race, sex, native-country. 3. Применить **One-Hot Encoding** только к признакам с числом уникальных значений < 10. 4. Для education (имеет порядок: Preschool→...→Doctorate) выполнить **Ordinal Encoding** с заданием порядка вручную. 5. Применить **Label Encoding** к sex (Male/Female). 6. Числовые признаки: age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week. 7. Применить **RobustScaler** (устойчив к выбросам). 8. Построить boxplot capital-gain до и после масштабирования. 9. Создать **ColumnTransformer**: для категориальных – OneHotEncoder, для числовых – RobustScaler. 10. Преобразовать данные, сохранить результат в CSV.
3	1. Загрузить датасет diamonds из Seaborn. 2. Категориальные признаки: cut, color, clarity (все порядковые). 3. Создать словарь порядка для cut: Fair=1, Good=2, Very Good=3, Premium=4, Ideal=5. Выполнить Ordinal Encoding. 4. Для color (D–J) задать порядок D=1,...,J=7. 5. Для clarity (I1=1, SI2=2, SI1=3, VS2=4, VS1=5, VVS2=6, VVS1=7, IF=8). 6. Числовые признаки: carat, depth, table, price, x, y, z. Применить **StandardScaler**. 7. Проверить, изменилось ли распределение price после масштабирования (график KDE). 8. Использовать MinMaxScaler для carat и price, сравнить диапазоны.

	9. Построить scatter plot carat vs price до и после масштабирования. 10. Сохранить преобразованный DataFrame в Excel.
4	1. Загрузить датасет mpg из Seaborn. 2. Категориальный признак origin (USA, Europe, Japan) – применить **One-Hot Encoding** с drop_first=True. 3. Признак cylinders – хотя числовой, но по сути категориальный. Выполнить **Label Encoding**. 4. Признак name – удалить (неинформативен). 5. Числовые признаки: mpg, displacement, horsepower, weight, acceleration. 6. Применить **StandardScaler**, затем восстановить исходные значения через inverse_transform (проверить). 7. Применить **MinMaxScaler** к horsepower (с пропусками – сначала заполнить медианой). 8. Построить гистограммы horsepower до и после MinMaxScaler. 9. Создать Pipeline: SimpleImputer(median) → StandardScaler для всех числовых. 10. Применить Pipeline, сохранить результат в CSV.
5	1. Загрузить датасет penguins из Seaborn. 2. Категориальные: species, island, sex. Применить **One-Hot Encoding** (удалив первый столбец). 3. Для sex также выполнить LabelEncoder (Male=1, Female=0) – сравнить с One-Hot. 4. Числовые: bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g. 5. Применить **RobustScaler** (медиана и IQR). 6. Сравнить средние значения до и после масштабирования (должны измениться). 7. Построить pairplot для масштабированных данных. 8. Выполнить **StandardScaler** и сравнить разброс (std) признаков. 9. Создать ColumnTransformer: OneHotEncoder для категориальных, RobustScaler для числовых. 10. Сохранить результат в Parquet.
6	1. Сгенерировать синтетический DataFrame: 200 строк, признаки: city (10 городов), income (логнормальное), age (равномерное). 2. Для city применить **One-Hot Encoding**, вычислить размерность после кодирования.

	3. Для income применить **StandardScaler**, затем построить гистограмму. 4. Для age применить **MinMaxScaler** в диапазон [0,1]. 5. Создать новый признак age_group (категории: молодой, средний, пожилой) на основе квантилей. 6. Применить **Ordinal Encoding** к age_group (задать порядок). 7. Сравнить корреляцию исходного age и закодированного age_group с income. 8. Использовать pd.get_dummies с параметром prefix. 9. Построить тепловую карту корреляции всех полученных признаков. 10. Сохранить результат в CSV.
7	1. Загрузить датасет tips из Seaborn. 2. Категориальные: sex, smoker, day, time. Применить **One-Hot Encoding** для всех. 3. Для day (порядок: Thur, Fri, Sat, Sun) выполнить **Ordinal Encoding** вручную. 4. Числовые: total_bill, tip, size. Применить **StandardScaler**. 5. Создать новый признак tip_percent = tip/total_bill*100. Масштабировать его отдельно MinMaxScaler. 6. Сравнить распределение size до и после StandardScaler (boxplot). 7. Построить scatter total_bill vs tip для исходных и масштабированных данных (два графика). 8. Использовать LabelEncoder для smoker (Yes/No). 9. Объединить все преобразования в Pipeline с FeatureUnion (два параллельных преобразования). 10. Сохранить результат в Excel.
8	1. Загрузить датасет wine из sklearn, преобразовать в DataFrame. Целевая переменная target – исключить из кодирования. 2. Все признаки числовые – масштабирование не требуется? Но для демонстрации: применить **StandardScaler**. 3. Добавить искусственный категориальный признак wine_type на основе квантилей alcohol (низкий, средний, высокий). 4. Выполнить **One-Hot Encoding** для wine_type. 5. Сравнить качество кластеризации k-means (n_clusters=3) на исходных и масштабированных данных (inertia). 6. Построить scatter plot первых двух главных компонент PCA до и после масштабирования.

	7. Применить **MinMaxScaler** и сравнить с StandardScaler через гистограммы. 8. Использовать RobustScaler для признака malic_acid (там могут быть выбросы). 9. Создать Pipeline: StandardScaler → PCA(n_components=2). Применить к данным. 10. Сохранить результат (2 компоненты) в CSV.
9	1. Загрузить датасет flights из Seaborn (после сводной таблицы). 2. Признаки: year, month, passengers. year и month – категориальные? Преобразовать month в LabelEncoder. 3. year оставить как есть (числовой). 4. passengers – применить **MinMaxScaler** (диапазон [0,1]). 5. Создать циклическое кодирование для month (sin, cos) – дополнительно. 6. Сравнить исходный и масштабированный ряды passengers на линейном графике. 7. Применить **StandardScaler** к логарифму passengers (log1p). 8. Построить гистограмму исходного и преобразованного passengers. 9. Создать ColumnTransformer: для month – OneHotEncoder, для year – StandardScaler. 10. Сохранить результат в CSV.
10	1. Загрузить датасет credit_card_fraud (малая выборка). 2. Признаки: Time, Amount, остальные анонимные V1–V28. 3. Применить **RobustScaler** к Amount (из-за выбросов). 4. Time преобразовать в часы (Time/3600) и применить **StandardScaler**. 5. Категориальных признаков нет. 6. Сравнить распределение Amount до и после RobustScaler (boxplot). 7. Применить **MinMaxScaler** к V1–V28 и сравнить диапазоны. 8. Построить тепловую карту корреляции после масштабирования. 9. Создать Pipeline: RobustScaler для Amount, StandardScaler для Time и MinMaxScaler для V1–V28 (через ColumnTransformer). 10. Сохранить преобразованные данные в HDF5.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём разница между **Label Encoding** и **One-Hot Encoding**? Когда каждый из них следует применять?
2. Почему для порядковых категорий (например, «малый», «средний», «большой») лучше использовать **Ordinal Encoding**, а не One-Hot?
3. Какой метод масштабирования (StandardScaler, MinMaxScaler, RobustScaler) следует выбрать, если в данных есть выбросы? Почему?
4. Что делает StandardScaler? Как преобразуются значения (формула)?
5. Какой диапазон значений получается после MinMaxScaler? Как изменить диапазон на [a, b]?
6. Что такое **Pipeline** в sklearn? Какие преимущества даёт его использование?
7. Что такое **ColumnTransformer** и для чего он нужен? Приведите пример использования.
8. Как избежать утечки данных (data leakage) при масштабировании и кодировании?
9. Какие проблемы могут возникнуть при применении One-Hot Encoding к признаку с большим количеством уникальных значений (например, 1000 городов)?

10. Можно ли применять масштабирование к категориальным признакам после One-Hot Encoding? Почему?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора методов кодирования. Соответствие типа кодирования природе признака (номинативный/порядковый).	10
Правильность выбора scaler'a. Учёт наличия выбросов, требуемого диапазона.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ изменений, сравнение методов, обоснование выбора.	10
Использование Pipeline/ColumnTransformer. Применение этих инструментов (если требуется в варианте) или демонстрация понимания.	10

Лабораторная работа № 6

Разделение данных и кросс-валидация

Цель: Научиться правильно разделять данные на обучающие и тестовые выборки, применять кросс-валидацию.

Теоретическое обоснование

Train-test split. Данные делятся на обучающую (70–80%) и тестовую (20–30%) выборки. Тестовая выборка **никогда не должна использоваться** для настройки модели — только для финальной оценки. Параметр stratify сохраняет пропорции классов (важно при дисбалансе).

Кросс-валидация (Cross-Validation). Метод оценки модели, при котором данные многократно разбиваются на обучающие и валидационные фолды.

- **K-Fold:** данные делятся на K равных частей. В каждом цикле одна часть — валидация, остальные K-1 — обучение. Итоговая оценка — среднее по K фолдам.
- **Stratified K-Fold:** то же, но с сохранением пропорций классов в каждом фолде (для классификации).
- **Leave-One-Out (LOO):** K = N (число объектов). Очень ресурсоёмко, но даёт несмещённую оценку.
- **ShuffleSplit:** случайное разбиение с заданным числом итераций и долей теста.

Зачем нужна CV? Одна случайная разбивка может быть неудачной (смещённая оценка). Кросс-валидация даёт среднюю оценку и её стандартное отклонение, позволяя судить о стабильности модели. Большое стандартное отклонение указывает на чувствительность модели к составу обучающей выборки.

cross_val_score — удобная функция, возвращающая массив оценок. **cross_validate** — возвращает дополнительно время обучения и предсказания. **cross_val_predict** — возвращает предсказания для каждого объекта, полученные при использовании его в валидационном фолде (полезно для построения confusion matrix).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris из sklearn (load_iris()). 2. Разделить данные на обучающую (70%) и тестовую (30%) выборки с помощью train_test_split, зафиксировать random_state=42. 3. Вывести размеры обучающей и тестовой выборок. 4. Выполнить стратифицированное разделение по целевому признаку (stratify=y). Проверить, сохранились ли пропорции классов. 5. Обучить модель LogisticRegression (max_iter=200) на обучающей выборке, оценить accuracy на тестовой. 6. Применить 5-кратную кросс-валидацию (KFold с shuffle=True) на всём датасете с той же моделью. Вывести среднюю accuracy и стандартное отклонение. 7. Сравнить accuracy из кросс-валидации с accuracy на одном тестовом разбиении. 8. Использовать cross_val_score с параметром scoring='f1_macro'. 9. Построить график зависимости качества от номера фолда (линия). 10. Сделать вывод о стабильности модели.
2	1. Загрузить датасет wine из sklearn (load_wine()). 2. Разделить данные на train (80%) и test (20%) со стратификацией. 3. Применить **Stratified K-Fold** с 10 фолдами (StratifiedKFold(n_splits=10, shuffle=True, random_state=42)). 4. Для каждого фолда вывести размеры обучающей и валидационной выборок. 5. Обучить DecisionTreeClassifier(max_depth=5) и оценить accuracy на каждом валидационном фолде. 6. Вычислить среднюю accuracy и доверительный интервал (среднее ± 2*std). 7. Сравнить с результатом простого train_test_split (одно разбиение). 8. Использовать cross_validate для получения времени обучения и оценки. 9. Построить boxplot оценок по фолдам. 10. Сохранить результаты кросс-валидации в CSV.
3	1. Загрузить датасет digits из sklearn (8×8 изображения). 2. Разделить на train/test (75/25) с random_state=123. 3. Обучить KNeighborsClassifier(n_neighbors=3). Оценить accuracy на тесте.

	4. Выполнить **Leave-One-Out** кросс-валидацию (LeaveOneOut()) – только для первых 50 объектов (из-за ресурсоёмкости). 5. Вывести среднюю accuracy по LOO. 6. Выполнить 5-кратную кросс-валидацию с KFold. 7. Сравнить результаты LOO и KFold. 8. Для KFold построить график обучения (learning curve) – зависимость качества от размера обучающей выборки. 9. Использовать cross_val_predict для получения предсказаний на каждом фолде, построить confusion matrix. 10. Сделать вывод о целесообразности LOO для данного размера данных.
4	1. Загрузить датасет breast_cancer из sklearn. 2. Разделить на train/test (60/40) со стратификацией. 3. Обучить SVC(kernel='rbf', C=1). Оценить precision, recall, f1 на тесте. 4. Применить **Repeated Stratified K-Fold** (5 фолдов, 3 повтора) – RepeatedStratifiedKFold. 5. Вывести средние и стандартные отклонения для accuracy, precision, recall. 6. Сравнить с результатами однократного разбиения. 7. Построить ящик с усами (boxplot) для метрики F1 по всем фолдам и повторам. 8. Использовать cross_val_score с scoring='roc_auc'. 9. Выполнить разделение с train_test_split без стратификации и проверить дисбаланс классов в выборках. 10. Сохранить результаты в Excel.
5	1. Загрузить датасет diabetes из sklearn (задача регрессии). 2. Разделить на train/test (80/20). 3. Обучить LinearRegression. Вычислить MSE и R^2 на тесте. 4. Применить **K-Fold** (5 фолдов) для регрессии, используя cross_val_score с scoring='neg_mean_squared_error'. 5. Преобразовать отрицательные MSE в положительные, вывести среднее. 6. Выполнить кросс-валидацию с ShuffleSplit (n_splits=10, test_size=0.2). 7. Сравнить разброс оценок между KFold и ShuffleSplit. 8. Построить график предсказанных vs истинных значений для одного из фолдов. 9. Оценить стабильность R^2 через кросс-валидацию. 10. Сохранить модель, обученную на всех данных, в файл (joblib).
6	1. Загрузить датасет california_housing из sklearn.

	2. Разделить на train/test (70/30) с random_state=42. 3. Нормализовать признаки StandardScaler (обучить на train, применить к train и test). 4. Обучить Ridge(alpha=1.0). Оценить MAE и RMSE. 5. Выполнить **5-кратную кросс-валидацию** внутри Pipeline (StandardScaler + Ridge). 6. Использовать cross_val_score с scoring='neg_root_mean_squared_error'. 7. Вывести средний RMSE и его std. 8. Сравнить с RMSE на одном тестовом разбиении. 9. Построить learning curve (зависимость качества от количества объектов в обучении) с помощью learning_curve. 10. Сделать вывод о переобучении или недообучении.
7	1. Загрузить датасет titanic из Seaborn. Провести предобработку (заполнить age медианой, one-hot для sex, embarked). 2. Разделить на train/test (75/25) со стратификацией по survived. 3. Обучить RandomForestClassifier(n_estimators=100). Оценить accuracy и AUC-ROC. 4. Применить **Stratified K-Fold** с 10 фолдами. 5. Для каждого фолда вычислить AUC-ROC, построить ROC-кривые на одном графике (по одному на фолд). 6. Вычислить среднюю AUC-ROC и доверительный интервал. 7. Сравнить с AUC-ROC на тестовом разбиении. 8. Использовать cross_validate с несколькими метриками: 'accuracy', 'roc_auc', 'f1'. 9. Построить heatmap ошибок (confusion matrix) для предсказаний, полученных через cross_val_predict. 10. Сохранить результаты в CSV.
8	1. Сгенерировать синтетический датасет для классификации с помощью make_classification (1000 образцов, 20 признаков, 2 класса, дисбаланс 0.9). 2. Разделить на train/test (80/20) без стратификации. Проверить пропорции классов – они смещены? 3. Выполнить стратифицированное разделение и снова проверить пропорции. 4. Обучить LogisticRegression. Сравнить accuracy на тесте в обоих случаях (без стратификации и со стратификацией). 5. Применить **Stratified K-Fold** (5 фолдов) и обычный KFold. Сравнить средние accuracy. 6. Построить boxplot оценок для обоих методов кросс-валидации. 7. Использовать GroupKFold – сгенерировать группы (например,

	по 10 объектов в группе). 8. Оценить модель с GroupKFold. 9. Сравнить разброс оценок между KFold, StratifiedKFold и GroupKFold. 10. Сделать вывод о важности стратификации при дисбалансе.
9	1. Загрузить датасет mnist_784 (fetch_openml, только 1000 образцов для скорости). 2. Разделить на train/test (50/50) – нестандартное разбиение для демонстрации. 3. Обучить KNeighborsClassifier(n_neighbors=5). Оценить accuracy. 4. Выполнить **5-кратную кросс-валидацию** с перемешиванием (KFold(shuffle=True)). 5. Сравнить среднюю accuracy кросс-валидации с accuracy на тестовом разбиении. 6. Использовать cross_val_score с scoring='balanced_accuracy'. 7. Построить график зависимости accuracy от k (числа соседей) на кросс-валидации (k=1,3,5,7,9). 8. Для оптимального k выполнить кросс-валидацию повторно. 9. Применить LeaveOneOut для первых 100 объектов и сравнить время выполнения с KFold. 10. Сохранить оптимальную модель в файл.
10	1. Загрузить датасет heart_disease (из UCI или sklearn). 2. Разделить на train/test (70/30) со стратификацией. 3. Построить Pipeline: StandardScaler → SVC(kernel='rbf'). 4. Выполнить **10-кратную стратифицированную кросс-валидацию** всего Pipeline. 5. Использовать cross_val_score с метриками 'accuracy', 'precision', 'recall', 'f1'. 6. Вывести таблицу со средними и стандартными отклонениями для каждой метрики. 7. Сравнить с результатами однократного разбиения (на тесте). 8. Построить ROC-кривые для каждого фолда (на одном графике) и среднюю ROC-кривую. 9. Вычислить средний AUC-ROC. 10. Сделать вывод о том, насколько стабильна модель при различных разбиениях.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Зачем нужно разделять данные на обучающую и тестовую выборки? Что произойдёт, если оценивать модель на тех же данных, на которых она обучалась?
2. Что такое стратифицированное разделение (stratify)? В каких случаях оно необходимо?
3. Чем отличается обычный K-Fold от Stratified K-Fold?
4. Что такое кросс-валидация? Назовите три её разновидности и укажите, для каких задач каждая подходит.
5. Какие преимущества даёт кросс-валидация перед однократным разбиением на train/test?
6. Что такое cross_val_score? Какие параметры она принимает и что возвращает?
7. Как интерпретировать стандартное отклонение оценок, полученных при кросс-валидации? Большое std – это хорошо или плохо?
8. Что такое Leave-One-Out (LOO) кросс-валидация? В чём её недостатки при большом объёме данных?
9. Какой метод кросс-валидации следует использовать для временных рядов? Почему нельзя применять обычный K-Fold?

10. Что такое cross_val_predict? В чём отличие от cross_val_score? Когда её используют?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора метода разделения. Использование стратификации при дисбалансе, выбор K-Fold или LOO по ситуации.	10
Качество кросс-валидации. Корректное применение cross_val_score, интерпретация разброса оценок.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Сравнение методов, анализ стабильности, обоснование выбора.	10
Визуализации и сохранение результатов. Графики (boxplot, ROC-кривые, learning curve) и сохранение моделей/CSV по заданию.	10

Лабораторная работа № 7

Линейная регрессия

Цель: Реализовать и оценить модель линейной регрессии.

Теоретическое обоснование

Модель. Для набора признаков $X = (x_1, x_2, \ldots, x_n)$ предсказание $\hat{y} = w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$, где w_0 — свободный член (intercept), w_i — коэффициенты.

Обучение (метод наименьших квадратов, OLS). Минимизируется сумма квадратов остатков: $\Sigma(y_i - \hat{y}_i)^2 \rightarrow \min$. Решение в матричной форме: $w = (X^T X)^{-1} X^T y$.

Метрики качества регрессии:

- **MSE (Mean Squared Error)** — средний квадрат ошибки. Чувствительна к выбросам. Размерность в квадрате целевой переменной.
- **MAE (Mean Absolute Error)** — средняя абсолютная ошибка. Устойчива к выбросам.
- **RMSE (Root Mean Squared Error)** = $\sqrt{MSE}$. Имеет ту же размерность, что и y. Более интерпретируема.
- **R^2 (коэффициент детерминации)** = $1 - \Sigma(y_i - \hat{y}_i)^2 / \Sigma(y_i - \bar{y})^2$. Доля дисперсии, объяснённая моделью. Может быть отрицательным (если модель хуже, чем константное предсказание среднего).

Диагностика модели:

- **График «фактические vs предсказанные»** — точки должны лежать на диагонали.
- **График остатков** (predicted vs residuals) — не должно быть явного тренда или гетероскедастичности (изменяющейся дисперсии). Остатки должны быть распределены нормально около нуля.

Интерпретация коэффициентов: При увеличении признака x_i на 1 (при фиксированных остальных), целевая переменная y изменяется на w_i. Если признаки стандартизированы, коэффициенты сравнимы по величине — больший по модулю означает большее влияние.

Ограничения линейной регрессии: предполагает линейную связь,

независимость и нормальность остатков, отсутствие мультиколлинеарности. Нарушения снижают достоверность интервальных оценок, но точечные предсказания могут быть всё ещё полезны.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет california_housing из sklearn.datasets.fetch_california_housing(). 2. Создать DataFrame, разделить данные на признаки (X) и целевую переменную (MedHouseVal). 3. Разделить выборку на обучающую (80%) и тестовую (20%) с random_state=42. 4. Обучить модель LinearRegression на обучающей выборке. 5. Вычислить MSE, MAE, RMSE, R^2 на тестовой выборке. 6. Построить график «фактические vs предсказанные значения» (точки + диагональ). 7. Построить график остатков (residuals) – предсказанные vs остатки. 8. Вывести коэффициенты модели и интерпретировать три самых значимых (по модулю). 9. Сравнить R^2 с константной моделью (предсказание среднего). 10. Сохранить модель в файл linear_model.pkl.
2	1. Загрузить датасет diabetes из sklearn.datasets.load_diabetes(). 2. Изучить описание признаков (DESCR). 3. Разделить на train/test (75/25) со стратификацией? (регрессия – без стратификации). 4. Обучить LinearRegression. 5. Вывести коэффициенты для каждого признака, назвать признак с наибольшим влиянием. 6. Оценить модель по метрикам: MAE, MSE, R^2. 7. Построить график остатков и проверить их нормальность (гистограмма + Q-Q plot). 8. Добавить признак-полином (квадрат одного из признаков) и сравнить R^2. 9. Использовать Ridge с α=1.0, сравнить коэффициенты с

	линейной регрессией. 10. Сделать вывод о мультиколлинеарности (посмотреть корреляцию признаков).
3	1. Загрузить датасет boston из sklearn.datasets.load_boston() (если доступен, иначе fetch_openml). 2. Создать DataFrame, вывести описательную статистику. 3. Разделить на train/test (70/30). 4. Обучить LinearRegression. 5. Построить heatmap корреляции признаков, выявить признаки, сильно коррелирующие с целевой переменной. 6. Вычислить RMSE и R^2 на тесте. 7. Построить график фактических vs предсказанных, подписать оси. 8. Найти объект с наибольшей ошибкой (абсолютной) и проанализировать его признаки. 9. Выполнить стандартизацию признаков (StandardScaler) и обучить модель заново. Сравнить коэффициенты (масштабированные vs исходные). 10. Сохранить предсказания и остатки в CSV.
4	1. Сгенерировать синтетические данные: make_regression(n_samples=200, n_features=3, noise=10, random_state=42). 2. Разделить на train/test (80/20). 3. Обучить линейную регрессию. 4. Сравнить истинные коэффициенты (из генерации) с полученными. 5. Построить график остатков, проверить гомоскедастичность (равномерность разброса). 6. Добавить шумовой признак (случайные числа) и обучить модель заново – изменился ли R^2? 7. Использовать RidgeCV для автоматического подбора α, сравнить коэффициенты. 8. Построить график зависимости R^2 от количества признаков (forward selection вручную). 9. Вычислить доверительные интервалы для коэффициентов (используя statsmodels). 10. Сделать вывод о значимости признаков.
5	1. Загрузить датасет mpg из Seaborn, удалить строки с пропусками в horsepower. 2. Выбрать

	признаки: displacement, horsepower, weight, acceleration, cylinders. Цель: mpg. 3. Разделить на train/test (80/20). 4. Обучить LinearRegression. 5. Вывести R^2 и MAE. 6. Построить график остатков, добавить горизонтальную линию на уровне 0. 7. Выполнить логарифмическое преобразование целевой переменной (np.log(mpg)) и обучить новую модель. Сравнить R^2. 8. Интерпретировать коэффициент для weight в исходной и логарифмической модели. 9. Построить scatter plot weight vs mpg с наложением линии регрессии. 10. Сохранить обе модели (исходную и с логарифмом) в один файл.
6	1. Загрузить датасет concrete (прочность бетона) из sklearn или открытого источника. 2. Провести первичный анализ: гистограммы, корреляции. 3. Разделить на train/test (80/20). 4. Обучить LinearRegression. 5. Оценить качество по R^2 и RMSE. 6. Построить график «предсказанные vs фактические», раскрасить точки по величине ошибки. 7. Применить PolynomialFeatures (степень 2) и обучить модель на полиномиальных признаках. Сравнить R^2 – есть ли переобучение? 8. Для полиномиальной модели построить график остатков. 9. Выполнить регуляризацию Lasso (α=0.1) на полиномиальных признаках, сравнить количество ненулевых коэффициентов. 10. Сделать вывод о выборе модели.
7	1. Загрузить датасет fish_market (предсказание веса рыбы). 2. Признаки: длина, высота, ширина и др. 3. Разделить на train/test (70/30). 4. Обучить линейную регрессию. 5. Вычислить MSE и R^2. 6. Построить график остатков и выявить выбросы (остатки > 2σ). 7. Удалить выбросы и обучить модель заново – как изменились метрики?

	8. Добавить взаимодействие признаков (например, length * height) и сравнить R^2. 9. Использовать SVR с линейным ядром, сравнить с линейной регрессией. 10. Сохранить лучшую модель в joblib.
8	1. Загрузить датасет real_estate (цена дома от площади, возраста и т.д.). 2. Разделить на train/test (80/20). 3. Обучить линейную регрессию. 4. Вывести коэффициенты и их стандартные ошибки (через statsmodels). 5. Построить частичные графики зависимости (partial dependence plots) для двух важнейших признаков. 6. Проверить предположение о линейности: добавить квадрат признака area^2, проверить значимость. 7. Выполнить масштабирование признаков и сравнить коэффициенты. 8. Построить learning curve (зависимость ошибки от размера обучения). 9. Использовать cross_val_score с 5 фолдами для оценки R^2. 10. Сделать вывод о стабильности модели.
9	1. Загрузить датасет airfoil (уровень шума аэродинамической трубы). 2. Разделить на train/test (75/25). 3. Обучить LinearRegression. 4. Вычислить MAE, RMSE, R^2. 5. Построить график остатков, проверить на наличие тренда. 6. Применить QuantileTransformer к целевой переменной и обучить модель на трансформированной цели. 7. Сравнить RMSE в исходной шкале (обратное преобразование). 8. Использовать HuberRegressor (робастная регрессия) и сравнить с обычной. 9. Визуализировать фактические vs предсказанные для обеих моделей. 10. Сохранить таблицу сравнения метрик в CSV.
10	1. Загрузить датасет wine_quality (красное вино, предсказание качества от 0 до 10). 2. Признаки: кислотность, сахар, pH, алкоголь и др. 3. Разделить на train/test (80/20). 4. Обучить LinearRegression (несмотря на то, что качество –

	целые числа, считаем регрессией). 5. Оценить R^2 и MAE. 6. Построить scatter plot фактических vs предсказанных, добавить jitter для дискретных значений. 7. Выполнить бинаризацию целевой переменной (хорошее/плохое вино) и решать задачу классификации логистической регрессией. Сравнить accuracy. 8. Для регрессии вычислить корреляцию предсказанных и фактических. 9. Построить график важности признаков (абсолютные значения коэффициентов). 10. Сделать вывод о том, насколько линейная модель подходит для этого датасета.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Запишите формулу линейной регрессии для одного признака. Что означают коэффициенты w_0 и w_1?
2. Какой метод используется для нахождения коэффициентов линейной регрессии? Опишите его суть.
3. Что такое R^2 (коэффициент детерминации)? В каких пределах он может изменяться и как интерпретируется?

4. В чём разница между MSE, MAE и RMSE? Какую метрику легче интерпретировать и почему?
5. Что такое «остатки» (residuals)? Как по графику остатков определить, что модель неадекватна?
6. Какие предположения делает классическая линейная регрессия относительно остатков?
7. Как интерпретировать коэффициент признака в линейной регрессии, если признаки были стандартизированы?
8. Что такое мультиколлинеарность и как она влияет на коэффициенты линейной регрессии?
9. Как проверить, что добавление нового признака значимо улучшает модель? Какие критерии используют?
10. В чём отличие линейной регрессии от гребневой (Ridge) и лассо (Lasso) регрессии?

Критерии оценивания

Параметр	**Баллы**
Выполнение индивидуального задания (10 пунктов).	40
Правильность расчёта метрик. Корректное вычисление MSE, MAE, RMSE, R^2, интерпретация.	10
Качество визуализаций (фактические vs предсказанные, остатки). Наличие подписей, анализ паттернов.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация коэффициентов. Глубина анализа, связь с бизнес-контекстом (где возможно).	10
Сравнение с другими методами (по заданию). Ridge, Lasso, полиномы, робастные методы – если предусмотрены вариантом.	10

Лабораторная работа № 8

Полиномиальная регрессия и регуляризация

Цель: Исследовать влияние степени полинома и регуляризации на качество модели.

Теоретическое обоснование

Полиномиальная регрессия. Линейная модель не может уловить нелинейные зависимости. Расширяем пространство признаков, добавляя степени и произведения: для одного признака x создаём x, x^2, x^3, … (через PolynomialFeatures). Получаем линейную модель в расширенном пространстве, которая нелинейна относительно исходного x.

Проблема переобучения. С увеличением степени полинома модель начинает подстраиваться под шум, резко ухудшая качество на тесте. Коэффициенты становятся очень большими по модулю.

Регуляризация — добавление штрафа за большие коэффициенты в функцию потерь:

- **Ridge (L2-регуляризация):** штраф = $\alpha \cdot \Sigma w_i^2$. Все коэффициенты сжимаются к нулю, но не становятся нулевыми. Стабилизирует решение при мультиколлинеарности.
- **Lasso (L1-регуляризация):** штраф = $\alpha \cdot \Sigma |w_i|$. Способна обнулять коэффициенты, выполняя отбор признаков. Полезна, когда признаков много, но многие неинформативны.
- **ElasticNet:** комбинация L1 и L2: $\alpha \cdot (\rho \cdot \Sigma |w_i| + (1-\rho)/2 \cdot \Sigma w_i^2)$.

Параметр α (или C в некоторых реализациях): чем больше α, тем сильнее регуляризация, тем меньше коэффициенты. При очень большом α модель недообучается (смещение), при очень малом — переобучается (высокая дисперсия).

Выбор степени полинома и α обычно выполняется по кросс-валидации (GridSearchCV).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Сгенерировать синтетические данные: make_regression(n_samples=100, n_features=1, noise=10, random_state=42). 2. Добавить нелинейность: преобразовать целевую переменную как y = y + 2 * X**2. 3. Разделить данные на train/test (80/20). 4. Обучить линейную регрессию на исходных признаках (только X). Вычислить R^2 на тесте. 5. Создать полиномиальные признаки степени 2 с помощью PolynomialFeatures(include_bias=False). 6. Обучить линейную регрессию на полиномиальных признаках. Сравнить R^2 с п.4. 7. Построить график исходных данных и предсказаний обеих моделей (линия регрессии). 8. Создать полиномиальные признаки степени 5 и обучить модель. Пронаблюдать переобучение (график). 9. Применить гребневую регрессию Ridge(alpha=1.0) к полиномам степени 5, сравнить R^2. 10. Построить график зависимости R^2 на тесте от степени полинома (от 1 до 10) – кривая переобучения.
2	1. Сгенерировать данные по формуле: y = 0.5 * X + 3 * X**2 - 2 * X**3 + нормальный шум. 2. Разделить на train/test (70/30). 3. Обучить линейную регрессию на исходном X (без полиномов). Зафиксировать R^2. 4. Применить PolynomialFeatures степени 3, обучить линейную регрессию. 5. Вывести коэффициенты модели степени 3, сравнить с истинными (0.5, 3, -2). 6. Построить график истинной функции (без шума) и предсказаний модели. 7. Обучить модель степени 10 – показать переобучение (колебания на краях). 8. Применить Lasso(alpha=0.01) к полиномам степени 10 – какие коэффициенты обнулились? 9. Подобрать оптимальную степень полинома через кросс-валидацию (5-fold) для степеней 1–8. 10. Сохранить лучшую модель (Pipeline: PolynomialFeatures + Ridge) в joblib.
3	1. Загрузить датасет diabetes из sklearn, использовать только один

	признак (например, bmi). 2. Разделить на train/test (80/20). 3. Визуализировать зависимость целевой переменной от BMI (scatter plot). 4. Обучить линейную регрессию, вычислить R^2 и MAE. 5. Добавить полиномиальные признаки степени 2, обучить модель, сравнить R^2. 6. Добавить степень 3 и 4 – построить кривые предсказаний на одном графике. 7. Применить Ridge с $\alpha=10$ к полиномам степени 4, сравнить с обычной линейной. 8. Использовать GridSearchCV для подбора α (0.1, 1, 10, 100) для Ridge на полиномах степени 3. 9. Построить график зависимости коэффициентов от α (штрафной траектории). 10. Сделать вывод о том, какая степень полинома оптимальна.
4	1. Сгенерировать синтетические данные: n_samples=150, один признак, функция $y = \sin(2\pi X) + шум$. 2. Разделить на train/test. 3. Обучить линейную регрессию – оценить качество (будет низким). 4. Применить PolynomialFeatures степени 3, 6, 10 – построить графики предсказаний. 5. Для степени 10 вычислить норму коэффициентов (L2) – она будет большой. 6. Применить Ridge(alpha=0.1) и Lasso(alpha=0.1) к полиномам степени 10. Сравнить норму коэффициентов. 7. Построить графики предсказаний для обычной линейной (степень 10), Ridge и Lasso на одном поле. 8. Подобрать α для Ridge по кросс-валидации (RidgeCV). 9. Сравнить R^2 лучшей Ridge-модели с линейной регрессией на исходных данных. 10. Сохранить таблицу метрик (MSE, R^2) для всех моделей в DataFrame.
5	1. Загрузить датасет bike_sharing (число велосипедов в зависимости от погоды, дня недели). 2. Выбрать признаки: temp, humidity, windspeed. Цель: count. 3. Разделить на train/test (75/25). 4. Обучить линейную регрессию, вычислить RMSE. 5. Создать полиномиальные признаки степени 2 (включая взаимодействия) с PolynomialFeatures.

	6. Обучить линейную регрессию на полиномах – сравнить RMSE. 7. Применить Lasso с α=0.001 – посмотреть, сколько признаков обнулилось. 8. Использовать ElasticNet (смесь L1 и L2) с α=0.01, l1_ratio=0.5. 9. Построить график важности признаков (абсолютные коэффициенты) для Lasso. 10. Сохранить лучшую модель (Pipeline с масштабированием + полиномы + ElasticNet) в pickle.
6	1. Сгенерировать данные с двумя признаками: y = 2 + 3*X1 - 2*X2 + 0.5*X1*X2 + X1^2 + шум. 2. Разделить на train/test (80/20). 3. Обучить линейную регрессию на исходных X1, X2 – R^2 будет низким из-за отсутствия взаимодействия. 4. Создать полиномиальные признаки степени 2 (включая произведение и квадраты). 5. Обучить модель на полиномах – сравнить R^2 и коэффициенты с истинными. 6. Построить heatmap корреляции полиномиальных признаков – выявить мультиколлинеарность. 7. Применить Ridge(alpha=1) для борьбы с мультиколлинеарностью. 8. Подобрать α по кросс-валидации (от 0.01 до 100). 9. Визуализировать предсказанную поверхность (3D plot) для лучшей модели. 10. Сравнить норму коэффициентов у обычной линейной (на полиномах) и Ridge.
7	1. Загрузить датасет california_housing, использовать признаки MedInc, AveRooms, HouseAge. 2. Разделить на train/test (70/30), масштабировать признаки (StandardScaler). 3. Обучить линейную регрессию, зафиксировать R^2. 4. Добавить полиномы степени 2 (только квадраты, без взаимодействий). 5. Сравнить R^2 – есть ли улучшение? 6. Добавить все взаимодействия степени 2 – сравнить с п.4. 7. Применить Lasso с α=0.01 и Ridge с α=1 к полному набору полиномов (степень 2 с взаимодействиями). 8. Построить график коэффициентов Lasso и Ridge (сравнить). 9. Использовать LassoCV для автоматического подбора α, вывести оптимальное α.

	10. Сделать вывод о том, какие исходные признаки или их комбинации важны.
8	1. Сгенерировать данные с высоким уровнем шума: y = 2 + X - 0.5*X^2 + шум(σ=5). 2. Разделить на train/test (80/20). 3. Обучить полиномиальную регрессию степени 2 – получить некоторое качество. 4. Обучить степень 10 – показать сильное переобучение (высокий R^2 на обучении, низкий на тесте). 5. Построить график кривых обучения (learning curves) для степени 10. 6. Применить Ridge с α=100 к степени 10 – как изменились предсказания (график). 7. Применить Lasso с α=10 – какие коэффициенты обнулились? 8. Сравнить метрики (MSE, R^2) для трёх моделей: степень 2, степень 10, Ridge (степень 10). 9. Подобрать оптимальную степень и α с помощью GridSearchCV (степени 1–8, α = 0.01,0.1,1,10). 10. Сохранить лучший Pipeline в файл.
9	1. Загрузить датасет energy_efficiency (нагрузка на отопление в зависимости от площади, остекления и др.). 2. Разделить на train/test (75/25). 3. Обучить линейную регрессию на исходных признаках. 4. Создать полиномиальные признаки степени 3 (будет очень много) – оценить размерность. 5. Обучить на полиномах степени 3 – модель может переобучиться. 6. Применить Lasso с α=0.001 – сократить число признаков. 7. Построить график зависимости количества ненулевых коэффициентов от α (α=0.0001, 0.001, 0.01, 0.1). 8. Использовать SelectFromModel для отбора признаков на основе Lasso. 9. Обучить простую линейную регрессию на отобранных признаках, сравнить R^2. 10. Сохранить отобранные признаки и их коэффициенты.
10	1. Сгенерировать данные по формуле: y = 5 * sin(π * X) + X^3 + шум. 2. Разделить на train/test. 3. Визуализировать данные. 4. Обучить линейную регрессию, полиномы степени 3, 7, 15 – построить графики предсказаний.

	5. Для степени 15 вычислить RMSE на обучении и тесте – убедиться в переобучении. 6. Применить Ridge(alpha=0.5) к полиномам степени 15 – оценить улучшение на тесте. 7. Применить Lasso(alpha=0.5) – посмотреть, сколько коэффициентов стало нулевыми. 8. Использовать ElasticNet(alpha=0.1, l1_ratio=0.7). 9. Сравнить R^2 для всех регуляризованных моделей. 10. Построить график предсказаний лучшей регуляризованной модели на фоне истинной функции.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Зачем нужна полиномиальная регрессия, если есть нелинейные зависимости?
2. Что такое переобучение (overfitting) в контексте полиномиальной регрессии? Как его распознать по метрикам?
3. Какой параметр в PolynomialFeatures отвечает за включение признаков взаимодействия (произведений)?
4. В чём разница между L1-регуляризацией (Lasso) и L2-регуляризацией (Ridge)?
5. Какой эффект оказывает увеличение коэффициента регуляризации α в Ridge и Lasso?
6. Почему при использовании полиномиальных признаков высокой степени коэффициенты модели становятся очень большими?

7. Как регуляризация помогает бороться с переобучением? Приведите интуитивное объяснение.
8. Что такое ElasticNet и когда его применяют?
9. Как выбрать оптимальную степень полинома и параметр регуляризации? Назовите два метода.
10. Нужно ли масштабировать признаки перед созданием полиномиальных признаков? Почему?

Критерии оценивания

Параметр	**Баллы**
Выполнение индивидуального задания (10 пунктов).	40
Демонстрация переобучения. Чёткое выявление случаев, когда модель показывает высокое качество на обучении и низкое на тесте.	10
Правильность применения регуляризации. Корректный выбор α, сравнение эффектов L1 и L2.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ влияния степени полинома и α на коэффициенты и качество.	10
Визуализации (графики предсказаний, кривые обучения). Наличие и качество графиков, сравнение разных моделей на одном поле.	10

Лабораторная работа № 9

Логистическая регрессия для бинарной классификации

Цель: Применить логистическую регрессию для задачи бинарной классификации.

Теоретическое обоснование

Модель. Вероятность принадлежности к классу 1: p = 1 / (1 + e^{-z}), где z = $w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n$. Функция называется сигмоидой. Решающее правило: если p ≥ 0.5 → класс 1, иначе класс 0.

Обучение. Минимизируется логистическая функция потерь (log loss, binary cross-entropy): -[y·log(p) + (1-y)·log(1-p)]. Нет аналитического решения, используется градиентный спуск или его модификации (lbfgs, newton-cg, sag, saga).

Метрики качества (для бинарной классификации):

- **Accuracy** = (TP+TN)/(TP+TN+FP+FN). Не подходит при сильном дисбалансе.
- **Precision (точность)** = TP/(TP+FP). Доля правильно предсказанных положительных среди всех, кого модель назвала положительными.
- **Recall (полнота)** = TP/(TP+FN). Доля положительных, которые модель смогла обнаружить.
- **F1-score** = 2·(precision·recall)/(precision+recall). Гармоническое среднее.
- **Confusion matrix** — таблица 2×2 с TP, TN, FP, FN.
- **ROC-кривая** — график зависимости TPR (recall) от FPR (1 - specificity) при варьировании порога. AUC-ROC — площадь под кривой. 0.5 — случайный классификатор, 1.0 — идеальный.

Интерпретация коэффициентов: $\exp(w_i)$ — отношение шансов (odds ratio). При увеличении x_i на 1, шансы (p/(1-p)) умножаются на $\exp(w_i)$. Знак коэффициента показывает направление влияния.

Параметры регуляризации: C — обратный α. Чем меньше C, тем сильнее регуляризация.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание

1	1. Загрузить датасет Titanic из Seaborn (sns.load_dataset('titanic')). 2. Провести предобработку: удалить столбцы deck, alive, who, adult_male, embark_town; заполнить пропуски в age медианой по классу (pclass); заполнить пропуски в embarked модой. 3. Выбрать признаки: pclass, sex, age, sibsp, parch, fare, embarked. Применить One-Hot Encoding для sex и embarked. 4. Разделить данные на обучающую (75%) и тестовую (25%) выборки со стратификацией по survived. 5. Обучить LogisticRegression(solver='lbfgs', max_iter=1000) на обучающей выборке. 6. Вычислить accuracy, precision, recall, F1-score на тестовой выборке. 7. Построить confusion matrix (матрицу ошибок) и интерпретировать её. 8. Построить ROC-кривую и вычислить AUC-ROC. 9. Проанализировать коэффициенты модели: какой признак сильнее всего влияет на выживание (по модулю коэффициента). 10. Сохранить обученную модель в файл logreg_titanic.pkl.
2	1. Загрузить датасет Pima Indians Diabetes из openml (fetch_openml('diabetes', version=1)). 2. Вывести информацию о данных, проверить пропуски (в данном датасете пропусков нет, но показать метод). 3. Разделить данные на признаки (X) и целевую переменную (y). 4. Разделить выборку на train/test (70/30) со стратификацией. 5. Масштабировать признаки с помощью StandardScaler. 6. Обучить LogisticRegression(C=1.0, solver='lbfgs'). 7. Оценить модель по метрикам: accuracy, precision, recall, F1. 8. Построить confusion matrix и рассчитать вручную TP, TN, FP, FN. 9. Построить ROC-кривую и вычислить AUC-ROC. 10. Подобрать оптимальный порог классификации (не 0.5) по максимуму F1-меры и сравнить accuracy с порогом 0.5.
3	1. Загрузить датасет creditcard (малая выборка для скорости) или синтетический с дисбалансом классов (make_classification). 2. Вывести распределение классов (доля положительных и отрицательных). 3. Разделить данные на train/test (80/20) со стратификацией. 4. Обучить LogisticRegression(class_weight='balanced') для борьбы с дисбалансом.

	5. Обучить вторую модель без учёта весов (class_weight=None). 6. Сравнить precision, recall и F1 для обеих моделей (особенно для миноритарного класса). 7. Построить confusion matrix для обеих моделей. 8. Построить ROC-кривые для обеих моделей на одном графике. 9. Сравнить AUC-ROC. 10. Сделать вывод о влиянии class_weight на качество предсказания миноритарного класса.
4	1. Загрузить датасет heart_disease (UCI Heart Disease, через fetch_openml). 2. Провести предобработку: проверить пропуски, закодировать категориальные признаки. 3. Разделить данные на train/test (75/25) со стратификацией. 4. Обучить LogisticRegression(penalty='l2', C=1.0). 5. Вывести коэффициенты модели и интерпретировать три самых важных признака. 6. Построить confusion matrix и рассчитать метрики: accuracy, precision, recall, F1. 7. Построить ROC-кривую, вычислить AUC-ROC. 8. Найти оптимальный порог по индексу Юдена (Youden's J = TPR - FPR). 9. Применить найденный порог и сравнить метрики с порогом 0.5. 10. Сохранить метрики в CSV.
5	1. Загрузить датасет bank (marketing campaign) через fetch_openml. 2. Целевая переменная: подписка на депозит. Проверить дисбаланс. 3. Выбрать числовые признаки: age, balance, duration и категориальные: job, marital, education. 4. Создать ColumnTransformer: StandardScaler для числовых, OneHotEncoder для категориальных. 5. Построить Pipeline: трансформация → LogisticRegression. 6. Обучить модель, оценить accuracy и ROC-AUC. 7. Построить Precision-Recall кривую (особенно важна при дисбалансе). 8. Рассчитать среднюю точность (average precision). 9. Построить матрицу ошибок в виде тепловой карты. 10. Сделать вывод о том, подходит ли accuracy для оценки данной модели.
6	1. Сгенерировать синтетические данные с

	помощью make_classification (1000 образцов, 10 признаков, 2 класса, дисбаланс 0.95). 2. Разделить на train/test (70/30). 3. Обучить LogisticRegression с настройками по умолчанию. 4. Вычислить accuracy и сравнить с долей мажоритарного класса – почему accuracy вводит в заблуждение? 5. Вычислить precision, recall, F1 для миноритарного класса. 6. Построить confusion matrix. 7. Построить ROC-кривую и вычислить AUC-ROC. 8. Построить Precision-Recall кривую. 9. Подобрать порог, максимизирующий F1-меру. 10. Сравнить метрики при новом пороге с исходными.
7	1. Загрузить датасет spambase (UCI) через fetch_openml. 2. Разделить данные на train/test (80/20) со стратификацией. 3. Масштабировать признаки (StandardScaler). 4. Обучить LogisticRegression(penalty='l1', solver='saga', C=1.0) – L1-регуляризация. 5. Вывести количество ненулевых коэффициентов (отбор признаков). 6. Обучить модель с penalty='l2', сравнить количество ненулевых коэффициентов. 7. Сравнить accuracy, precision, recall для обеих моделей. 8. Построить ROC-кривые на одном графике. 9. Сравнить AUC-ROC. 10. Сделать вывод о влиянии типа регуляризации на качество и интерпретируемость.
8	1. Загрузить датасет breast_cancer из sklearn. 2. Разделить на train/test (75/25) со стратификацией. 3. Обучить LogisticRegression с C=0.1, C=1, C=10. 4. Для каждого значения C вывести accuracy и AUC-ROC. 5. Построить график зависимости метрик от C (логарифмическая шкала). 6. Выбрать оптимальное C по кросс-валидации (GridSearchCV). 7. Обучить модель с оптимальным C, оценить на тесте. 8. Построить confusion matrix для лучшей модели. 9. Вывести коэффициенты и определить самые важные признаки. 10. Сохранить лучшую модель в joblib.
9	1. Загрузить датасет adult (fetch_openml) для задачи прогнозирования дохода (>50K). 2. Провести предобработку: заменить '?' на NaN, удалить строки с пропусками.

	3. Выбрать признаки: age, education-num, hours-per-week, sex, race, marital-status. 4. Применить One-Hot Encoding к категориальным признакам. 5. Разделить на train/test (80/20) со стратификацией. 6. Обучить логистическую регрессию. 7. Построить ROC-кривую, вычислить AUC-ROC. 8. Проанализировать коэффициенты: какой признак вносит наибольший вклад в предсказание высокого дохода? 9. Построить confusion matrix и вычислить precision и recall для класса >50K. 10. Сделать вывод о том, насколько хорошо модель предсказывает высокий доход.
10	1. Загрузить датасет student (оценки студентов, целевая переменная – отчисление или нет) из открытого источника. 2. Провести полную предобработку (заполнение пропусков, кодирование категориальных). 3. Разделить на train/test (70/30) со стратификацией. 4. Обучить логистическую регрессию. 5. Построить confusion matrix. 6. Рассчитать метрики: accuracy, precision, recall, F1. 7. Построить ROC-кривую и вычислить AUC-ROC. 8. Построить калибровочную кривую (calibration curve) – насколько вероятности соответствуют реальным долям. 9. Использовать calibrated_classifier_cv для улучшения калибровки. 10. Сравнить калибровку до и после, сделать вывод.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Запишите формулу логистической функции (сигмоиды). Почему она подходит для бинарной классификации?
2. Как интерпретировать коэффициенты логистической регрессии? Что означают положительные и отрицательные значения?
3. Что такое confusion matrix для бинарной классификации? Что означает каждая из четырёх ячеек (TP, TN, FP, FN)?
4. В чём разница между precision и recall? В каких задачах важнее precision, а в каких recall? Приведите примеры.
5. Что такое F1-мера и почему она считается гармоническим средним precision и recall?
6. Что показывает ROC-кривая? Как интерпретировать AUC-ROC (значение 0.5, 0.8, 1.0)?
7. Как выбрать оптимальный порог принятия решения, отличный от 0.5? Какой критерий можно использовать?
8. Почему accuracy может быть плохой метрикой при дисбалансе классов? Приведите пример.
9. Как логистическая регрессия связана с линейной регрессией? В чём ключевое отличие?
10. Как параметр регуляризации `C` в `LogisticRegression` влияет на модель (большие и малые значения)? Что делает `class_weight='balanced'`?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность вычисления метрик. Корректный расчёт accuracy, precision, recall, F1, AUC-ROC.	10
Качество визуализаций (confusion matrix, ROC-кривая). Наличие подписей, легенд, читаемость, диагональ на ROC.	10
Ответы на контрольные вопросы (10 вопросов).	10

Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ метрик, интерпретация коэффициентов, обоснование выбора порога (если применимо).	10
Работа с дисбалансом (по варианту). Корректное применение class_weight или подбора порога, сравнение с базовым вариантом.	10

Лабораторная работа № 10

Логистическая регрессия для многоклассовой классификации

Цель: Расширить применение логистической регрессии на задачи с несколькими классами.

Теоретическое обоснование

Стратегии многоклассовой классификации:

- **One-vs-Rest (OvR) (или One-vs-All):** обучается K бинарных классификаторов, каждый отделяет свой класс от всех остальных. Предсказание — класс с максимальной вероятностью (или максимальным откликом). Является стандартной стратегией в sklearn при multi_class='ovr'.
- **One-vs-One (OvO):** обучается K(K-1)/2 классификаторов для каждой пары классов. Предсказание — класс, набравший большее число голосов в парных дуэлях. Вычислительно дороже, но может быть точнее для некоторых моделей (например, SVM).
- **Multinomial (кросс-энтропийная) логистическая регрессия:** обобщение сигмоиды на K классов — softmax: p_k = e^{z_k} / Σ e^{z_j}. Обучается единая модель. В sklearn multi_class='multinomial' (требует solver='lbfgs' или 'newton-cg').

Метрики для многоклассовой классификации:

- **Macro F1** — среднее арифметическое F1 по всем классам (каждый класс имеет равный вес, независимо от его размера).
- **Micro F1** — глобальный F1, вычисленный по сумме TP, FP, FN по всем классам. Близок к accuracy.

- **Weighted F1** — среднее F1 с весами, пропорциональными поддержке (числу объектов) каждого класса. Компромисс между macro и micro.

Матрица ошибок для K классов имеет размер K×K. Диагональ — правильные предсказания. Строки — истинные классы, столбцы — предсказанные. Анализ матрицы позволяет выявить, какие классы часто путаются.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет Iris из sklearn.datasets.load_iris(). 2. Разделить данные на обучающую (70%) и тестовую (30%) выборки со стратификацией по целевому признаку. 3. Обучить LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=200) (One-vs-Rest). 4. Вычислить accuracy на тестовой выборке. 5. Вычислить macro F1-score и micro F1-score. 6. Построить матрицу ошибок (confusion matrix) в виде тепловой карты. 7. Обучить модель со стратегией multi_class='multinomial' (кросс-энтропийная). 8. Сравнить accuracy и macro F1 для OvR и multinomial. 9. Вывести коэффициенты модели (веса) для каждого класса – сколько признаков, какая размерность. 10. Сохранить лучшую модель в файл iris_logreg.pkl.
2	1. Загрузить датасет Wine из sklearn.datasets.load_wine(). 2. Разделить на train/test (75/25) со стратификацией. 3. Масштабировать признаки с помощью StandardScaler. 4. Обучить LogisticRegression(multi_class='ovr', solver='lbfgs', max_iter=500). 5. Вычислить weighted F1-score (учитывает поддержку классов). 6. Построить confusion matrix и нормализовать её по строкам (доля правильных предсказаний для каждого класса). 7. С помощью cross_val_score (5 фолдов) оценить стабильность accuracy. 8. Обучить модель со стратегией multi_class='multinomial', сравнить метрики. 9. Определить, для каких классов модель ошибается чаще всего (по матрице ошибок). 10. Сделать вывод о применимости логистической регрессии к

	этому датасету.
3	1. Загрузить датасет Digits (8×8 изображения) из sklearn.datasets.load_digits(). 2. Разделить на train/test (80/20) со стратификацией. 3. Применить PCA для снижения размерности до 20 компонент (сохранив 95% дисперсии). 4. Обучить логистическую регрессию с multi_class='ovr' на преобразованных данных. 5. Вычислить accuracy и macro F1. 6. Построить матрицу ошибок. 7. Обучить модель без PCA (на исходных 64 признаках) и сравнить время обучения и качество. 8. Использовать LogisticRegression(multi_class='multinomial', penalty='l2', C=0.1). 9. Сравнить micro и macro F1. 10. Визуализировать веса модели для каждого класса (изображения 8×8).
4	1. Загрузить датасет glass (идентификация типа стекла) из openml (fetch_openml('glass', version=1)). 2. Целевая переменная содержит 7 классов, есть дисбаланс. Вывести распределение классов. 3. Разделить на train/test (70/30) со стратификацией. 4. Обучить логистическую регрессию с multi_class='ovr' и class_weight='balanced'. 5. Вычислить accuracy, macro F1, weighted F1. 6. Построить confusion matrix. 7. Обучить модель без class_weight, сравнить метрики для миноритарных классов. 8. Использовать GridSearchCV для подбора C (0.01, 0.1, 1, 10) и стратегии ('ovr', 'multinomial'). 9. Вывести лучшие параметры и качество. 10. Сохранить лучшую модель и результаты поиска в CSV.
5	1. Загрузить датасет MNIST (только первые 1000 образцов для скорости) через fetch_openml('mnist_784', version=1). 2. Разделить на train/test (80/20). 3. Обучить LogisticRegression(multi_class='ovr', solver='saga', max_iter=1000) – поскольку признаков много. 4. Вычислить accuracy на тесте. 5. Построить confusion matrix (можно в виде изображения с цветовой шкалой). 6. Вычислить macro и micro F1.

	7. Использовать cross_val_score с 3 фолдами (из-за ресурсоёмкости) для оценки стабильности. 8. Обучить модель с penalty='l1' (saga) – посмотреть, сколько признаков обнулилось. 9. Сравнить accuracy и время обучения L1 и L2. 10. Сделать вывод о влиянии разреженности признаков.
6	1. Загрузить датасет Fashion-MNIST (только 2000 образцов) через fetch_openml('Fashion-MNIST', version=1). 2. Нормализовать пиксели (разделить на 255). 3. Разделить на train/test (75/25). 4. Обучить логистическую регрессию с multi_class='multinomial' и solver='lbfgs'. 5. Вывести accuracy. 6. Построить матрицу ошибок и определить, какие категории одежды чаще путаются. 7. Вычислить precision, recall, F1 для каждого класса в отдельности (вывести таблицу). 8. Построить ROC-кривые для каждого класса в стиле One-vs-Rest (можно для первых 3 классов). 9. Использовать RandomizedSearchCV для подбора C и penalty. 10. Сохранить лучшую модель в joblib.
7	1. Сгенерировать синтетический датасет с 3 классами с помощью make_blobs(n_samples=300, centers=3, cluster_std=1.5). 2. Визуализировать данные (scatter plot, цвет по классам). 3. Разделить на train/test (80/20). 4. Обучить логистическую регрессию с multi_class='ovr'. 5. Визуализировать разделяющие границы (для двух признаков) – построить график с областями решений. 6. Вычислить accuracy и confusion matrix. 7. Обучить модель с multi_class='multinomial', визуализировать границы. 8. Сравнить визуально и по метрикам. 9. Изменить параметр C на 0.01 и на 100, визуализировать изменение границ. 10. Сделать вывод о влиянии регуляризации на сложность границ.
8	1. Загрузить датасет yeast (предсказание локализации белков) из openml. 2. Вывести количество классов и распределение. 3. Разделить на train/test (70/30) со стратификацией. 4. Обучить логистическую регрессию с multi_class='ovr', затем

	с multinomial. 5. Вычислить для обеих моделей accuracy, macro F1, weighted F1. 6. Построить confusion matrix для лучшей модели. 7. Определить класс с самым низким recall, предложить способ улучшения. 8. Применить class_weight='balanced' для OvR и сравнить macro F1. 9. Использовать OneVsOneClassifier из sklearn (явно) и сравнить с OvR. 10. Сохранить таблицу сравнения метрик в DataFrame.
9	1. Загрузить датасет segment (сегментация изображений) из openml. 2. Разделить на train/test (80/20). 3. Масштабировать признаки с помощью RobustScaler. 4. Обучить LogisticRegression(multi_class='multinomial', max_iter=1000). 5. Построить confusion matrix, нормализованную по строкам. 6. Вычислить precision, recall, F1 для каждого класса (вывести в таблице). 7. Построить график важности признаков для каждого класса (абсолютные значения коэффициентов). 8. Использовать cross_val_predict для получения предсказаний на всех данных и построить матрицу ошибок. 9. Сравнить время обучения и предсказания с KNeighborsClassifier(n_neighbors=5). 10. Сделать вывод о том, когда логистическая регрессия предпочтительнее.
10	1. Загрузить датасет letter (распознавание букв от A до Z) из openml (только 5000 образцов). 2. Целевая переменная – 26 классов. Разделить на train/test (80/20). 3. Обучить LogisticRegression(multi_class='ovr', solver='saga', max_iter=500). 4. Вычислить accuracy – ожидается не очень высокая из-за сложности. 5. Построить confusion matrix (можно в виде изображения 26×26). 6. Вычислить macro и micro F1. 7. Определить самые «трудные» буквы (те, которые часто путаются). 8. Применить PCA с 30 компонентами и повторить обучение – сравнить accuracy и время.

	9. Использовать GridSearchCV для подбора C (0.01, 0.1, 1, 10) и solver ('saga', 'lbfgs'). 10. Сохранить лучшую модель и список признаков (коэффициенты) в файл.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Как логистическая регрессия обобщается на случай многоклассовой классификации? Какие две основные стратегии существуют?
2. В чём суть стратегии One-vs-Rest (OvR)? Сколько классификаторов обучается для K классов?
3. В чём суть стратегии One-vs-One (OvO)? Сколько классификаторов обучается и как принимается итоговое решение?
4. Что такое multinomial логистическая регрессия (кросс-энтропийная) и чем она отличается от OvR?
5. Какая метрика называется macro F1-score? Как она вычисляется и когда её следует использовать?
6. Чем macro F1 отличается от micro F1? В каких случаях micro F1 будет близок к accuracy?
7. Что такое weighted F1-score и для чего он нужен при дисбалансе классов?
8. Как интерпретировать матрицу ошибок (confusion matrix) для многоклассовой классификации? Что показывает элемент на позиции (i, j)?

9. Можно ли использовать ROC-кривую для многоклассовой классификации? Если да, то как?
10. Какие параметры LogisticRegression в sklearn отвечают за выбор стратегии многоклассовой классификации и тип регуляризации?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора стратегии многоклассовой классификации. Обоснование использования OvR или multinomial, понимание различий.	10
Корректное вычисление метрик (macro/micro/weighted F1). Правильные формулы и интерпретация.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация матрицы ошибок. Анализ ошибок, определение проблемных классов, сравнение стратегий.	10
Визуализации (матрица ошибок, разделяющие границы, ROC для многоклассового случая). Качество графиков, подписи, легенды.	10

Лабораторная работа № 11

Метод k-ближайших соседей (kNN)

Цель: Реализовать и настроить классификатор на основе kNN.

Теоретическое обоснование

Принцип работы. Для классификации: объект относится к тому классу, который преобладает среди его k ближайших соседей в пространстве признаков. Для регрессии: предсказание — среднее (или медианное) значение целевой переменной соседей.

Метрики расстояния (параметр p в Minkowski):

- p=1: Манхэттенское (L1) — сумма модулей разностей.
- p=2: Евклидово (L2) — корень из суммы квадратов.
- p→∞: Чебышёва (max разность).

Гиперпараметры:

- n_neighbors (k): малое k → высокая сложность, чувствительность к шуму, переобучение. Большое k → сглаживание, потеря локальных особенностей, недообучение.
- weights: 'uniform' (все соседи равны), 'distance' (вклад обратно пропорционален расстоянию).
- algorithm: 'brute' (прямой перебор), 'kd_tree' (для размерности < 20), 'ball_tree' (общий случай). Выбор влияет на скорость.

Влияние масштабирования. kNN основан на расстояниях. Если один признак имеет диапазон [0, 1000], а другой [0, 1], то первый будет доминировать. **Обязательно масштабирование** (StandardScaler, MinMaxScaler).

Проклятие размерности. В высоких размерностях расстояния между точками становятся почти одинаковыми, и понятие «близости» теряет смысл. Для kNN эффективная размерность не должна превышать 20–30.

Выбор k обычно выполняется по кросс-валидации (GridSearchCV). Для несбалансированных классов полезны взвешенные метрики.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris из sklearn. Разделить на train/test (70/30) со стратификацией. 2. Обучить KNeighborsClassifier(n_neighbors=3) на исходных (немасштабированных) данных. Вычислить accuracy. 3. Применить StandardScaler к признакам, повторить обучение. Сравнить accuracy. 4. Использовать GridSearchCV для подбора n_neighbors от 1 до 15 (5 фолдов). Найти оптимальное k. 5. Вывести лучший параметр и соответствующую accuracy. 6. Обучить модель с оптимальным k на масштабированных данных. Построить confusion matrix. 7. Вычислить precision, recall, F1 для каждого класса (macro и weighted). 8. Изменить метрику расстояния на манхэттенскую (p=1) и сравнить accuracy с евклидовой (p=2). 9. Построить график зависимости accuracy от k (1–20) на валидации (из GridSearchCV). 10. Сохранить лучшую модель в файл.
2	1. Загрузить датасет wine из sklearn. Разделить на train/test (75/25) со стратификацией. 2. Масштабировать данные с помощью MinMaxScaler. 3. Обучить kNN с n_neighbors=5. Оценить accuracy и F1-macro. 4. Использовать RandomizedSearchCV для подбора n_neighbors (1–30) и weights (uniform, distance). 5. Вывести лучшие параметры. 6. Обучить модель с лучшими параметрами, сравнить с базовой (k=5, uniform). 7. Построить матрицу ошибок. 8. Исследовать влияние метрики: metric='chebyshev' и сравнить с евклидовой. 9. Визуализировать зависимость accuracy от k для uniform и distance весов (на одном графике). 10. Сделать вывод о том, как веса влияют на оптимальное k.
3	1. Загрузить датасет digits (8×8) из sklearn. Разделить на train/test (80/20). 2. Нормализовать пиксели (разделить на 16, максимум). 3. Обучить kNN с n_neighbors=3. Вычислить accuracy. 4. Построить матрицу ошибок (8×8) в виде тепловой карты. 5. Определить, какие цифры чаще всего путаются. 6. Использовать GridSearchCV для подбора n_neighbors (1–15)

	и algorithm (auto, ball_tree, kd_tree). 7. Сравнить время обучения для разных алгоритмов. 8. Выбрать лучшую модель, оценить на тесте. 9. Построить график зависимости accuracy от k для разных алгоритмов. 10. Сохранить предсказания для тестовой выборки в CSV.
4	1. Сгенерировать синтетические данные с помощью make_classification(n_samples=500, n_features=2, n_classes=3, n_clusters_per_class=1, random_state=42). 2. Визуализировать данные (scatter plot, цвет по классам). 3. Разделить на train/test (80/20). 4. Обучить kNN с n_neighbors=5 на немасштабированных данных. Визуализировать границы решений. 5. Применить StandardScaler, повторить визуализацию границ. Сравнить. 6. Подобрать оптимальное k с помощью кросс-валидации (1–20). 7. Построить график границ решений для оптимального k. 8. Использовать weights='distance' и сравнить границы с uniform. 9. Визуализировать влияние параметра p (1, 2, 3) на границы. 10. Сделать вывод о влиянии масштабирования на kNN.
5	1. Загрузить датасет breast_cancer из sklearn. Разделить на train/test (70/30) со стратификацией. 2. Масштабировать с помощью RobustScaler. 3. Обучить kNN с n_neighbors=5. Вычислить accuracy, precision, recall, F1. 4. Использовать GridSearchCV для подбора n_neighbors (1–25) и leaf_size (10, 30, 50). 5. Вывести лучшие параметры. 6. Обучить модель с лучшими параметрами, сравнить с базовой. 7. Построить ROC-кривую и вычислить AUC-ROC (для бинарной классификации). 8. Использовать cross_val_score с 10 фолдами для оценки стабильности лучшей модели. 9. Сравнить качество kNN с логистической регрессией на этом датасете. 10. Сохранить лучшую модель в joblib.
6	1. Загрузить датасет titanic из Seaborn, провести предобработку (заполнение пропусков, one-hot кодирование). 2. Разделить на train/test (80/20) со стратификацией. 3. Масштабировать числовые признаки (age, fare) с помощью StandardScaler.

	4. Обучить kNN с n_neighbors=5. Вычислить accuracy, precision, recall, F1. 5. Построить confusion matrix. 6. Использовать GridSearchCV для подбора n_neighbors (1–20) и p (1, 2). 7. Найти лучшую модель, оценить на тесте. 8. Сравнить с логистической регрессией (по F1-мере). 9. Построить ROC-кривую для kNN и логистической регрессии на одном графике. 10. Сделать вывод о том, какой алгоритм лучше для данного датасета.
7	1. Загрузить датасет fruits (изображения фруктов или табличные данные) из открытого источника. 2. Разделить на train/test (75/25). 3. Масштабировать данные (MinMaxScaler). 4. Обучить kNN с n_neighbors=3. Оценить accuracy. 5. Использовать cross_val_score для оценки среднего качества при k=3,5,7,9. 6. Подобрать оптимальное k по максимуму cross-val accuracy. 7. Для оптимального k вычислить матрицу ошибок и классификационный отчёт. 8. Эксперимент: добавить шумовые признаки (случайные числа) – как изменится accuracy? 9. Применить PCA для снижения размерности до 2 компонент и визуализировать данные с цветом классов и предсказаниями kNN. 10. Сохранить результаты эксперимента в CSV.
8	1. Сгенерировать данные с помощью make_moons(n_samples=300, noise=0.1). 2. Визуализировать данные. 3. Разделить на train/test (80/20). 4. Обучить kNN с n_neighbors=3 на исходных данных, визуализировать границы решений. 5. Подобрать оптимальное k (1–20) по кросс-валидации. 6. Визуализировать границы для оптимального k. 7. Увеличить шум до 0.3 – как меняется оптимальное k? 8. Использовать weights='distance' при том же шуме, сравнить границы. 9. Построить график зависимости accuracy от k для разных уровней шума. 10. Сделать вывод о робастности kNN к шуму.

9	1. Загрузить датасет mnist_784 (только 2000 образцов). Разделить на train/test (70/30). 2. Нормализовать пиксели. 3. Обучить kNN с n_neighbors=5. Измерить время предсказания. 4. Использовать algorithm='kd_tree' и сравнить время с brute. 5. Подобрать оптимальное k (1–15) с учётом времени выполнения. 6. Применить PCA для снижения размерности до 50 компонент. Обучить kNN на сокращённых данных. 7. Сравнить accuracy и время предсказания с исходными данными. 8. Построить график зависимости accuracy от числа компонент PCA (10, 20, 50, 100). 9. Использовать BallTree и сравнить скорость с KDTree. 10. Сохранить лучшую модель (с PCA) в файл.
10	1. Загрузить датасет vehicle (классификация транспортных средств) из openml. 2. Разделить на train/test (80/20). 3. Масштабировать с помощью StandardScaler. 4. Обучить kNN с n_neighbors=5. Вычислить accuracy и macro F1. 5. Использовать GridSearchCV для подбора n_neighbors (1–20), weights (uniform, distance), p (1,2). 6. Вывести лучшие параметры. 7. Для лучшей модели построить матрицу ошибок. 8. Сравнить качество с RadiusNeighborsClassifier (радиусный kNN). 9. Построить график зависимости accuracy от радиуса (для RadiusNeighbors). 10. Сделать вывод о том, когда kNN лучше, чем RadiusNeighbors.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие

трудности, анализ результатов).

5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Опишите принцип работы метода k-ближайших соседей (kNN) для классификации.
2. Какие метрики расстояния можно использовать в kNN? Чем отличается евклидово расстояние от манхэттенского?
3. Как выбор параметра k влияет на склонность модели к переобучению? Что будет при k=1 и при k=N?
4. Почему для kNN критически важно масштабирование признаков? Приведите пример.
5. Что означает параметр `weights='distance'` в sklearn? Как это изменяет предсказание?
6. Какие алгоритмы поиска соседей реализованы в sklearn (brute, kd_tree, ball_tree)? В каких случаях каждый из них предпочтительнее?
7. Как подобрать оптимальное значение k? Какие методы используются?
8. Как влияет размерность пространства признаков на эффективность kNN (проклятие размерности)?
9. Может ли kNN использоваться для регрессии? Если да, как он работает?
10. Каковы основные преимущества и недостатки метода kNN по сравнению с логистической регрессией?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность подбора гиперпараметров (k, weights, metric). Использование кросс-валидации, обоснование выбора.	10
Демонстрация влияния масштабирования. Сравнение accuracy до и после масштабирования, визуализация границ.	10
Ответы на контрольные вопросы (10 вопросов).	10

Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ графиков, объяснение оптимального k, сравнение с другими моделями (если применимо).	10
Визуализации (границы решений, графики зависимости). Качество, читаемость, подписи.	10

Лабораторная работа № 12

Решающие деревья: построение и визуализация

Цель: Построить, визуализировать и интерпретировать решающее дерево.

Теоретическое обоснование

Структура. Дерево решений — бинарное дерево, где каждый внутренний узел проверяет значение одного признака ($x_i < t$ или $x_i \geq t$), а листья содержат предсказание класса (или среднее значение для регрессии).

Критерии разделения (impurity measures):

- **Gini impurity:** $G = 1 - \Sigma\, p\_k^2$, где p_k — доля класса k в узле. Минимизируется.
- **Энтропия (Shannon):** $H = - \Sigma\, p\_k \cdot \log_2(p\_k)$. Часто даёт похожие результаты.
- Для регрессии: MSE (среднеквадратичная ошибка) или MAE.

Алгоритм построения (CART). Рекурсивно: для каждого узла перебираются все признаки и все возможные пороги, выбирается тот, который даёт максимальное уменьшение impurity (information gain). Рост прекращается, когда достигнута максимальная глубина, минимальное число объектов в узле или все объекты одного класса.

Гиперпараметры для регуляризации:

- max_depth — максимальная глубина. Малая глубина → недообучение, большая → переобучение.
- min_samples_split — минимальное число объектов для разбиения узла.
- min_samples_leaf — минимальное число объектов в листе.
- max_features — число признаков, рассматриваемых при каждом разбиении (увеличивает случайность).

Важность признаков вычисляется как суммарное уменьшение impurity по всем узлам, где использовался признак, взвешенное на число объектов в узле. Сумма важностей всех признаков равна 1.

Визуализация: plot_tree (sklearn), export_graphviz (Graphviz), export_text (текстовое представление).

Преимущества: интерпретируемость, не требует масштабирования, работа с категориями. **Недостатки:** склонность к переобучению, нестабильность (малое изменение данных → другое дерево).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris из sklearn. Разделить на train/test (70/30) со стратификацией. 2. Обучить DecisionTreeClassifier(random_state=42) с настройками по умолчанию. Вычислить accuracy. 3. Визуализировать дерево с помощью plot_tree (подписать признаки, классы). 4. Вывести важность признаков (feature_importances_) в виде столбчатой диаграммы. 5. Ограничить глубину дерева max_depth=3 и снова визуализировать. Сравнить accuracy. 6. Использовать GridSearchCV для подбора max_depth (2–10) и min_samples_split (2,5,10). 7. Вывести лучшие параметры и accuracy лучшей модели. 8. Построить матрицу ошибок для лучшей модели. 9. Сравнить важность признаков в лучшей модели с исходной. 10. Сохранить лучшее дерево в DOT-формате (export_graphviz).
2	1. Загрузить датасет wine из sklearn. Разделить на train/test (75/25) со стратификацией. 2. Обучить DecisionTreeClassifier(criterion='entropy', random_state=42). 3. Визуализировать дерево (можно уменьшенное, если большое). 4. Вычислить accuracy и F1-macro. 5. Исследовать влияние max_depth от 1 до 10 на accuracy (построить график). 6. Использовать min_samples_leaf (1,3,5,10) – сравнить accuracy. 7. Обучить модель с max_features='sqrt' и сравнить с использованием всех признаков. 8. Построить confusion matrix в виде тепловой карты. 9. Проанализировать важность признаков – какой признак самый важный? 10. Сделать вывод о том, насколько дерево интерпретируемо.
3	1. Загрузить датасет titanic из Seaborn, провести предобработку (заполнение пропусков, one-hot). 2. Разделить на train/test (80/20) со стратификацией. 3. Обучить DecisionTreeClassifier(max_depth=4, random_state=42). 4. Визуализировать дерево с поворотом меток (rotated). 5. Вычислить accuracy, precision, recall, F1 для класса «выжил».

	6. Построить ROC-кривую и вычислить AUC-ROC. 7. Использовать GridSearchCV для подбора max_depth (2–8), min_samples_split (2,5,10), criterion (gini, entropy). 8. Вывести лучшую модель и её параметры. 9. Сравнить важность признаков в лучшей модели с логистической регрессией (коэффициенты). 10. Сохранить лучшую модель в joblib.
4	1. Сгенерировать синтетические данные с помощью make_moons(n_samples=300, noise=0.2). 2. Разделить на train/test (80/20). 3. Обучить DecisionTreeClassifier(max_depth=2). Визуализировать границы решений (для двух признаков). 4. Обучить дерево с max_depth=10. Визуализировать границы – показать переобучение. 5. Построить график зависимости accuracy на тесте от max_depth (1–15). 6. Подобрать оптимальную глубину по кросс-валидации. 7. Визуализировать дерево оптимальной глубины. 8. Использовать min_samples_leaf=5 и сравнить границы решений. 9. Показать важность признаков (должны быть оба признака важны). 10. Сделать вывод о том, как глубина влияет на сложность границы.
5	1. Загрузить датасет breast_cancer из sklearn. Разделить на train/test (70/30) со стратификацией. 2. Обучить DecisionTreeClassifier(random_state=42). Вычислить accuracy, precision, recall. 3. Визуализировать дерево (можно в виде текста через export_text). 4. Вычислить важность признаков – найти топ-3 самых важных. 5. Использовать RandomizedSearchCV для подбора max_depth (2–20), min_samples_split (2–20), min_samples_leaf (1–10). 6. Обучить лучшую модель, сравнить accuracy с исходной. 7. Построить confusion matrix. 8. Построить ROC-кривую и вычислить AUC-ROC. 9. Сравнить важность признаков до и после настройки. 10. Сохранить лучшее дерево в PNG.
6	1. Загрузить датасет diabetes (бинарная классификация) из sklearn. 2. Разделить на train/test (75/25).

	3. Обучить DecisionTreeClassifier(max_depth=3, random_state=42). 4. Визуализировать дерево, интерпретировать правила (какие признаки на каких уровнях). 5. Вычислить accuracy, F1. 6. Использовать cost_complexity_pruning (CCP) – построить график зависимости accuracy от ccp_alpha. 7. Подобрать оптимальный ccp_alpha с помощью кросс-валидации. 8. Обучить дерево с оптимальным ccp_alpha, сравнить с исходным. 9. Построить график важности признаков для обрезанного дерева. 10. Сделать вывод о эффективности обрезки.
7	1. Загрузить датасет heart_disease из openml. Провести предобработку. 2. Разделить на train/test (80/20) со стратификацией. 3. Обучить DecisionTreeClassifier(max_depth=5). 4. Визуализировать дерево (использовать graphviz для красивого отображения). 5. Вычислить precision, recall, F1. 6. Построить матрицу ошибок. 7. Использовать GridSearchCV для подбора criterion, max_depth, min_samples_leaf. 8. Сравнить лучшую модель с логистической регрессией (по AUC-ROC). 9. Проанализировать, какие признаки оказались важными, соответствуют ли они медицинским знаниям. 10. Сохранить лучшее дерево в PDF.
8	1. Загрузить датасет seeds (классификация зерен пшеницы) из openml. 2. Разделить на train/test (70/30) со стратификацией. 3. Обучить DecisionTreeClassifier(max_depth=4). 4. Построить тепловую карту важности признаков. 5. Визуализировать дерево и записать одно из правил в виде «если … то …». 6. Использовать cross_val_score для оценки стабильности дерева (5 фолдов). 7. Сравнить со случайным лесом из 10 деревьев (RandomForestClassifier). 8. Построить график зависимости важности признаков от max_depth.

	9. Определить, какой признак является самым информативным. 10. Сделать вывод о том, почему дерево может давать нестабильные предсказания.
9	1. Загрузить датасет banknote (аутентификация банкнот) из openml. 2. Разделить на train/test (80/20). 3. Обучить DecisionTreeClassifier(criterion='gini', max_depth=5). 4. Визуализировать дерево. 5. Вычислить accuracy и построить ROC-кривую. 6. Использовать export_graphviz с feature_names и class_names. 7. Подобрать max_depth по кросс-валидации (от 2 до 15). Построить график. 8. Обучить дерево регрессии (DecisionTreeRegressor) на том же датасете (преобразовав метки в 0/1) – сравнить. 9. Проанализировать, насколько глубина влияет на интерпретируемость. 10. Сохранить лучшее дерево в текстовом виде (export_text).
10	1. Сгенерировать данные с помощью make_classification(n_samples=500, n_features=5, n_informative=3, n_redundant=2). 2. Разделить на train/test (75/25). 3. Обучить DecisionTreeClassifier(max_depth=10). 4. Вывести важность признаков – определить, какие признаки информативные, а какие шумовые. 5. Визуализировать дерево (можно только верхние уровни). 6. Использовать min_samples_split=20 и сравнить важность признаков. 7. Построить график зависимости accuracy на тесте от min_samples_split (2–50). 8. Применить max_features=3 и сравнить с использованием всех признаков. 9. Построить матрицу ошибок для лучшей модели. 10. Сделать вывод о том, как параметры регуляризации дерева влияют на качество.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания

с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.

4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Как работает решающее дерево для классификации? Опишите процесс построения.
2. Что такое критерии разделения (Gini impurity, энтропия)? Какой из них чаще используется и почему?
3. Что показывает важность признаков (feature_importances_) в решающем дереве? Как она вычисляется?
4. Как параметр max_depth влияет на склонность дерева к переобучению?
5. В чём разница между min_samples_split и min_samples_leaf?
6. Что такое max_features и зачем его ограничивать?
7. Как визуализировать решающее дерево в sklearn? Какие инструменты можно использовать?
8. Что такое cost-complexity pruning (ccp_alpha)? Как он помогает бороться с переобучением?
9. Каковы основные преимущества решающих деревьев (интерпретируемость) и недостатки (нестабильность)?
10. Можно ли использовать решающие деревья для регрессии? Если да, то какой критерий разделения используется?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Качество визуализации дерева. Читаемость, подписи,	10

использование plot_tree или graphviz.	
Правильность подбора гиперпараметров. Использование кросс-валидации, анализ влияния параметров на качество.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ важности признаков, интерпретация правил, обоснование выбора глубины.	10
Сравнение с другими моделями (если по заданию). Сравнение с логистической регрессией, случайным лесом или другими.	10

Лабораторная работа № 13

Ансамблевые методы: Random Forest

Цель: Применить Random Forest и сравнить его качество с одиночным деревом.

Теоретическое обоснование

Идея случайного леса. Строится множество решающих деревьев, каждое на своей случайной подвыборке данных и случайном подмножестве признаков. Предсказание — усреднение (регрессия) или голосование (классификация).

Два источника случайности:

1. **Bootstrap** — из исходного набора с возвращением выбирается N объектов (где N — размер выборки). Около 37% объектов не попадают в выборку (out-of-bag, OOB).
2. **Random subspace** — при каждом разбиении узла рассматривается случайное подмножество из max_features признаков (обычно sqrt(p) для классификации, p/3 для регрессии).

Преимущества перед одиночным деревом:

- Снижает дисперсию, устойчив к переобучению.
- Не требует тщательной настройки (работает хорошо «из коробки»).
- OOB-оценка — внутренняя кросс-валидация (средняя ошибка по тем объектам, которые не вошли в bootstrap-выборку дерева).

Гиперпараметры:

- n_estimators — число деревьев. Чем больше, тем лучше, но дольше. Сходимость обычно после 100–200.
- max_depth — ограничение глубины (по умолчанию без ограничений).
- min_samples_split, min_samples_leaf — как в дереве.
- max_features — число признаков для разбиения.
- bootstrap — использовать ли бутстреп (обычно True).

Важность признаков в случайном лесе усредняется по всем деревьям, что даёт более надёжные оценки, чем в одном дереве.

Сравнение с градиентным бустингом: случайный лес менее склонен к переобучению, проще в настройке, но часто уступает бустингу по точности на

больших данных.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris из sklearn. Разделить на train/test (70/30) со стратификацией. 2. Обучить RandomForestClassifier(n_estimators=100, random_state=42). Вычислить accuracy. 3. Обучить одиночное решающее дерево (DecisionTreeClassifier). Сравнить accuracy. 4. Вывести важность признаков для случайного леса (построить горизонтальную столбчатую диаграмму). 5. Вычислить OOB-ошибку (oob_score=True) и сравнить с accuracy на тесте. 6. Исследовать влияние n_estimators (10, 50, 100, 200, 500) на accuracy – построить график. 7. Использовать GridSearchCV для подбора max_depth (3–10) и min_samples_split (2,5,10). 8. Вывести лучшие параметры и accuracy. 9. Построить матрицу ошибок для лучшей модели. 10. Сохранить лучшую модель в joblib.
2	1. Загрузить датасет wine из sklearn. Разделить на train/test (75/25) со стратификацией. 2. Обучить RandomForestClassifier(n_estimators=100, random_state=42). 3. Сравнить F1-macro со случайным лесом и одиночным деревом. 4. Визуализировать важность признаков – какой признак самый важный? 5. Использовать RandomForestClassifier(oob_score=True). Вывести OOB-score. 6. Подобрать оптимальное n_estimators через validation_curve (от 10 до 300). 7. Исследовать влияние max_features (auto, sqrt, log2, 0.5) на accuracy. 8. Построить матрицу ошибок в виде тепловой карты. 9. Сравнить время обучения случайного леса и одиночного дерева. 10. Сделать вывод о преимуществах ансамбля.

3	1. Загрузить датасет titanic из Seaborn, провести предобработку (заполнение пропусков, one-hot). 2. Разделить на train/test (80/20) со стратификацией. 3. Обучить RandomForestClassifier(n_estimators=200, random_state=42). 4. Вычислить accuracy, precision, recall, F1. 5. Построить ROC-кривую и вычислить AUC-ROC. 6. Сравнить с логистической регрессией (по AUC-ROC). 7. Использовать GridSearchCV для подбора n_estimators (100,200,300) и max_depth (5,10,15). 8. Вывести лучшие параметры и качество. 9. Построить confusion matrix. 10. Сохранить лучшую модель и вывести важность признаков.
4	1. Загрузить датасет breast_cancer из sklearn. Разделить на train/test (70/30). 2. Обучить RandomForestClassifier(n_estimators=100, random_state=42). 3. Вычислить accuracy, precision, recall, F1. 4. Построить ROC-кривую. 5. Использовать RandomizedSearchCV для подбора n_estimators (50–500), max_depth (5–20), min_samples_split (2–10), min_samples_leaf (1–5). 6. Вывести лучшие параметры. 7. Обучить лучшую модель, сравнить с базовой (100 деревьев). 8. Построить график зависимости OOB-ошибки от n_estimators. 9. Сравнить важность признаков в случайном лесе и в одиночном дереве. 10. Сохранить лучшую модель.
5	1. Загрузить датасет digits из sklearn. Разделить на train/test (80/20). 2. Обучить RandomForestClassifier(n_estimators=100, random_state=42). 3. Вычислить accuracy и macro F1. 4. Построить матрицу ошибок (8×8). 5. Использовать cross_val_score с 5 фолдами для оценки стабильности. 6. Исследовать влияние bootstrap=False (без бутстрепа) – сравнить с bootstrap=True. 7. Подобрать max_features (4, 8, 16, 32, 64) с помощью кросс-валидации. 8. Построить график зависимости accuracy от max_features. 9. Визуализировать одно дерево из леса (любое) с

	помощью plot_tree. 10. Сделать вывод о роли случайности в случайном лесе.
6	1. Загрузить датасет california_housing (регрессия) из sklearn. Разделить на train/test (80/20). 2. Обучить RandomForestRegressor(n_estimators=100, random_state=42). Вычислить RMSE и R^2. 3. Сравнить с линейной регрессией (по R^2). 4. Вывести важность признаков для регрессии. 5. Использовать RandomForestRegressor(oob_score=True). Вывести OOB-R^2. 6. Подобрать n_estimators (50,100,200,300) и max_depth (5,10,15,20) с помощью GridSearchCV. 7. Вывести лучшие параметры и R^2. 8. Построить график предсказанных vs истинных значений. 9. Сравнить время обучения с градиентным бустингом (XGBoost). 10. Сохранить лучшую модель.
7	1. Сгенерировать синтетические данные с помощью make_classification(n_samples=1000, n_features=20, n_informative=10, n_redundant=10, random_state=42). 2. Разделить на train/test (75/25). 3. Обучить RandomForestClassifier(n_estimators=100). 4. Вывести важность признаков – определить, какие признаки информативные, а какие шумовые. 5. Использовать SelectFromModel для отбора признаков по важности (порог 0.02). 6. Обучить случайный лес только на отобранных признаках, сравнить accuracy. 7. Исследовать влияние max_samples (0.5, 0.7, 0.9, 1.0) на качество. 8. Построить график зависимости accuracy от max_samples. 9. Сравнить с ExtraTreesClassifier (экстремально случайные деревья). 10. Сделать вывод о роли подвыборки объектов.
8	1. Загрузить датасет penguins из Seaborn, удалить строки с пропусками, закодировать категориальные. 2. Разделить на train/test (80/20) со стратификацией. 3. Обучить RandomForestClassifier(n_estimators=150, random_state=42). 4. Вычислить accuracy и F1-macro. 5. Построить confusion matrix. 6. Использовать RandomForestClassifier(class_weight='balanced') дл

	я учёта дисбаланса (если есть). 7. Сравнить с моделью без class_weight. 8. Построить график важности признаков. 9. Визуализировать одно дерево из леса (ограничив глубину до 3 для читаемости). 10. Сохранить лучшую модель.
9	1. Загрузить датасет mnist_784 (только 2000 образцов). Разделить на train/test (70/30). 2. Обучить RandomForestClassifier(n_estimators=50, random_state=42). Замерить время обучения. 3. Обучить одиночное дерево, сравнить accuracy и время. 4. Использовать RandomForestClassifier(n_estimators=50, max_features='sqrt') – сравнить с max_features='auto'. 5. Построить график зависимости accuracy от n_estimators (10–200). 6. Вычислить OOB-ошибку и сравнить с accuracy на тесте. 7. Применить PCA для снижения размерности до 50 компонент, обучить случайный лес на PCA-данных. 8. Сравнить accuracy и время обучения с исходными данными. 9. Построить матрицу ошибок для лучшей модели. 10. Сделать вывод о масштабируемости случайного леса.
10	1. Загрузить датасет adult из openml (предсказание дохода >50K). 2. Провести предобработку (замена '?' на NaN, удаление пропусков, one-hot кодирование). 3. Разделить на train/test (80/20) со стратификацией. 4. Обучить RandomForestClassifier(n_estimators=100, n_jobs=-1, random_state=42). 5. Вычислить accuracy, precision, recall, F1 для класса >50K. 6. Построить ROC-кривую и вычислить AUC-ROC. 7. Использовать GridSearchCV для подбора n_estimators (100,200) и max_depth (10,20,30). 8. Вывести лучшие параметры. 9. Сравнить важность признаков с логистической регрессией. 10. Сохранить лучшую модель и вывести top-5 самых важных признаков.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).

3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём суть метода случайного леса (Random Forest)? Какие два источника случайности используются?
2. Что такое бутстреп (bootstrap) в контексте случайного леса? Зачем он нужен?
3. Что такое OOB (Out-of-Bag) ошибка? Как она вычисляется и для чего используется?
4. Чем случайный лес отличается от одиночного решающего дерева по склонности к переобучению?
5. Как параметр `n_estimators` влияет на качество и время обучения?
6. Что такое `max_features` и как его выбор влияет на разнообразие деревьев?
7. Как в случайном лесе вычисляется важность признаков? Чем она отличается от важности в одном дереве?
8. В чём разница между `RandomForestClassifier` и `ExtraTreesClassifier`?
9. Как случайный лес справляется с большим количеством признаков (проклятие размерности)?
10. Можно ли использовать случайный лес для регрессии? Если да, то как меняется предсказание?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Сравнение с одиночным деревом. Чёткое сравнение метрик, демонстрация преимущества ансамбля.	10

Правильность использования OOB-оценки. Корректное включение oob_score, интерпретация.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация важности признаков. Анализ важности, связь с предметной областью.	10
Настройка гиперпараметров (GridSearchCV/RandomizedSearchCV). Корректный подбор, обоснование выбора.	10

Лабораторная работа № 14

Градиентный бустинг: XGBoost

Цель: Применить XGBoost для классификации и регрессии.

Теоретическое обоснование

Принцип градиентного бустинга. Деревья строятся последовательно, каждое следующее исправляет ошибки предыдущих. На каждом шаге добавляется дерево, аппроксимирующее отрицательный градиент функции потерь по предсказаниям.

XGBoost — оптимизированная реализация. Особенности:

- Использует вторые производные (метод Ньютона) для ускорения.
- Регуляризация (L1, L2) в функции потерь деревьев.
- Обработка пропусков (по умолчанию направляет объект в сторону, дающую лучший выигрыш).
- Встроенная кросс-валидация и ранняя остановка.
- Эффективная работа с разреженными данными.

Основные гиперпараметры:

- learning_rate (eta) — шаг обновления. Маленький шаг требует больше деревьев, но снижает переобучение.
- n_estimators — число деревьев. Вместе с early_stopping_rounds подбирается автоматически.
- max_depth — глубина деревьев (обычно 3–8).
- subsample — доля объектов для обучения каждого дерева.
- colsample_bytree — доля признаков для каждого дерева.
- reg_alpha (L1) и reg_lambda (L2) — регуляризация коэффициентов.

Ранняя остановка (early stopping). Обучение прекращается, если качество на валидационной выборке не улучшается в течение early_stopping_rounds итераций. Требует передачи eval_set.

Важность признаков может быть оценена разными способами:

- 'weight' — сколько раз признак использовался для разбиения.
- 'gain' — суммарное улучшение accuracy (или другой метрики) от использования признака.
- 'cover' — количество объектов, прошедших через разбиения с этим признаком.

Сравнение со случайным лесом: XGBoost обычно даёт более высокую точность, но требует более тщательной настройки и дольше обучается.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Установить XGBoost (pip install xgboost). Загрузить датасет iris из sklearn. 2. Разделить данные на train/test (70/30) со стратификацией. 3. Обучить XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42). Вычислить accuracy. 4. Вывести важность признаков (feature_importances_) с помощью plot_importance. 5. Использовать GridSearchCV для подбора max_depth (3,5,7) и learning_rate (0.01, 0.1, 0.3). 6. Вывести лучшие параметры и accuracy. 7. Добавить параметр early_stopping_rounds=10 с валидационной выборкой (eval_set). 8. Построить график логарифмической функции потерь (log loss) по эпохам. 9. Сравнить accuracy с RandomForestClassifier(n_estimators=100). 10. Сохранить лучшую модель в файл xgboost_iris.json.
2	1. Загрузить датасет wine из sklearn. Разделить на train/test (75/25). 2. Обучить XGBClassifier(n_estimators=200, learning_rate=0.05, random_state=42). 3. Вычислить F1-macro и confusion matrix. 4. Использовать XGBClassifier(eval_metric='mlogloss') и вывести логи потерь. 5. Подобрать n_estimators с помощью ранней остановки (early_stopping_rounds=20). 6. Визуализировать деревья XGBoost (первое дерево) через plot_tree. 7. Исследовать влияние subsample (0.6, 0.8, 1.0) на accuracy. 8. Сравнить время обучения XGBoost и Random Forest. 9. Вывести важность признаков (тип 'gain'). 10. Сохранить модель в универсальном формате (.ubj).
3	1. Загрузить датасет titanic из Seaborn, провести предобработку. 2. Разделить на train/test (80/20) со стратификацией.

	3. Обучить XGBClassifier(n_estimators=100, max_depth=4, random_state=42). 4. Вычислить accuracy, precision, recall, F1 для класса «выжил». 5. Построить ROC-кривую и вычислить AUC-ROC. 6. Использовать XGBClassifier(scale_pos_weight=...) для балансировки классов. 7. Применить GridSearchCV для подбора learning_rate (0.01,0.05,0.1) и max_depth (3,5,7). 8. Обучить лучшую модель с early_stopping_rounds=15. 9. Сравнить AUC-ROC с логистической регрессией. 10. Сохранить важность признаков в DataFrame.
4	1. Загрузить датасет breast_cancer из sklearn. Разделить на train/test (70/30). 2. Обучить XGBClassifier(n_estimators=150, learning_rate=0.1). 3. Вычислить accuracy, precision, recall, F1. 4. Построить confusion matrix. 5. Использовать cross_val_score с 5 фолдами для оценки стабильности. 6. Подобрать gamma (0, 0.1, 0.5, 1) и min_child_weight (1,3,5) с помощью GridSearchCV. 7. Обучить лучшую модель, сравнить с базовой. 8. Построить график важности признаков (тип 'weight' и 'cover'). 9. Сохранить модель в joblib. 10. Сделать вывод о влиянии gamma на регуляризацию.
5	1. Загрузить датасет california_housing (регрессия) из sklearn. 2. Разделить на train/test (80/20). 3. Обучить XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42). Вычислить RMSE, R^2. 4. Построить график фактических vs предсказанных значений. 5. Использовать early_stopping_rounds=20 с eval_set. 6. Подобрать max_depth (3,5,7,9) и learning_rate (0.01,0.05,0.1) через RandomizedSearchCV. 7. Вывести лучшие параметры и RMSE. 8. Сравнить с RandomForestRegressor(n_estimators=100). 9. Визуализировать важность признаков (вклад каждого признака в снижение ошибки). 10. Сохранить модель.
6	1. Загрузить датасет diabetes (классификация) из sklearn. 2. Разделить на train/test (75/25). 3. Обучить XGBClassifier(n_estimators=100, learning_rate=0.01). 4. Вычислить accuracy и ROC-AUC.

	5. Использовать XGBClassifier(eval_metric='auc') с ранней остановкой. 6. Подобрать colsample_bytree (0.5,0.7,0.9) и colsample_bylevel (0.5,0.7,0.9). 7. Построить график зависимости AUC от n_estimators (с ранней остановкой). 8. Сравнить с LGBMClassifier (LightGBM). 9. Вывести таблицу метрик сравнения. 10. Сохранить лучшую модель.
7	1. Сгенерировать синтетические данные с дисбалансом классов (make_classification, weights=[0.9,0.1]). 2. Разделить на train/test (80/20). 3. Обучить XGBClassifier(scale_pos_weight=9) и без него. Сравнить recall миноритарного класса. 4. Использовать XGBClassifier(eval_metric='logloss') с ранней остановкой. 5. Построить Precision-Recall кривую. 6. Подобрать max_delta_step (0,1,5) для стабилизации. 7. Обучить XGBClassifier(objective='binary:logistic') с eval_metric='aucpr'. 8. Сравнить AUC-PR с базовой моделью. 9. Визуализировать важность признаков. 10. Сделать вывод о работе XGBoost с дисбалансом.
8	1. Загрузить датасет mnist_784 (только 2000 образцов). Разделить на train/test (70/30). 2. Обучить XGBClassifier(n_estimators=50, max_depth=6, learning_rate=0.1). Замерить время. 3. Вычислить accuracy. 4. Использовать XGBClassifier(tree_method='hist') для ускорения – сравнить время. 5. Подобрать n_estimators через раннюю остановку (eval_set). 6. Построить матрицу ошибок. 7. Визуализировать первые 5 деревьев (plot_tree). 8. Сравнить с RandomForestClassifier(n_estimators=50). 9. Сохранить модель и загрузить обратно – проверить предсказания. 10. Сделать вывод о масштабируемости XGBoost.
9	1. Загрузить датасет adult из openml (предсказание дохода). Провести предобработку (one-hot). 2. Разделить на train/test (80/20). 3. Обучить XGBClassifier(n_estimators=200, learning_rate=0.05,

	random_state=42). 4. Вычислить accuracy, precision, recall, F1 для класса >50K. 5. Построить ROC-кривую. 6. Использовать XGBClassifier(reg_alpha=0.1, reg_lambda=1) – L1 и L2 регуляризация. 7. Подобрать reg_alpha и reg_lambda с помощью GridSearchCV. 8. Сравнить важность признаков до и после регуляризации. 9. Сохранить лучшую модель. 10. Сделать вывод о роли регуляризации в XGBoost.
10	1. Загрузить датасет airline (задержка рейсов) – малую выборку. 2. Разделить на train/test (75/25). 3. Обучить XGBClassifier(n_estimators=100, learning_rate=0.1). 4. Вычислить accuracy и AUC-ROC. 5. Использовать XGBClassifier(eval_metric='error') – ранняя остановка. 6. Подобрать max_depth и min_child_weight с помощью RandomizedSearchCV. 7. Построить график обучения (log loss на train и validation). 8. Визуализировать важность признаков (тип 'gain'). 9. Сравнить с CatBoostClassifier (если установлен). 10. Сохранить модель и метрики в CSV.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое градиентный бустинг? Чем он отличается от случайного леса?

2. Какие основные гиперпараметры XGBoost отвечают за скорость обучения и количество деревьев?
3. Как работает ранняя остановка (early stopping) в XGBoost? Какие параметры нужно задать?
4. Что такое subsample и colsample_bytree? Как они влияют на регуляризацию?
5. Какие функции потерь (objective) поддерживает XGBoost для классификации и регрессии?
6. Чем отличается важность признаков типа 'weight' от 'gain' в XGBoost?
7. Что такое gamma (min_split_loss) в XGBoost? Как он влияет на построение деревьев?
8. Как XGBoost обрабатывает пропущенные значения?
9. В чём преимущества tree_method='hist' перед стандартным 'exact'?
10. Как сохранить и загрузить модель XGBoost? Какие форматы поддерживаются?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность использования ранней остановки. Корректное задание eval_set, early_stopping_rounds, анализ графика.	10
Настройка гиперпараметров (GridSearchCV / RandomizedSearchCV). Обоснованный выбор, сравнение качества.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация важности признаков. Анализ важности, связь с задачей.	10
Сравнение с другими моделями (Random Forest, логистическая регрессия). Сравнение метрик, времени обучения.	10

Лабораторная работа № 15

Градиентный бустинг: LightGBM и CatBoost

Цель: Ознакомиться с альтернативными реализациями градиентного бустинга.

Теоретическое обоснование

LightGBM (Microsoft). Ключевые отличия от XGBoost:

- **Leaf-wise рост** (в отличие от level-wise). На каждой итерации разбивается лист, дающий наибольшее уменьшение ошибки. Быстрее сходится, но может привести к переобучению на малых данных.
- **GOSS (Gradient-based One-Side Sampling)** — выборка объектов с большим градиентом (важных) и случайная выборка остальных.
- **EFB (Exclusive Feature Bundling)** — связывание разреженных признаков для уменьшения размерности.
- **Поддержка категориальных признаков** через параметр categorical_feature.

Основные гиперпараметры: num_leaves (основной, связан с max_depth = floor(log2(num_leaves))), min_data_in_leaf, feature_fraction, bagging_fraction, boosting _type (gbdt, dart, goss).

CatBoost (Yandex). Специализация на категориальных признаках:

- **Ordered boosting** — для предотвращения целевого перекоса (target leakage) при вычислении целевой статистики категорий.
- **Автоматическое кодирование категорий** с использованием one-hot для низкокардинальных и ordered target statistics для высококардинальных. Не требует предварительного кодирования.
- **Симметричные деревья** (облегчают инференс).
- **Overfitting detector** (аналог early stopping).

Гиперпараметры: iterations (аналог n_estimators), depth (обычно 6–10), learning_rate, l2_leaf_reg, border_count (дискретизация признаков).

Сравнение трёх реализаций:

- XGBoost — классика, надёжен, много настроек.
- LightGBM — самый быстрый, хорош для больших данных.
- CatBoost — лучший для данных с категориальными признаками, часто даёт

хорошие результаты по умолчанию.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Установить LightGBM (pip install lightgbm). Загрузить датасет iris. Разделить на train/test (70/30). 2. Обучить LGBMClassifier(n_estimators=100, random_state=42). Вычислить accuracy. 3. Использовать параметр categorical_feature для обработки категориальных признаков (если есть). 4. Применить early_stopping_rounds=10 с eval_set. Построить график log loss. 5. Подобрать num_leaves (5,10,20,31) и learning_rate (0.01,0.05,0.1) через GridSearchCV. 6. Обучить CatBoostClassifier на том же датасете (без предварительного кодирования категориальных). 7. Сравнить accuracy LightGBM и CatBoost. 8. Визуализировать важность признаков для обеих моделей. 9. Сравнить время обучения LightGBM и CatBoost. 10. Сохранить лучшую модель.
2	1. Загрузить датасет titanic, провести предобработку (заполнение пропусков). 2. Разделить на train/test (80/20). 3. Обучить LGBMClassifier с параметрами num_leaves=31, learning_rate=0.05. 4. Вычислить accuracy, precision, recall, F1 для класса «выжил». 5. Построить ROC-кривую, вычислить AUC-ROC. 6. Обучить CatBoostClassifier (без one-hot кодирования, передав список категориальных признаков). 7. Сравнить AUC-ROC с LightGBM. 8. Использовать cat_features в CatBoost для указания категориальных столбцов. 9. Построить матрицу ошибок для лучшей модели. 10. Сохранить обе модели и сравнить размеры файлов.
3	1. Загрузить датасет adult из openml. Провести предобработку (замена '?' на NaN). 2. Разделить на train/test (75/25) со стратификацией.

	3. Обучить LGBMClassifier(n_estimators=200, random_state=42). 4. Использовать feature_fraction=0.8 и bagging_fraction=0.8 для регуляризации. 5. Применить раннюю остановку (early_stopping_rounds=20). 6. Обучить CatBoostClassifier с параметрами iterations=200, depth=6. 7. Вывести важность признаков для CatBoost (метод get_feature_importance()). 8. Сравнить F1-score для класса >50K. 9. Построить график сравнения метрик. 10. Сделать вывод о том, какой алгоритм лучше справился с дисбалансом.
4	1. Загрузить датасет california_housing (регрессия). Разделить на train/test (80/20). 2. Обучить LGBMRegressor(n_estimators=100, random_state=42). Вычислить RMSE, R^2. 3. Обучить CatBoostRegressor(iterations=100, learning_rate=0.05). 4. Сравнить RMSE и R^2. 5. Построить график предсказанных vs истинных значений для обеих моделей. 6. Использовать GridSearchCV для LightGBM (num_leaves, learning_rate, min_child_samples). 7. Использовать GridSearchCV для CatBoost (depth, learning_rate, l2_leaf_reg). 8. Вывести лучшие параметры и итоговые метрики. 9. Визуализировать важность признаков. 10. Сохранить лучшую модель в joblib.
5	1. Загрузить датасет wine. Разделить на train/test (75/25). 2. Обучить LGBMClassifier с настройками по умолчанию. 3. Обучить CatBoostClassifier с настройками по умолчанию. 4. Сравнить accuracy и F1-macro. 5. Исследовать влияние num_leaves (5,15,31,50) на accuracy LightGBM (построить график). 6. Исследовать влияние depth (3,6,9,12) на accuracy CatBoost (построить график). 7. Использовать RandomizedSearchCV для подбора гиперпараметров обеих моделей. 8. Сравнить лучшее качество и время поиска. 9. Построить confusion matrix для лучшей модели. 10. Сделать вывод о сложности настройки каждого алгоритма.
6	1. Загрузить датасет penguins из Seaborn, удалить строки с

	пропусками. 2. Категориальные признаки: species, island, sex. Не кодировать их перед обучением. 3. Разделить на train/test (80/20). 4. Обучить LGBMClassifier, передав список категориальных признаков в categorical_feature. 5. Обучить CatBoostClassifier, передав список категориальных признаков в cat_features. 6. Сравнить accuracy и F1-macro. 7. Вывести важность признаков для обеих моделей. 8. Использовать cross_val_score с 5 фолдами для оценки стабильности. 9. Построить матрицу ошибок. 10. Сделать вывод о том, как каждая модель обрабатывает категориальные признаки.
7	1. Сгенерировать синтетические данные с помощью make_classification (1000 образцов, 20 признаков). 2. Разделить на train/test (80/20). 3. Обучить LGBMClassifier с num_leaves=31. 4. Обучить CatBoostClassifier с depth=6. 5. Обучить XGBClassifier с настройками по умолчанию. 6. Сравнить accuracy и F1 всех трёх моделей. 7. Сравнить время обучения. 8. Построить столбчатую диаграмму сравнения метрик. 9. Использовать early_stopping_rounds=10 для всех трёх моделей. 10. Сохранить таблицу сравнения в CSV.
8	1. Загрузить датасет breast_cancer. Разделить на train/test (70/30). 2. Обучить LGBMClassifier с num_leaves=15, learning_rate=0.05. 3. Использовать lgb.cv для кросс-валидации с ранней остановкой. 4. Обучить CatBoostClassifier с od_type='Iter', od_wait=20 (overfitting detector). 5. Построить график логарифмической функции потерь для CatBoost (метод evals_result_). 6. Сравнить AUC-ROC. 7. Использовать GridSearchCV для CatBoost (l2_leaf_reg от 1 до 10, border_count от 128 до 255). 8. Вывести лучшие параметры. 9. Построить ROC-кривые на одном графике. 10. Сделать вывод о влиянии регуляризации на качество.
9	1. Загрузить датасет digits из sklearn. Разделить на train/test (80/20). 2. Обучить LGBMClassifier с num_leaves=50, min_child_samples=2

	0. 3. Обучить CatBoostClassifier с depth=8, iterations=300. 4. Сравнить accuracy и macro F1. 5. Построить матрицу ошибок (10×10). 6. Использовать RandomizedSearchCV для LightGBM (num_leaves, learning_rate, feature_fraction). 7. Использовать RandomizedSearchCV для CatBoost (depth, learning_rate, l2_leaf_reg). 8. Сравнить лучшее качество и время обучения. 9. Визуализировать важность признаков (изображения 8×8). 10. Сохранить лучшую модель.
10	1. Загрузить датасет flight_delays (малая выборка). 2. Разделить на train/test (80/20). 3. Обучить LGBMClassifier с boosting_type='gbdt'. 4. Обучить LGBMClassifier с boosting_type='dart' (Dropouts meet Multiple Additive Regression Trees). 5. Сравнить accuracy и время обучения. 6. Обучить CatBoostClassifier с boosting_type='Ordered'. 7. Сравнить все три конфигурации. 8. Построить график сравнения метрик. 9. Использовать early_stopping_rounds=15 для всех. 10. Сделать вывод о применимости разных boosting_type.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основное отличие LightGBM от XGBoost? Что означает leaf-wise рост деревьев?
2. Какие гиперпараметры LightGBM отвечают за регуляризацию и предотвращение переобучения?
3. Как в LightGBM обрабатываются категориальные признаки? Какой параметр за это отвечает?
4. Что такое Ordered Boosting в CatBoost и какую проблему он решает?
5. Как CatBoost автоматически обрабатывает категориальные признаки без предварительного кодирования?
6. В чём разница между `num_leaves` в LightGBM и `max_depth` в классических деревьях? Как они связаны?
7. Как в CatBoost используется early stopping для предотвращения переобучения?
8. Какие преимущества LightGBM даёт при работе с большими объёмами данных?
9. В каких случаях CatBoost может быть предпочтительнее LightGBM?
10. Сравните скорость обучения и качество XGBoost, LightGBM и CatBoost.

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность работы с категориальными признаками. Использование categorical_feature / cat_features без ручного кодирования.	10
Использование ранней остановки. Корректное применение early_stopping_rounds, eval_set, overfitting detector.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ различий между LightGBM и	10

CatBoost, выбор оптимального алгоритма.	
Сравнение с другими моделями (XGBoost / Random Forest). Сравнение метрик, времени обучения, удобства настройки.	10

Лабораторная работа № 16

Метод опорных векторов (SVM)

Цель: Применить SVM для классификации и настроить гиперпараметры.

Теоретическое обоснование

Линейный SVM. Ищет гиперплоскость (разделяющую прямую в 2D), которая максимизирует отступ (margin) — расстояние до ближайших точек (опорных векторов). Плоскость определяется как $w \cdot x + b = 0$, отступ = $2/\|w\|$. Задача оптимизации: min $\|w\|^2$ при условии, что все точки правильно классифицированы и находятся за отступом.

Мягкий отступ (soft margin). Добавляется параметр C, который контролирует штраф за ошибки классификации. Малое C → широкий отступ, допускает больше ошибок (меньше переобучения). Большое C → узкий отступ, стремится правильно классифицировать все точки (риск переобучения).

Ядерный трюк (kernel trick). Для нелинейных данных SVM проецирует данные в пространство более высокой размерности (или бесконечной) с помощью ядерной функции, где они становятся линейно разделимыми. Популярные ядра:

- **Линейное:** $K(x, y) = x \cdot y$.
- **Полиномиальное:** $K(x, y) = (\gamma \cdot x \cdot y + r)^d$.
- **RBF (Radial Basis Function):** $K(x, y) = \exp(-\gamma \cdot \|x-y\|^2)$. Наиболее популярно, моделирует сложные границы.

Гиперпараметры для RBF:

- gamma (γ) — радиус влияния одного обучающего образца. Большой γ → очень извилистые границы (переобучение), малый γ → гладкие (недообучение).
- C — штраф за ошибки.

SVM для регрессии (SVR). Вместо отступа используется ε-чувствительная функция потерь: ошибки меньше ε игнорируются, ошибки больше ε штрафуются

линейно.

Масштабирование признаков критически важно для SVM, так как метрики расстояний чувствительны к масштабу.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris из sklearn. Оставить два класса (например, setosa и versicolor) для бинарной классификации. 2. Разделить на train/test (70/30) со стратификацией. 3. Обучить SVC(kernel='linear', C=1.0, random_state=42). Вычислить accuracy. 4. Визуализировать разделяющую гиперплоскость и опорные векторы (используя первые два признака). 5. Заменить ядро на kernel='rbf' и сравнить accuracy. 6. Использовать GridSearchCV для подбора C (0.1,1,10,100) и gamma (0.01,0.1,1,10) для RBF. 7. Вывести лучшие параметры и accuracy. 8. Построить матрицу ошибок для лучшей модели. 9. Сравнить время обучения линейного и RBF ядра. 10. Сохранить лучшую модель в joblib.
2	1. Загрузить датасет wine (классификация на 3 класса). Разделить на train/test (75/25). 2. Обучить SVC(kernel='linear', decision_function_shape='ovr'). Вычислить accuracy. 3. Обучить SVC(kernel='rbf', decision_function_shape='ovo'). Сравнить accuracy. 4. Использовать StandardScaler перед обучением – сравнить качество. 5. Подобрать C и gamma для RBF с помощью RandomizedSearchCV. 6. Построить confusion matrix в виде тепловой карты. 7. Визуализировать важность признаков (используя абсолютные значения коэффициентов для линейного SVM). 8. Сравнить SVM с логистической регрессией. 9. Сохранить лучшую модель. 10. Сделать вывод о влиянии масштабирования на SVM.
3	1. Загрузить датасет breast_cancer. Разделить на train/test (80/20) со стратификацией.

	2. Масштабировать данные StandardScaler. 3. Обучить SVC(kernel='rbf', C=1, gamma='scale'). Вычислить accuracy, precision, recall, F1. 4. Построить ROC-кривую и вычислить AUC-ROC. 5. Использовать GridSearchCV для подбора C (0.01,0.1,1,10,100) и gamma (0.001,0.01,0.1,1). 6. Вывести лучшие параметры и метрики. 7. Построить матрицу ошибок. 8. Сравнить с SVC(kernel='linear'). 9. Визуализировать опорные векторы (для первых двух признаков после PCA). 10. Сохранить лучшую модель.
4	1. Сгенерировать синтетические данные с помощью make_circles(n_samples=300, noise=0.1, factor=0.5). 2. Визуализировать данные. 3. Разделить на train/test (80/20). 4. Обучить линейный SVM – качество будет низким. 5. Обучить SVC(kernel='rbf', C=1, gamma=1) – визуализировать границы решений. 6. Подобрать gamma (0.1,1,10,100) и C (0.1,1,10) с помощью кросс-валидации. 7. Визуализировать границы для лучшей модели. 8. Сравнить с SVC(kernel='poly', degree=3). 9. Построить график зависимости accuracy от gamma. 10. Сделать вывод о влиянии gamma на форму границы.
5	1. Загрузить датасет digits (8×8) из sklearn. Разделить на train/test (80/20). 2. Масштабировать пиксели (разделить на 16). 3. Обучить SVC(kernel='rbf', C=1, gamma=0.01). Вычислить accuracy. 4. Использовать GridSearchCV для подбора C и gamma (с небольшим числом вариантов). 5. Вывести лучшие параметры и accuracy. 6. Построить confusion matrix. 7. Применить PCA для снижения размерности до 20 компонент, обучить SVM на PCA-данных. 8. Сравнить accuracy и время обучения. 9. Визуализировать веса линейного SVM (если использовать linear) в виде изображений. 10. Сохранить лучшую модель.
6	1. Загрузить датасет california_housing (регрессия). Разделить на

	train/test (80/20). 2. Масштабировать признаки. 3. Обучить SVR(kernel='rbf', C=1, gamma='scale'). Вычислить RMSE и R^2. 4. Подобрать C (0.1,1,10,100) и gamma (0.01,0.1,1) с помощью GridSearchCV. 5. Вывести лучшие параметры и метрики. 6. Построить график предсказанных vs истинных значений. 7. Сравнить с SVR(kernel='linear'). 8. Сравнить с линейной регрессией (по R^2). 9. Визуализировать важность признаков (коэффициенты линейного SVR). 10. Сохранить лучшую модель.
7	1. Загрузить датасет titanic после предобработки. Разделить на train/test (80/20). 2. Масштабировать числовые признаки. 3. Обучить SVC(kernel='rbf', class_weight='balanced') для учёта дисбаланса. 4. Вычислить accuracy, precision, recall, F1. 5. Построить ROC-кривую и вычислить AUC-ROC. 6. Сравнить с SVC(kernel='linear', class_weight='balanced'). 7. Использовать GridSearchCV для подбора C и gamma. 8. Вывести лучшие параметры. 9. Построить confusion matrix. 10. Сравнить с логистической регрессией.
8	1. Сгенерировать данные с помощью make_classification(n_features=2, n_redundant=0, n_clusters_per_class=1). 2. Разделить на train/test (80/20). 3. Обучить SVC(kernel='linear', C=0.01), C=1, C=100. Визуализировать границы и опорные векторы. 4. Показать, как меняется ширина отступа (margin) при изменении C. 5. Обучить SVC(kernel='rbf', gamma=0.1) и gamma=10. Визуализировать границы. 6. Сравнить влияние gamma на переобучение. 7. Использовать GridSearchCV для нахождения оптимальных параметров. 8. Визуализировать границы лучшей модели. 9. Сохранить модель. 10. Сделать вывод о роли C и gamma.

9	1. Загрузить датасет mnist_784 (2000 образцов). Разделить на train/test (70/30). 2. Нормализовать пиксели. 3. Обучить SVC(kernel='rbf', C=1, gamma=0.001). Замерить время обучения и предсказания. 4. Использовать SVC(kernel='linear') – сравнить время и accuracy. 5. Применить RandomizedSearchCV для подбора C и gamma (ограниченное число итераций). 6. Построить confusion matrix. 7. Сравнить с KNeighborsClassifier(n_neighbors=5). 8. Сократить обучающую выборку до 500 образцов – сравнить качество. 9. Сохранить лучшую модель. 10. Сделать вывод о масштабируемости SVM на большие данные.
10	1. Загрузить датасет adult (предобработанный, с one-hot кодированием). Разделить на train/test (80/20). 2. Масштабировать признаки. 3. Обучить SVC(kernel='linear') – вычислить accuracy, precision, recall для класса >50K. 4. Обучить SVC(kernel='rbf') – сравнить метрики. 5. Использовать GridSearchCV для подбора C (0.01,0.1,1,10) для линейного ядра. 6. Построить ROC-кривую для лучшей линейной модели. 7. Вывести коэффициенты линейного SVM (важность признаков). 8. Сравнить с логистической регрессией. 9. Сохранить лучшую модель. 10. Сделать вывод о том, какое ядро лучше для этого датасета.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).

5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основная идея метода опорных векторов (SVM) для классификации? Что такое «отступ» (margin)?
2. Что такое опорные векторы (support vectors)? Как они влияют на построение гиперплоскости?
3. В чём разница между линейным и нелинейным ядрами (например, RBF)? Когда следует использовать RBF?
4. Что делает параметр C в SVM? Как он влияет на компромисс между шириной отступа и ошибками классификации?
5. Что такое gamma в ядре RBF? Как её изменение влияет на форму разделяющей границы?
6. Как SVM работает в задачах многоклассовой классификации (стратегии OvR, OvO)?
7. Почему масштабирование признаков критически важно для SVM?
8. Как SVM можно применить для регрессии (SVR)? В чём отличие от классификации?
9. Каковы основные преимущества и недостатки SVM по сравнению с логистической регрессией?
10. Что такое «ядровой трюк» (kernel trick) и как он позволяет SVM работать с нелинейными данными?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность масштабирования данных. Обязательное масштабирование перед обучением SVM, объяснение важности.	10
Настройка гиперпараметров (C, gamma). Использование GridSearchCV/RandomizedSearchCV, обоснование выбора.	10

Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ влияния C и gamma, сравнение ядер, визуализация границ.	10
Визуализации (границы решений, опорные векторы). Качество, читаемость, подписи.	10

Лабораторная работа № 17

Кластеризация методом k-средних

Цель: Реализовать кластеризацию данных без учителя.

Теоретическое обоснование

Алгоритм:

1. Инициализировать K центроидов (обычно случайно из данных, или k-means++).
2. Повторять до сходимости:
   - Приписать каждый объект к ближайшему центроиду (по евклидову расстоянию).
   - Пересчитать центроиды как средние координат объектов в каждом кластере.
3. Алгоритм сходится к локальному минимуму суммы квадратов расстояний (inertia).

Инерция (inertia) = $\Sigma \Sigma ||x - \mu_c||^2$ — сумма квадратов расстояний от объектов до своего центроида. Уменьшается с ростом K, поэтому не может служить критерием выбора K.

Выбор числа кластеров K:

- **Метод локтя (elbow method):** строится график inertia(K). Точка излома (локоть) — оптимальное K.
- **Силуэтный коэффициент (silhouette score):** для каждого объекта a: s = (b-a)/max(a,b), где a — среднее расстояние до объектов своего кластера, b —

среднее расстояние до объектов соседнего кластера. Средний s по всем объектам от -1 до 1. Чем ближе к 1, тем лучше.

- **Индекс Калински-Харабаса** и **индекс Дэвиса-Болдина** — другие метрики.

Инициализация k-means++: выбирает первые центроиды с вероятностью, пропорциональной квадрату расстояния до ближайшего уже выбранного центроида. Улучшает сходимость и снижает зависимость от начальных условий.

Недостатки k-means: предполагает сферические кластеры одинакового размера и плотности; чувствителен к выбросам и масштабу признаков; не работает с несферическими кластерами (луна, круги).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Сгенерировать синтетические данные с помощью make_blobs(n_samples=300, centers=4, cluster_std=0.8, random_state=42). 2. Визуализировать данные (scatter plot). 3. Применить KMeans(n_clusters=4, random_state=42) и визуализировать результат (точки + центроиды). 4. Вычислить инерцию (inertia) для полученной кластеризации. 5. Использовать метод локтя: построить график зависимости инерции от числа кластеров (1–10). 6. Выбрать оптимальное число кластеров по локтю. 7. Вычислить силуэтный коэффициент (silhouette score) для оптимального числа кластеров. 8. Визуализировать силуэтные оценки для каждого объекта (silhouette plot). 9. Сравнить качество кластеризации при k=3, k=4, k=5. 10. Сохранить метки кластеров и координаты центроидов в CSV.
2	1. Загрузить датасет iris из sklearn. Удалить целевую переменную (оставить только признаки). 2. Масштабировать признаки с помощью StandardScaler. 3. Применить KMeans(n_clusters=3, random_state=42). 4. Визуализировать результат кластеризации (используя первые два главных компонента PCA). 5. Сравнить полученные кластеры с истинными метками (визуально и через adjusted Rand index).

	6. Вычислить силуэтный коэффициент. 7. Построить график метода локтя (инерция vs k) для k от 2 до 10. 8. Определить оптимальное k по силуэтному коэффициенту. 9. Сравнить кластеризацию k-means с иерархической (AgglomerativeClustering). 10. Сохранить модель KMeans в файл.
3	1. Загрузить датасет wine из sklearn. Признаки – химический состав вин. 2. Масштабировать признаки. 3. Применить KMeans(n_clusters=3, random_state=42). 4. Вычислить силуэтный коэффициент. 5. Построить график метода локтя. 6. Сравнить кластеры с истинными сортами вин (используя cross_tab и adjusted Rand index). 7. Проанализировать центроиды кластеров – какие признаки различаются сильнее всего. 8. Визуализировать кластеры в пространстве первых двух главных компонент. 9. Выполнить кластеризацию с k=4 и k=5 – изменился ли силуэтный коэффициент? 10. Сделать вывод о соответствии кластеров реальным сортам.
4	1. Сгенерировать синтетические данные с помощью make_blobs(n_samples=500, centers=5, cluster_std=1.0, random_state=42). 2. Применить KMeans с k=3,5,7. Для каждого вычислить инерцию и силуэт. 3. Визуализировать кластеризацию для k=5 (точки и центроиды). 4. Использовать KMeans с разными инициализациями (init='random' и init='k-means++'), сравнить инерцию. 5. Задать n_init=1 и n_init=10, сравнить инерцию и время выполнения. 6. Построить график зависимости инерции от n_init. 7. Использовать KMeans с max_iter=10 и max_iter=300 – сравнить результат. 8. Выполнить кластеризацию на немасштабированных данных и сравнить силуэт с масштабированными. 9. Сохранить метки кластеров в CSV. 10. Сделать вывод о влиянии параметров инициализации.
5	1. Загрузить датасет digits (только признаки, 64 измерения). 2. Применить PCA для снижения размерности до 2 компонент.

	3. Выполнить KMeans(n_clusters=10, random_state=42) на исходных (64D) данных. 4. Визуализировать кластеры в пространстве двух первых главных компонент. 5. Сравнить с истинными цифрами (adjusted Rand index, mutual information). 6. Построить матрицу соответствия кластеров и истинных меток (heatmap). 7. Для каждой цифры определить доминирующий кластер. 8. Выполнить метод локтя для k от 2 до 15 (на исходных данных). 9. Подобрать оптимальное k по силуэтному коэффициенту. 10. Сделать вывод о том, насколько хорошо k-means выделяет цифры.
6	1. Загрузить датасет customer_segmentation (например, Mall Customers из Kaggle). 2. Признаки: Annual Income, Spending Score. Масштабировать. 3. Визуализировать данные (scatter plot). 4. Применить метод локтя для k от 2 до 10. 5. Выбрать оптимальное k и выполнить KMeans. 6. Визуализировать кластеры и центроиды. 7. Проинтерпретировать каждый кластер (например, «богатые, но мало тратят»). 8. Вычислить средние значения признаков по кластерам. 9. Добавить новый клиент с координатами (60, 50) – определить его кластер. 10. Сохранить модель и центроиды в файл.
7	1. Загрузить датасет olivetti_faces (только 100 изображений для простоты) или fetch_olivetti_faces. 2. Применить KMeans(n_clusters=10, random_state=42) к изображениям (каждое изображение – вектор). 3. Визуализировать центроиды кластеров (как изображения). 4. Вычислить силуэтный коэффициент. 5. Построить график метода локтя. 6. Выполнить PCA до 50 компонент, затем кластеризацию – сравнить силуэт. 7. Для одного из кластеров показать несколько исходных изображений, попавших в него. 8. Проверить, соответствуют ли кластеры разным людям (используя истинные метки, если есть). 9. Подобрать оптимальное k с помощью силуэтного анализа.

	10. Сохранить центроиды кластеров в виде изображений.
8	1. Сгенерировать данные с помощью make_moons(n_samples=300, noise=0.05). 2. Визуализировать данные. 3. Применить KMeans(n_clusters=2) – показать, что k-means плохо работает с несферическими кластерами. 4. Визуализировать границы кластеров k-means. 5. Сравнить с результатом DBSCAN(eps=0.2, min_samples=5). 6. Для k-means вычислить силуэт, для DBSCAN – тоже. 7. Попробовать KMeans с n_clusters=5 – что получится? 8. Применить SpectralClustering и сравнить. 9. Сделать вывод о применимости k-means к данным сложной формы. 10. Сохранить результаты визуализации.
9	1. Загрузить датасет seeds (пшеница). Масштабировать признаки. 2. Выполнить KMeans(n_clusters=3, random_state=42). 3. Вычислить силуэтный коэффициент и инерцию. 4. Построить силуэтный график (silhouette plot). 5. Использовать метод локтя и метод силуэта для выбора k (2–8). 6. Выбрать оптимальное k. 7. Сравнить кластеры с истинными сортами пшеницы (adjusted Rand index). 8. Визуализировать кластеры в пространстве двух главных компонент. 9. Определить, какие признаки наиболее важны для разделения кластеров (по центроидам). 10. Сохранить таблицу средних признаков по кластерам.
10	1. Загрузить датасет wholesale_customers (оптовые клиенты). 2. Признаки: годовые расходы по категориям. Масштабировать. 3. Выполнить KMeans с k=4. 4. Построить график метода локтя. 5. Визуализировать кластеры с помощью PCA (первые 2 компоненты). 6. Проинтерпретировать кластеры (например, «клиенты, закупающие много молочных продуктов»). 7. Вычислить средние значения по кластерам, построить тепловую карту. 8. Определить размер каждого кластера. 9. Использовать MiniBatchKMeans для ускорения, сравнить центроиды с обычным KMeans. 10. Сохранить модель и предсказания для новых данных.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Опишите алгоритм k-means кластеризации. Какие шаги выполняются на каждой итерации?
2. Что такое инерция (inertia) в k-means? Как она вычисляется и как интерпретируется?
3. Как выбрать оптимальное число кластеров в k-means? Опишите метод локтя и силуэтный анализ.
4. Что такое силуэтный коэффициент (silhouette score)? В каких пределах он изменяется и что означает близость к 1?
5. Почему k-means чувствителен к масштабированию признаков? Приведите пример.
6. Что такое проблема «проклятия инициализации» в k-means? Как k-means++ помогает её решить?
7. Каковы основные недостатки k-means? Когда он неэффективен (форма кластеров, выбросы)?
8. Чем отличается KMeans от MiniBatchKMeans? Когда стоит использовать второй?
9. Как оценить качество кластеризации, если известны истинные метки? Назовите 2–3 метрики.

10. Как в k-means обрабатываются категориальные признаки? Нужно ли их кодировать?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора числа кластеров. Использование метода локтя и/или силуэтного анализа, обоснование выбора.	10
Интерпретация кластеров. Анализ центроидов, средних значений, описание кластеров.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и сравнение. Сравнение с истинными метками (если есть), анализ силы и слабости k-means.	10
Визуализации (scatter plot кластеров, метод локтя, силуэтный график). Качество, читаемость, подписи.	10

Лабораторная работа № 18

Иерархическая кластеризация

Цель: Применить иерархическую кластеризацию и построить дендрограмму.

Теоретическое обоснование

Агломеративная (восходящая) кластеризация. Начинаем с каждого объекта как отдельного кластера. Затем на каждом шаге объединяем два ближайших кластера. Повторяем, пока не останется один кластер. Результат — дендрограмма (дерево).

Методы связи (linkage criteria) — способ измерения расстояния между двумя кластерами:

- **Single linkage (ближайший сосед):** расстояние между самыми близкими точками кластеров. Приводит к вытянутым кластерам, чувствителен к шуму (эффект цепочки).
- **Complete linkage (дальний сосед):** расстояние между самыми удалёнными точками. Стремится к компактным кластерам.
- **Average linkage:** среднее расстояние между всеми парами точек.
- **Ward linkage:** минимизирует увеличение внутрикластерной дисперсии при объединении. Один из лучших для сферических кластеров.

Дендрограмма. Вертикальная ось — расстояние объединения. Горизонтальный разрез на определённой высоте даёт набор кластеров. Высота разреза выбирается либо визуально, либо по максимальному разрыву расстояний (метод «колена»).

Преимущества иерархической кластеризации: не требует заранее задавать K, даёт полную иерархию, визуализируема. **Недостатки:** вычислительная сложность $O(N^3)$ (для наивной реализации), плохо масштабируется на большие данные.

Сравнение с k-means: иерархическая может выделять кластеры произвольной формы, но для больших данных предпочтительнее k-means.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Сгенерировать синтетические данные с помощью make_blobs(n_samples=200, centers=4, cluster_std=0.8, random_state=42). 2. Выполнить агломеративную кластеризацию с AgglomerativeClustering(n_clusters=4, linkage='ward'). 3. Визуализировать результат (scatter plot, цвет по кластерам). 4. Построить дендрограмму с помощью scipy.cluster.hierarchy.dendrogram (предварительно вычислив матрицу связей). 5. По дендрограмме определить оптимальное число кластеров (визуально). 6. Сравнить кластеризацию с linkage='single', 'complete', 'average', 'ward' – визуализировать результаты. 7. Вычислить силуэтный коэффициент для каждого метода связи, выбрать лучший. 8. Сравнить с KMeans(n_clusters=4) (по силуэту и визуально). 9. Построить матрицу соответствия (если известны истинные метки). 10. Сохранить дендрограмму в файл.
2	1. Загрузить датасет iris (только признаки). Масштабировать. 2. Построить дендрограмму (метод Ward). 3. По дендрограмме определить оптимальное число кластеров (выбрать уровень отсечения). 4. Выполнить AgglomerativeClustering(n_clusters=3, linkage='ward'). 5. Сравнить полученные кластеры с истинными видами ирисов (adjusted Rand index). 6. Повторить с linkage='single' – как изменился результат? 7. Визуализировать кластеры в пространстве первых двух главных компонент. 8. Построить тепловую карту расстояний между объектами (pdist + squareform). 9. Вычислить силуэтный коэффициент для разных методов связи. 10. Сделать вывод о том, какой метод связи лучше для iris.
3	1. Загрузить датасет wine из sklearn. Масштабировать признаки. 2. Построить дендрограмму (метод Ward) с отображением цвета ветвей (color_threshold). 3. Выбрать число кластеров = 3 (по количеству сортов). 4. Выполнить AgglomerativeClustering(n_clusters=3, linkage='ward').

	5. Сравнить кластеры с истинными сортами (adjusted Rand index, mutual information). 6. Повторить для linkage='complete' – сравнить качество. 7. Построить график силуэтного коэффициента для числа кластеров от 2 до 10. 8. Определить оптимальное число кластеров по силуэту. 9. Визуализировать иерархическую кластеризацию в виде тепловой карты с переупорядоченными строками (clustermap из seaborn). 10. Сохранить матрицу связей для дальнейшего использования.
4	1. Сгенерировать данные с помощью make_circles(n_samples=300, noise=0.05, factor=0.5). 2. Визуализировать данные. 3. Выполнить агломеративную кластеризацию с linkage='ward' и n_clusters=2. 4. Визуализировать результат – почему Ward не справился? 5. Повторить с linkage='single' – что изменилось? 6. Построить дендрограмму для метода single. 7. Сравнить с DBSCAN(eps=0.15, min_samples=5). 8. Вычислить силуэтный коэффициент для single linkage и DBSCAN. 9. Сделать вывод о применимости иерархической кластеризации к несферическим кластерам. 10. Сохранить дендрограмму в PNG.
5	1. Загрузить датасет digits (только 500 образцов для скорости). 2. Выполнить PCA для снижения размерности до 20 компонент. 3. Построить дендрограмму на PCA-данных (метод Ward). 4. Выбрать число кластеров = 10 (по количеству цифр). 5. Выполнить AgglomerativeClustering(n_clusters=10, linkage='ward'). 6. Сравнить кластеры с истинными цифрами (confusion matrix, adjusted Rand index). 7. Визуализировать центроиды кластеров (восстановив из PCA). 8. Попробовать linkage='average' – сравнить качество. 9. Построить график зависимости adjusted Rand index от метода связи. 10. Сохранить предсказанные метки кластеров.
6	1. Загрузить датасет olivetti_faces (50 человек, 10 изображений на человека). 2. Выполнить агломеративную кластеризацию с n_clusters=50 (надеясь выделить каждого человека). 3. Оценить качество: для каждого кластера определить, сколько

	различных людей в него попало. 4. Построить дендрограмму для первых 50 изображений (ограничить). 5. Визуализировать матрицу связей в виде тепловой карты. 6. Использовать linkage='complete' и сравнить с ward. 7. Подобрать число кластеров, максимизирующее adjusted Rand index (если есть метки). 8. Визуализировать несколько примеров из каждого кластера. 9. Сохранить иерархическую кластеризацию в файл (модель нельзя сохранить, но можно сохранить матрицу связей). 10. Сделать вывод о том, насколько хорошо иерархическая кластеризация группирует лица.
7	1. Загрузить датасет countries (характеристики стран). Масштабировать. 2. Выполнить иерархическую кластеризацию с методом Ward. 3. Построить дендрограмму, подписать названия стран (если их не слишком много). 4. Выбрать число кластеров = 4 по визуальной оценке дендрограммы. 5. Проинтерпретировать каждый кластер (например, «развитые страны», «развивающиеся»). 6. Вычислить средние значения признаков по кластерам. 7. Сравнить с KMeans(n_clusters=4). 8. Построить карту мира с раскраской стран по кластерам (если есть координаты). 9. Использовать linkage='average' – как изменилась дендрограмма? 10. Сохранить таблицу с метками кластеров.
8	1. Сгенерировать данные с помощью make_blobs(n_samples=300, centers=5, cluster_std=1.0). 2. Выполнить иерархическую кластеризацию с linkage='ward'. 3. Построить дендрограмму и отсечь на уровне, дающем 5 кластеров. 4. Визуализировать кластеры. 5. Использовать fcluster для получения меток кластеров по порогу расстояния. 6. Эксперимент: изменить distance_threshold в AgglomerativeClustering вместо n_clusters. 7. Сравнить результаты с KMeans. 8. Вычислить силуэт для иерархической кластеризации. 9. Построить график зависимости числа кластеров от порога

	расстояния. 10. Сохранить метки кластеров.
9	1. Загрузить датасет penguins, удалить пропуски, масштабировать числовые признаки. 2. Построить дендрограмму (метод Ward). 3. Выбрать число кластеров = 3 (по количеству видов). 4. Выполнить AgglomerativeClustering(n_clusters=3). 5. Сравнить кластеры с истинными видами (confusion matrix). 6. Построить дендрограмму с использованием linkage='single' – показать эффект цепочки. 7. Визуализировать кластеры в пространстве двух главных компонент. 8. Вычислить силуэтный коэффициент для Ward и single. 9. Определить оптимальное число кластеров по силуэту. 10. Сохранить дендрограмму с цветными ветвями.
10	1. Загрузить датасет wholesale_customers. Масштабировать признаки. 2. Построить матрицу расстояний (евклидово) и тепловую карту с переупорядочиванием (clustermap). 3. Выполнить иерархическую кластеризацию с linkage='ward'. 4. Построить дендрограмму, определить число кластеров визуально. 5. Проинтерпретировать кластеры (средние расходы по категориям). 6. Сравнить с KMeans (силуэтный коэффициент). 7. Использовать linkage='complete' и сравнить дендрограммы. 8. Выделить субкластеры внутри одного из кластеров (рекурсивно). 9. Построить горизонтальную дендрограмму (orientation='top'). 10. Сохранить результат кластеризации в CSV.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.

4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основное отличие иерархической кластеризации от k-means?
2. Что такое агломеративная (восходящая) иерархическая кластеризация? Опишите алгоритм.
3. Что такое дендрограмма? Как её интерпретировать? Как по ней определить число кластеров?
4. Какие методы связи (linkage) существуют в иерархической кластеризации? Кратко опишите single, complete, average, Ward.
5. Что такое эффект «цепочки» (chaining effect) и для какого метода связи он характерен?
6. Чем отличается метод Ward от других методов связи? Какой критерий оптимизирует Ward?
7. Как вычислить матрицу связей (linkage matrix) с помощью scipy.cluster.hierarchy?
8. В чём преимущества и недостатки иерархической кластеризации по сравнению с k-means?
9. Можно ли использовать иерархическую кластеризацию для больших массивов данных? Почему?
10. Как в sklearn и scipy выполнить отсечение дендрограммы по порогу расстояния вместо фиксированного числа кластеров?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Качество построения дендрограммы. Читаемость, подписи, правильное использование параметров.	10
Анализ методов связи. Сравнение single, complete, average,	10

Ward, обоснование выбора.	
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Обоснование выбора числа кластеров, интерпретация кластеров, сравнение с k-means.	10
Визуализации (дендрограмма, тепловая карта, clustermap). Качество, читаемость, подписи.	10

Лабораторная работа № 19

Кластеризация на основе плотности: DBSCAN

Цель: Применить DBSCAN для кластеризации данных сложной формы.

Теоретическое обоснование

Основная идея. Кластеры — области с высокой плотностью точек, разделённые областями низкой плотности. DBSCAN не требует задания числа кластеров и способен находить кластеры произвольной формы.

Типы точек:

- **Core point (ядро):** в окрестности радиуса eps находится не менее min_samples точек (включая саму себя).
- **Border point (граничная):** находится в окрестности core точки, но сама не имеет достаточного числа соседей.
- **Noise point (шум):** не является ни core, ни border.

Алгоритм:

1. Для каждой точки определяем, является ли она core.
2. Каждая core точка порождает кластер: все точки, достижимые по цепочке core-точек (прямая плотностная достижимость).
3. Border точки присоединяются к кластеру любого core-соседа.
4. Noise точки остаются без кластера.

Параметры:

- eps — радиус окрестности. Слишком мал → много шума, слишком велик →

все в одном кластере.

- min_samples — минимальное число точек для core. Чем больше, тем менее чувствителен к локальным шумам.

Выбор eps часто делается с помощью **k-distance графика**: для каждого объекта вычисляется расстояние до его k-го ближайшего соседа (k = min_samples), все расстояния сортируются по убыванию; оптимальный eps — точка «излома» графика.

Преимущества DBSCAN:

- Не требует K.
- Находит кластеры произвольной формы.
- Устойчив к выбросам (маркирует их как шум).

Недостатки: не работает с данными сильно разной плотности; чувствителен к параметрам; плохо масштабируется на высокие размерности (проклятие размерности).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Сгенерировать синтетические данные с помощью make_moons(n_samples=300, noise=0.05, random_state=42). 2. Визуализировать данные. 3. Применить DBSCAN(eps=0.2, min_samples=5). Визуализировать результат (разные цвета кластеров, шум – чёрным). 4. Вывести количество кластеров и количество шумовых точек. 5. Подобрать параметр eps (0.1, 0.2, 0.3, 0.4, 0.5) – для каждого визуализировать и посчитать число кластеров. 6. Подобрать min_samples (2, 5, 10, 20) при фиксированном eps=0.2. 7. Сравнить результат DBSCAN с KMeans(n_clusters=2). 8. Вычислить силуэтный коэффициент для DBSCAN (игнорируя шум). 9. Построить график зависимости числа кластеров от eps. 10. Сохранить лучшую модель DBSCAN (параметры) и метки кластеров.

2	1. Сгенерировать данные с помощью make_circles(n_samples=400, noise=0.05, factor=0.5, random_state=42). 2. Визуализировать данные. 3. Применить DBSCAN(eps=0.15, min_samples=5). Визуализировать. 4. Попробовать KMeans(n_clusters=2) – показать, что k-means не работает. 5. Подобрать eps и min_samples так, чтобы DBSCAN выделил оба круга. 6. Для найденных параметров вывести количество шумовых точек. 7. Визуализировать границы кластеров (через plot_decision_regions для 2D). 8. Сравнить с AgglomerativeClustering(n_clusters=2, linkage='single'). 9. Вычислить силуэтный коэффициент для DBSCAN и для агломеративной кластеризации. 10. Сделать вывод о применимости DBSCAN к данным сложной формы.
3	1. Загрузить датасет iris (только признаки). Масштабировать. 2. Применить DBSCAN(eps=0.5, min_samples=5). Визуализировать (первые два признака или PCA). 3. Вывести количество кластеров и шума. 4. Сравнить с истинными метками (adjusted Rand index). 5. Подобрать eps с помощью анализа k-distance графика (построить график расстояний до k-го соседа). 6. Выбрать оптимальный eps по излому графика. 7. Повторить кластеризацию с оптимальными параметрами. 8. Сравнить с KMeans(n_clusters=3). 9. Построить график зависимости числа кластеров от eps. 10. Сохранить метки кластеров для оптимальных параметров.
4	1. Сгенерировать данные с помощью make_blobs(n_samples=500, centers=4, cluster_std=1.0, random_state=42). 2. Добавить 50 случайных шумовых точек (равномерное распределение по всему пространству). 3. Визуализировать данные. 4. Применить DBSCAN(eps=0.5, min_samples=5). Посмотреть, как выделяются шумовые точки. 5. Подобрать параметры, чтобы DBSCAN выделил 4 кластера и правильно идентифицировал шум.

	6. Визуализировать результат (шум – чёрным). 7. Попробовать KMeans(n_clusters=4) – как он обработает шум? 8. Сравнить время выполнения DBSCAN и KMeans. 9. Вычислить adjusted Rand index для DBSCAN (игнорируя шумовые точки или приравняв их к отдельному кластеру). 10. Сохранить результат.
5	1. Загрузить датасет penguins, удалить пропуски, оставить числовые признаки. 2. Масштабировать признаки. 3. Применить DBSCAN(eps=0.8, min_samples=5). 4. Визуализировать кластеры в пространстве двух главных компонент. 5. Вывести количество кластеров и шума. 6. Сравнить с истинными видами пингвинов (confusion matrix). 7. Подобрать eps (0.5, 0.8, 1.0, 1.2) и min_samples (3,5,10) через GridSearchCV (обёртка не предусмотрена, можно вручную). 8. Выбрать параметры, максимизирующие silhouette score. 9. Построить график зависимости silhouette score от eps. 10. Сохранить лучшую конфигурацию.
6	1. Сгенерировать данные с помощью make_classification(n_samples=300, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, n_classes=3, random_state=42). 2. Визуализировать данные (цвета – истинные классы). 3. Применить DBSCAN(eps=0.3, min_samples=4). Визуализировать. 4. Показать, что DBSCAN может выделить больше кластеров, чем истинных классов (если параметры не подобраны). 5. Подобрать параметры так, чтобы количество кластеров совпало с числом истинных классов. 6. Сравнить adjusted Rand index с KMeans(n_clusters=3). 7. Построить k-distance график для выбора eps. 8. Эксперимент: изменить metric='manhattan' и сравнить результат с евклидовой метрикой. 9. Сделать вывод о чувствительности DBSCAN к метрике расстояния. 10. Сохранить визуализацию с границами кластеров.
7	1. Загрузить датасет digits (500 образцов, 64 признака). 2. Применить PCA для снижения размерности до 2 компонент (для визуализации).

	3. Выполнить DBSCAN на исходных 64-мерных данных (параметры eps=5, min_samples=5). 4. Визуализировать результат в пространстве двух главных компонент. 5. Вывести количество кластеров и шума. 6. Подобрать eps с помощью k-distance графика (на 64-мерных данных). 7. Сравнить с истинными цифрами (adjusted Rand index). 8. Сравнить с KMeans(n_clusters=10). 9. Построить матрицу соответствия кластеров и истинных цифр. 10. Сделать вывод о том, насколько DBSCAN применим к многомерным данным.
8	1. Сгенерировать данные с помощью make_blobs(n_samples=200, centers=3, cluster_std=0.6). 2. Добавить 20 выбросов (случайные точки далеко от кластеров). 3. Применить DBSCAN(eps=0.4, min_samples=3). 4. Визуализировать, показать, что выбросы отмечены как шум. 5. Попробовать DBSCAN(eps=0.4, min_samples=10) – как изменилось число шумовых точек? 6. Объяснить, почему увеличение min_samples приводит к тому, что некоторые периферийные точки становятся шумом. 7. Для сравнения выполнить KMeans(n_clusters=3) и показать, что выбросы искажают центроиды. 8. Вычислить silhouette score для DBSCAN (без шума) и для KMeans. 9. Построить график зависимости числа шумовых точек от min_samples. 10. Сохранить метки кластеров.
9	1. Загрузить датасет wholesale_customers. Масштабировать признаки. 2. Применить DBSCAN(eps=2.5, min_samples=5). 3. Вывести количество кластеров и шума. 4. Визуализировать кластеры с помощью PCA (первые две компоненты). 5. Проинтерпретировать полученные кластеры (средние расходы по категориям). 6. Подобрать параметры eps и min_samples с помощью силуэтного анализа (перебором). 7. Сравнить с KMeans (оптимальное k подобрать по силуэту). 8. Определить, какой метод даёт более интерпретируемые кластеры.

	9. Построить график зависимости силуэта от eps. 10. Сохранить таблицу с метками кластеров для лучших параметров.
10	1. Сгенерировать данные с помощью make_moons(n_samples=200, noise=0.05) и make_circles(n_samples=200, noise=0.05, factor=0.5), объединить в один датасет. 2. Визуализировать данные (две фигуры вместе). 3. Применить DBSCAN(eps=0.2, min_samples=5). Визуализировать. 4. Должны ли выделиться два кластера (луна и круг)? Почему может не получиться? 5. Подобрать параметры, чтобы DBSCAN выделил обе фигуры как отдельные кластеры (возможно, не получится из-за разной плотности). 6. Объяснить ограничение DBSCAN: разные плотности кластеров. 7. Попробовать OPTICS (альтернатива DBSCAN) и сравнить. 8. Сравнить с AgglomerativeClustering(n_clusters=2, linkage='single'). 9. Вычислить silhouette score для DBSCAN и OPTICS. 10. Сделать вывод о применимости DBSCAN при разных плотностях.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основная идея DBSCAN? Какие типы точек выделяются (core, border, noise)?
2. Что означают параметры eps и min_samples в DBSCAN? Как они влияют на результат?
3. Как выбрать оптимальное значение eps? Что такое k-distance график?
4. Как DBSCAN обрабатывает выбросы (шум)? В чём преимущество перед k-means?
5. Какие формы кластеров может обнаруживать DBSCAN? Приведите примеры.
6. В чём недостаток DBSCAN? Когда он неэффективен (разная плотность, высокая размерность)?
7. Чем DBSCAN отличается от OPTICS?
8. Как оценить качество кластеризации DBSCAN, если есть шумовые точки? Как быть с силуэтным коэффициентом?
9. Как DBSCAN работает с метриками расстояния, отличными от евклидовой?
10. Почему DBSCAN не требует задания числа кластеров заранее? Как он определяет их автоматически?

Критерии оценивания

Параметр	**Баллы**
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора параметров (eps, min_samples). Использование k-distance графика или перебора, обоснование выбора.	10
Демонстрация работы с шумом. Выделение шумовых точек, интерпретация.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и сравнение с k-means. Анализ преимуществ DBSCAN для данных сложной формы, сравнение качества.	10
Визуализации (кластеры + шум, k-distance график). Качество,	10

читаемость, подписи.	

Лабораторная работа № 20

Снижение размерности: PCA и t-SNE

Цель: Применить методы снижения размерности для визуализации и предобработки.

Теоретическое обоснование

PCA (Principal Component Analysis) — линейный метод, проектирующий данные на оси максимальной дисперсии.

1. Центрируем данные (вычитаем среднее).
2. Вычисляем ковариационную матрицу.
3. Находим её собственные векторы (главные компоненты) и собственные значения.
4. Сортируем компоненты по убыванию собственных значений.
5. Проектируем данные на первые k компонент.

Объяснённая дисперсия для i-й компоненты = $\lambda_i / \Sigma\lambda_j$. Накопленная дисперсия позволяет выбрать число компонент (например, 95%).

Применение PCA: снижение размерности для ускорения обучения, визуализация (2D/3D), сжатие данных, удаление шума.

t-SNE (t-distributed Stochastic Neighbor Embedding) — нелинейный метод для визуализации, сохраняющий локальную структуру.

- Преобразует евклидовы расстояния между точками в условные вероятности (гауссово ядро).
- В низкоразмерном пространстве моделирует распределение t-распределением Стьюдента (с тяжёлыми хвостами).
- Минимизирует расхождение Кульбака-Лейблера между распределениями.

Параметр perplexity — баланс между локальным и глобальным аспектами (обычно 5–50). Низкий perplexity выявляет мелкие кластеры, высокий — глобальную структуру.

Важные замечания по t-SNE:

- Не детерминирован (разные запуски дают разную картинку).
- Не предназначен для предобработки перед обучением (нет отображения для новых точек).
- Сложность $O(N^2)$ — для больших данных требуется субсэмплирование или Barnes-Hut.
- Не сохраняет глобальные расстояния (размеры кластеров могут быть произвольными).

Сравнение: PCA — быстрый, линейный, детерминированный, сохраняет глобальную структуру. t-SNE — медленный, нелинейный, стохастический, отлично показывает локальные кластеры.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет digits (8×8 изображения) из sklearn. 2. Применить PCA без указания числа компонент, вывести долю объяснённой дисперсии для каждой компоненты. 3. Построить график накопленной доли объяснённой дисперсии (cumulative explained variance). 4. Выбрать число компонент, объясняющих 95% дисперсии. 5. Преобразовать данные с помощью PCA до выбранного числа компонент. 6. Визуализировать данные в пространстве первых двух главных компонент (scatter plot, цвет по цифре). 7. Применить t-SNE с perplexity=30, n_components=2. Визуализировать результат. 8. Сравнить визуализацию PCA и t-SNE (какой метод лучше разделяет цифры). 9. Вычислить время выполнения PCA и t-SNE. 10. Сохранить преобразованные данные (PCA и t-SNE) в CSV.
2	1. Загрузить датасет iris из sklearn. 2. Выполнить PCA с 2 компонентами. Визуализировать точки в пространстве PC1-PC2, раскрасив по видам. 3. Вывести loadings (коэффициенты) для каждой главной компоненты – какие признаки вносят наибольший вклад. 4. Построить biplot (график с векторами исходных признаков на плоскости PCA).

	5. Применить t-SNE с perplexity=20, n_components=2. Визуализировать. 6. Сравнить разделение классов на PCA и t-SNE. 7. Эксперимент: изменить perplexity на 5, 30, 50 – как меняется визуализация? 8. Вычислить silhouette score для PCA-данных (по истинным меткам) и для t-SNE. 9. Сделать вывод о сохранении глобальной и локальной структуры. 10. Сохранить графики сравнения.
3	1. Загрузить датасет wine из sklearn. Масштабировать признаки. 2. Выполнить PCA и построить график объяснённой дисперсии (scree plot). 3. Определить оптимальное число компонент по правилу «локтя». 4. Спроецировать данные на первые 2 компоненты и визуализировать. 5. Выполнить t-SNE с perplexity=30, random_state=42. 6. Сравнить качество разделения классов визуально. 7. Выполнить кластеризацию k-means на исходных данных и на PCA-данных (2 компоненты). Сравнить adjusted Rand index с истинными метками. 8. Повторить кластеризацию на t-SNE-данных – почему это плохая идея? 9. Построить график зависимости объяснённой дисперсии от числа компонент. 10. Сохранить PCA-модель в файл.
4	1. Загрузить датасет mnist_784 (1000 образцов). 2. Нормализовать пиксели (разделить на 255). 3. Применить PCA с n_components=2. Визуализировать (цвет по цифрам). 4. Применить PCA с n_components=50 (сохраняющим ~80% дисперсии). 5. На восстановленных из 50 компонент данных визуализировать несколько изображений – сравнить с оригиналами. 6. Вычислить среднеквадратичную ошибку реконструкции (MSE) для разного числа компонент (2, 10, 50, 100). 7. Построить график зависимости MSE от числа компонент. 8. Применить t-SNE с perplexity=30 к исходным данным (1000 образцов) – замерить время. 9. Сравнить визуализации PCA и t-SNE (качество разделения).

	10. Сохранить проекции PCA и t-SNE.
5	1. Загрузить датасет olivetti_faces (изображения лиц). 2. Применить PCA без указания числа компонент. 3. Построить график накопленной дисперсии – определить число компонент для 90% дисперсии. 4. Визуализировать первые 10 главных компонент как изображения («собственные лица» – eigenfaces). 5. Восстановить изображения по 10, 50, 100 компонентам – визуализировать сравнение (оригинал и восстановленное). 6. Применить t-SNE с perplexity=30 к исходным данным. 7. Визуализировать t-SNE результат (цвет по людям, если есть метки). 8. Сравнить, какой метод лучше сохраняет структуру «один человек – близкие точки». 9. Вычислить время выполнения PCA (fit + transform) и t-SNE. 10. Сохранить первые 10 eigenfaces в виде изображений.
6	1. Сгенерировать данные с высокой размерностью: make_classification(n_samples=500, n_features=50, n_informative=20, random_state=42). 2. Разделить данные на train/test. 3. Обучить KNeighborsClassifier(n_neighbors=5) на исходных данных (50 признаков). Замерить accuracy и время. 4. Применить PCA с сохранением 95% дисперсии – сколько компонент осталось? 5. Обучить kNN на PCA-данных. Сравнить accuracy и время. 6. Применить t-SNE с n_components=2 только для визуализации. 7. Сравнить время выполнения PCA и t-SNE. 8. Обучить логистическую регрессию на исходных и PCA-данных – сравнить качество. 9. Сделать вывод о влиянии снижения размерности на качество классификации. 10. Сохранить PCA-трансформер в joblib.
7	1. Загрузить датасет breast_cancer из sklearn. 2. Масштабировать признаки. 3. Выполнить PCA с 2 компонентами и визуализировать точки, раскрасив по диагнозу. 4. Построить график зависимости объяснённой дисперсии от числа компонент. 5. Выполнить t-SNE с perplexity=15. Визуализировать. 6. Обучить SVC(kernel='rbf') на исходных данных и на PCA-данных (2 компоненты). Сравнить accuracy.

	7. Объяснить, почему PCA с 2 компонентами может снизить качество. 8. Выполнить PCA с 10 компонентами и снова обучить SVM – сравнить. 9. Построить график зависимости accuracy от числа компонент PCA (от 2 до 20). 10. Сохранить лучшую PCA-модель.
8	1. Загрузить датасет fashion_mnist (2000 образцов) через fetch_openml. 2. Нормализовать пиксели. 3. Визуализировать несколько изображений из разных классов. 4. Применить PCA с 2 компонентами, визуализировать (цвет по классам). 5. Применить t-SNE с perplexity=40, n_components=2, random_state=42. 6. Сравнить визуализации – какой метод лучше разделяет одежду? 7. Использовать PCA для сжатия до 50 компонент, затем применить t-SNE к PCA-данным (ускорение). 8. Сравнить t-SNE на исходных и на PCA-сжатых данных (визуально). 9. Замерить время t-SNE на исходных данных и на PCA-сжатых. 10. Сохранить PCA + t-SNE Pipeline.
9	1. Загрузить датасет seeds (пшеница). Масштабировать. 2. Выполнить PCA и построить график нагрузок (loadings) – определить, какие признаки коррелируют с PC1. 3. Спроецировать данные на PC1 и PC2, визуализировать. 4. Выполнить t-SNE с разной perplexity (5, 20, 50, 100). Построить сетку графиков (2×2). 5. Для каждой perplexity вычислить силуэтный коэффициент (по истинным меткам). 6. Выбрать оптимальную perplexity, максимизирующую силуэт. 7. Сравнить разделение классов на PCA и на лучшем t-SNE. 8. Объяснить, почему t-SNE не используется для предобработки перед классификацией. 9. Сохранить t-SNE визуализацию с лучшей perplexity. 10. Сделать вывод о чувствительности t-SNE к параметрам.
10	1. Загрузить датасет hover (многомерный, из UCI) или другой с числом признаков > 20. 2. Вычислить корреляционную матрицу признаков, визуализировать heatmap.

	3. Выполнить PCA и вывести долю объяснённой дисперсии для первых 5 компонент. 4. Построить biplot (первые две компоненты + векторы признаков). 5. Проинтерпретировать: какие признаки наиболее важны для PC1 и PC2. 6. Выполнить t-SNE с perplexity=30 и визуализировать. 7. Сравнить, какие структуры данных видны на PCA и на t-SNE. 8. Выполнить кластеризацию k-means (k=3) на исходных данных и на PCA-данных – сравнить силуэт. 9. Сделать вывод о том, какой метод снижения размерности лучше для визуализации. 10. Сохранить проекции.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основная идея метода главных компонент (PCA)? Как находятся главные компоненты?
2. Что такое объяснённая дисперсия (explained variance) и как её использовать для выбора числа компонент?
3. В чём разница между PCA и t-SNE по целям и принципу работы?
4. Почему t-SNE не подходит для предобработки данных перед обучением моделей?
5. Что такое параметр perplexity в t-SNE? Как он влияет на результат?

6. В чём недостатки t-SNE (стохастичность, вычислительная сложность, чувствительность к параметрам)?
7. Как интерпретировать loadings (коэффициенты) в PCA?
8. Что такое «собственные лица» (eigenfaces) и как они получаются с помощью PCA?
9. Какой метод снижения размерности лучше сохраняет глобальную структуру, а какой – локальную?
10. Можно ли использовать PCA для сжатия данных с последующим восстановлением? Как оценить качество восстановления?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность выбора числа компонент PCA. Использование графика накопленной дисперсии или правила «локтя».	10
Качество визуализаций (PCA и t-SNE). Читаемость, подписи, сравнение на одном графике (если требуется).	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Сравнение PCA и t-SNE, анализ параметров, обоснование выбора.	10
Эксперименты с параметрами (perplexity, число компонент). Демонстрация влияния параметров на результат.	10

Лабораторная работа № 21

Введение в TensorFlow/Keras. Многослойный перцептрон для классификации

Цель: Реализовать простую нейронную сеть с использованием Keras.

Теоретическое обоснование

Многослойный перцептрон (MLP) — полносвязная нейронная сеть. Состоит из:

- **Входной слой:** число нейронов равно числу признаков (для MNIST — 784 после Flatten).
- **Скрытые слои:** Dense слои с нелинейной функцией активации (ReLU, tanh, sigmoid).
- **Выходной слой:** для многоклассовой классификации — Dense(число классов, activation='softmax').

Flatten преобразует многомерный вход (например, 28×28) в одномерный вектор.

Функции активации:

- **ReLU (Rectified Linear Unit):** f(x) = max(0, x). Быстрая, решает проблему исчезающего градиента.
- **Softmax:** превращает выходы в вероятности, сумма по классам = 1.
- **Sigmoid:** для бинарной классификации (выходной слой).

Компиляция: model.compile(optimizer, loss, metrics)

- Оптимизаторы: adam (адаптивный, хорош по умолчанию), sgd (стохастический градиентный спуск), rmsprop.
- Функции потерь: categorical_crossentropy (метки one-hot), sparse_categorical_crossentropy (метки целые числа), binary_crossentropy.

Обучение: model.fit(X_train, y_train, epochs, batch_size, validation_split, callbacks)

- epochs — число полных проходов по данным.
- batch_size — число объектов перед обновлением весов.
- validation_split — доля выборки для валидации.

Callback'и: EarlyStopping (остановка при

ухудшении), ModelCheckpoint (сохранение лучшей модели), ReduceLROnPlateau (уменьшение lr при затухании).

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Установить TensorFlow (pip install tensorflow). Загрузить датасет MNIST через keras.datasets.mnist.load_data(). 2. Провести предобработку: нормализовать пиксели (разделить на 255.0) и преобразовать метки в формат one-hot encoding. 3. Построить модель Sequential со следующей архитектурой: Flatten → Dense(128, activation='relu') → Dense(10, activation='softmax'). 4. Скомпилировать модель с оптимизатором adam, функцией потерь categorical_crossentropy и метрикой accuracy. 5. Обучить модель на 5 эпохах с размером батча 32, используя 20% обучающей выборки как валидационную. 6. Построить графики точности и функции потерь на обучении и валидации по эпохам. 7. Оценить модель на тестовой выборке, вывести accuracy. 8. Построить confusion matrix для тестовой выборки в виде тепловой карты. 9. Использовать model.predict() для первых 5 изображений тестовой выборки, сравнить предсказанные и истинные метки. 10. Сохранить модель в формате .h5 и загрузить обратно, проверив предсказания.
2	1. Загрузить датасет Fashion-MNIST (keras.datasets.fashion_mnist.load_data()). 2. Нормализовать данные и преобразовать метки в one-hot. 3. Создать модель Sequential с архитектурой: Flatten → Dense(256, activation='relu') → Dropout(0.5) → Dense(10, activation='softmax'). 4. Скомпилировать модель с оптимизатором sgd (скорость обучения 0.01), функцией потерь categorical_crossentropy и метрикой accuracy. 5. Обучить модель на 10 эпохах, добавив callback EarlyStopping(patience=3, monitor='val_loss'). 6. Построить графики accuracy и loss на обучении и валидации. 7. Вычислить accuracy, precision, recall, F1 для каждого класса на

	тестовой выборке. 8. Построить confusion matrix. 9. Сравнить результаты с вариантом 1 (MNIST). 10. Сохранить модель в формате SavedModel (.pb).
3	1. Загрузить датасет MNIST. 2. Создать модель с тремя скрытыми слоями: Flatten → Dense(512, activation='relu') → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam, функцией потерь sparse_categorical_crossentropy (метки не преобразовывать в one-hot). 4. Обучить модель на 20 эпохах. 5. Построить графики обучения. 6. Эксперимент: обучить такую же модель, но с активацией tanh вместо relu. Сравнить accuracy. 7. Визуализировать архитектуру модели с помощью plot_model. 8. Вывести количество обучаемых параметров в модели (model.summary()). 9. Использовать ModelCheckpoint для сохранения лучшей модели по валидационной accuracy. 10. Загрузить сохранённую лучшую модель и оценить на тесте.
4	1. Загрузить датасет Fashion-MNIST. 2. Нормализовать данные, метки оставить как целые числа. 3. Построить модель: Flatten → Dense(128, activation='relu') → Dense(64, activation='relu') → Dense(10, activation='softmax'). 4. Скомпилировать модель с оптимизатором RMSprop(learning_rate=0.001), функцией потерь sparse_categorical_crossentropy и метрикой accuracy. 5. Обучить модель на 15 эпохах с валидационной выборкой (20%). 6. Построить графики обучения. 7. Вывести classification report для тестовой выборки. 8. Визуализировать 10 тестовых изображений (по одному на класс) вместе с предсказанными и истинными метками. 9. Эксперимент: изменить размер батча на 64 и 128, сравнить время обучения и accuracy. 10. Сохранить модель в JSON (архитектуру) и HDF5 (веса).
5	1. Загрузить датасет MNIST. 2. Создать модель с одним скрытым слоем: Flatten → Dense(100, activation='relu') → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam, функцией

	потерь categorical_crossentropy. 4. Обучить модель с различным числом нейронов в скрытом слое: 50, 100, 200, 500. Для каждого варианта обучить 5 эпох и сравнить accuracy на тесте. 5. Построить график зависимости accuracy от числа нейронов. 6. Для лучшей конфигурации обучить модель на 20 эпохах с ReduceLROnPlateau (уменьшение learning rate при затухании валидационной accuracy). 7. Вывести историю обучения. 8. Построить матрицу ошибок. 9. Визуализировать веса первого скрытого слоя (как изображения 28×28). 10. Сделать вывод о том, как число нейронов влияет на качество.
6	1. Загрузить датасет Fashion-MNIST. 2. Создать модель с архитектурой: Flatten → Dense(256, activation='relu') → BatchNormalization() → Dense(128, activation='relu') → Dropout(0.3) → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam. 4. Обучить модель на 20 эпохах. 5. Сравнить с моделью без BatchNormalization и Dropout (такой же архитектуры). Построить графики точности для обеих моделей на одном поле. 6. Вычислить время обучения обеих моделей. 7. Для лучшей модели построить confusion matrix. 8. Визуализировать 10 изображений, которые модель предсказала неправильно (с наибольшей уверенностью в неправильном классе). 9. Сохранить лучшую модель. 10. Сделать вывод о влиянии регуляризации на качество.
7	1. Загрузить датасет MNIST. 2. Создать модель с двумя скрытыми слоями: Flatten → Dense(128, activation='sigmoid') → Dense(64, activation='sigmoid') → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam. 4. Обучить модель на 10 эпохах. Замерить accuracy. 5. Заменить сигмоиду на ReLU, обучить заново. Сравнить accuracy и скорость сходимости (по графикам loss). 6. Заменить сигмоиду на tanh, обучить заново. Сравнить. 7. Построить графики сравнения функций активации. 8. Для ReLU-модели построить confusion matrix.

	9. Эксперимент: добавить второй скрытый слой с 256 нейронами и сравнить. 10. Сделать вывод о влиянии выбора функции активации.
8	1. Загрузить датасет MNIST. 2. Создать модель с архитектурой: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель с оптимизатором SGD с learning rate 0.01. 4. Обучить модель с оптимизатором Adam с learning rate 0.001. 5. Сравнить графики сходимости (loss) и итоговую accuracy. 6. Построить графики сравнения оптимизаторов. 7. Для Adam-модели вывести classification report. 8. Использовать keras.utils.plot_model для визуализации архитектуры. 9. Сохранить модель и конвертировать её в TensorFlow Lite (.tflite). 10. Сделать вывод о выборе оптимизатора для данной задачи.
9	1. Загрузить датасет Fashion-MNIST. 2. Создать модель с архитектурой: Flatten → Dense(512, activation='relu') → Dropout(0.2) → Dense(256, activation='relu') → Dropout(0.2) → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam. 4. Обучить модель с разными уровнями Dropout: 0 (без dropout), 0.2, 0.5, 0.7. Для каждого обучить 10 эпох и сравнить accuracy на тесте и валидации. 5. Построить график зависимости accuracy от уровня Dropout. 6. Выбрать оптимальный уровень Dropout и обучить модель на 30 эпохах. 7. Построить confusion matrix. 8. Визуализировать кривые обучения (loss и accuracy) для выбранной модели. 9. Сохранить модель. 10. Сделать вывод о влиянии Dropout на переобучение.
10	1. Загрузить датасет MNIST. 2. Создать модель: Flatten → Dense(256, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель на 5 эпохах, затем на 10, 15, 20. Для каждого случая оценить accuracy на тесте. 4. Построить график зависимости accuracy на тесте от числа эпох. 5. Использовать EarlyStopping(patience=3, monitor='val_loss') и

	обучить модель на 50 эпохах – на какой эпохе остановится? 6. Построить графики обучения с EarlyStopping. 7. Для модели, обученной с EarlyStopping, построить матрицу ошибок. 8. Сравнить accuracy модели с EarlyStopping и модели, обученной фиксированным числом эпох (20). 9. Сохранить лучшую модель. 10. Сделать вывод о пользе ранней остановки.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Какова структура многослойного перцептрона (MLP)? Какие слои он содержит и для чего служит каждый?
2. Какие функции активации чаще всего используются в скрытых слоях MLP и на выходном слое для многоклассовой классификации? Почему?
3. В чём разница между функциями потерь `categorical_crossentropy` и `sparse_categorical_crossentropy`? Когда какую использовать?
4. Что делает метод `model.compile()`? Какие обязательные параметры он принимает?
5. Как параметры `batch_size` и `epochs` влияют на процесс обучения нейронной сети?
6. Что такое валидационная выборка и зачем она используется при обучении?

7. Какие callback-функции Keras вы знаете? Для чего нужны EarlyStopping и ModelCheckpoint?
8. Что такое one-hot encoding меток и для чего он нужен при классификации?
9. Как оценить качество модели классификации на тестовой выборке? Какие метрики используют?
10. В чём отличие между сохранением модели в форматах .h5 и SavedModel (.pb)? Как загрузить сохранённую модель?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность построения модели. Корректное использование Sequential API, выбор слоёв, функций активации, входной размерности.	10
Качество визуализаций (графики обучения, матрица ошибок). Наличие подписей, заголовков, легенд, читаемость.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ полученных результатов, сравнение экспериментов, обоснование выбора архитектуры/гиперпараметров.	10
Сохранение и загрузка модели. Корректное сохранение модели в указанном формате и восстановление для предсказаний.	10

Лабораторная работа № 22

Многослойный перцептрон для регрессии

Цель: Применить нейронную сеть для задачи регрессии.

Теоретическое обоснование

Отличия от классификации:

- **Выходной слой:** один нейрон (если прогнозируется одна величина) с **линейной активацией** (activation='linear' или без указания).
- **Функция потерь:** mean_squared_error (MSE) или mean_absolute_error (MAE).
- **Метрики:** mae, mse, root_mean_squared_error (можно вычислить отдельно).

Особенности архитектуры: количество нейронов в скрытых слоях обычно меньше, чем для классификации (зависит от сложности зависимости). Глубокие сети могут быть полезны для сложных регрессионных поверхностей.

Нормализация целевой переменной: иногда полезно (если у имеет большой разброс), но предсказания потом нужно будет вернуть в исходную шкалу. Можно стандартизовать y, а затем использовать inverse_transform.

Оценка качества: те же метрики, что и для линейной регрессии: MSE, MAE, RMSE, R^2. Для R^2 в Keras нет встроенной метрики, можно реализовать самостоятельно или вычислить после предсказаний.

Сравнение с линейной регрессией: MLP может улавливать нелинейности, но требует больше данных и настройки. На простых линейных данных может переобучаться или быть нестабильным.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет california_housing из sklearn.datasets.fetch_california_housing(). 2. Разделить данные на обучающую (80%) и тестовую (20%) выборки. 3. Нормализовать признаки с помощью StandardScaler (обучить на train, применить к train и test).

	4. Построить модель Sequential с архитектурой: Dense(64, activation='relu') → Dense(32, activation='relu') → Dense(1, activation='linear'). 5. Скомпилировать модель с оптимизатором adam, функцией потерь mse и метрикой mae. 6. Обучить модель на 50 эпохах с размером батча 32, используя 20% обучающей выборки как валидационную. 7. Построить графики MSE и MAE на обучении и валидации по эпохам. 8. Оценить модель на тестовой выборке: вычислить MSE, MAE, RMSE, R^2. 9. Построить график «фактические vs предсказанные значения» для тестовой выборки. 10. Сохранить модель в формате .h5 и загрузить обратно, проверив предсказания.
2	1. Загрузить датасет boston_housing (через keras.datasets.boston_housing.load_data() или fetch_openml). 2. Разделить на train/test (80/20). 3. Нормализовать признаки. 4. Построить модель: Dense(128, activation='relu') → Dropout(0.2) → Dense(64, activation='relu') → Dense(1). 5. Скомпилировать модель с оптимизатором RMSprop(learning_rate=0.001), функцией потерь mae и метрикой mse. 6. Обучить модель с EarlyStopping(patience=10, monitor='val_loss'). 7. Построить графики обучения. 8. Вычислить метрики на тесте: MAE, RMSE, R^2. 9. Сравнить результаты с линейной регрессией (обучить LinearRegression на тех же данных). 10. Сохранить лучшую модель.
3	1. Загрузить датасет diabetes (регрессия) из sklearn. 2. Разделить на train/test (75/25). 3. Нормализовать признаки. 4. Построить модель: Dense(32, activation='relu') → Dense(16, activation='relu') → Dense(1). 5. Скомпилировать модель с оптимизатором adam, функцией потерь mse. 6. Обучить модель на 100 эпохах, добавив ReduceLROnPlateau(factor=0.5, patience=10). 7. Построить график предсказанных vs истинных значений.

	8. Вычислить R^2 и сравнить с линейной регрессией. 9. Эксперимент: добавить ещё один скрытый слой (64 нейрона) и сравнить качество. 10. Сохранить модель в формате SavedModel.
4	1. Сгенерировать синтетические данные: y = 3*x + 2*x^2 + np.sin(2*pi*x) + шум. 2. Разделить на train/test (80/20). 3. Нормализовать признаки (x – одномерный, можно не масштабировать). 4. Построить модель с двумя скрытыми слоями (64 и 32 нейрона, ReLU). 5. Обучить модель на 200 эпох. 6. Построить график исходной функции и предсказаний модели. 7. Эксперимент: увеличить число нейронов до 256 – наблюдать переобучение (график с осцилляциями). 8. Добавить Dropout(0.2) и BatchNormalization – сравнить с моделью без регуляризации. 9. Вычислить MSE и R^2 для каждой модели. 10. Сохранить лучшую модель.
5	1. Загрузить датасет concrete (прочность бетона) из открытого источника. 2. Разделить на train/test (80/20). 3. Нормализовать признаки. 4. Построить модель: Dense(128, activation='relu') → Dense(64, activation='relu') → Dense(32, activation='relu') → Dense(1). 5. Скомпилировать модель с оптимизатором adam и функцией потерь mse. 6. Обучить модель с ModelCheckpoint для сохранения лучших весов по валидационной loss. 7. Построить графики обучения. 8. Оценить модель на тесте (MAE, RMSE, R^2). 9. Сравнить с RandomForestRegressor(n_estimators=100). 10. Сохранить лучшую модель.
6	1. Загрузить датасет airfoil (уровень шума) из openml. 2. Разделить на train/test (75/25). 3. Нормализовать признаки. 4. Построить модель с архитектурой: Dense(64, activation='tanh') → Dense(32, activation='tanh') → Dense(1). 5. Скомпилировать модель с оптимизатором SGD(learning_rate=0.01, momentum=0.9) и функцией потерь mae. 6. Обучить модель на 150 эпох.

	7. Построить график остатков (residuals) для тестовой выборки. 8. Эксперимент: заменить tanh на ReLU – сравнить скорость сходимости. 9. Вычислить R^2 и сравнить с линейной регрессией. 10. Сохранить модель.
7	1. Загрузить датасет energy_efficiency (нагрузка на отопление). 2. Выбрать целевую переменную heating_load. 3. Разделить на train/test (80/20). 4. Нормализовать признаки. 5. Построить модель с одним скрытым слоем (50 нейронов, ReLU). 6. Обучить модель при разной скорости обучения (0.001, 0.01, 0.1) – сравнить итоговую loss. 7. Построить графики сходимости для разных learning rate. 8. Выбрать лучшую скорость обучения и обучить модель на 200 эпох. 9. Вычислить метрики на тесте и построить scatter plot. 10. Сохранить модель.
8	1. Загрузить датасет wine_quality (красное вино). 2. Разделить на train/test (80/20). 3. Нормализовать признаки. 4. Построить модель: Dense(64, activation='relu') → Dense(32, activation='relu') → Dense(1). 5. Скомпилировать модель с оптимизатором adam и функцией потерь mse. 6. Обучить модель на 50 эпох, использовать валидационную выборку (20%). 7. Построить график фактических vs предсказанных значений (добавить jitter из-за дискретности оценки). 8. Эксперимент: преобразовать целевую переменную в логарифм и обучить модель – сравнить RMSE в исходной шкале. 9. Вычислить корреляцию Пирсона между предсказанными и истинными значениями. 10. Сохранить модель.
9	1. Загрузить датасет house_prices (Kaggle или синтетический). 2. Разделить на train/test (80/20). 3. Нормализовать признаки. 4. Построить модель с тремя скрытыми слоями (256, 128, 64 нейрона, ReLU). 5. Добавить BatchNormalization после каждого скрытого слоя. 6. Скомпилировать модель с оптимизатором adam. 7. Обучить модель с ранней остановкой (patience=15).

	8. Построить графики обучения. 9. Оценить модель на тесте (R^2, RMSE). 10. Сравнить с моделью без BatchNormalization (аналогичной архитектуры) – построить графики сравнения.
10	1. Загрузить датасет medical_cost (медицинские расходы). 2. Разделить на train/test (80/20). 3. Нормализовать признаки. 4. Построить модель: Dense(128, activation='relu') → Dropout(0.3) → Dense(64, activation='relu') → Dense(1). 5. Скомпилировать модель с оптимизатором adam и функцией потерь mae. 6. Обучить модель на 100 эпох с валидацией. 7. Построить график остатков (предсказанные vs остатки). 8. Вычислить метрики MAE, RMSE, R^2. 9. Эксперимент: увеличить Dropout до 0.5 – изменилось ли качество? 10. Сохранить лучшую модель.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Чем отличается архитектура MLP для регрессии от MLP для классификации (выходной слой, функция активации, функция потерь)?
2. Почему для регрессии на выходном слое используют activation='linear' (или без активации)?
3. Какие функции потерь используются в регрессионных задачах? В чём разница между MSE, MAE и RMSE?

4. Как оценить качество модели регрессии? Назовите 3 метрики и опишите их интерпретацию.
5. Почему важно нормализовать входные признаки для нейронной сети в регрессии?
6. Что такое R^2 (коэффициент детерминации) и как его интерпретировать (в том числе отрицательные значения)?
7. Как построить график «фактические vs предсказанные значения»? Что можно увидеть на этом графике?
8. Как график остатков (residuals) помогает диагностировать проблемы модели регрессии?
9. Как выбрать количество скрытых слоёв и нейронов для задачи регрессии? Какие есть эмпирические правила?
10. В чём преимущества нейросетевой регрессии перед линейной регрессией? В каких случаях нейронная сеть может быть хуже?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность построения модели регрессии. Корректный выходной слой (линейный), выбор функции потерь, метрик.	10
Нормализация данных. Обязательное применение StandardScaler (или аналогичного) перед обучением.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ метрик, сравнение с линейной регрессией, обсуждение графиков остатков.	10
Визуализации (графики обучения, фактические vs предсказанные, остатки). Качество, подписи, читаемость.	10

Лабораторная работа № 23

Настройка гиперпараметров нейронной сети

Цель: Исследовать влияние архитектуры и гиперпараметров на качество модели.

Теоретическое обоснование

Гиперпараметры, влияющие на обучение:

- **Архитектура:** число слоёв, число нейронов в каждом слое.
- **Функции активации:** ReLU, LeakyReLU, ELU, tanh, sigmoid.
- **Скорость обучения (learning rate):** от 0.0001 до 0.1. Слишком малая — медленная сходимость, слишком большая — расходимость.
- **Размер батча (batch size):** малые батчи (16–32) дают шумные градиенты (лучшая обобщающая способность), большие (128–512) — стабильнее и быстрее.
- **Количество эпох** — контролируется ранней остановкой.
- **Регуляризация:** Dropout, L1/L2, BatchNormalization.

Методы подбора гиперпараметров:

- **GridSearchCV** — полный перебор по сетке. Эффективен при малом числе параметров.
- **RandomizedSearchCV** — случайный поиск в заданных распределениях. Более эффективен в большом пространстве.
- **keras-tuner** — специализированный инструмент для Keras (RandomSearch, Hyperband, BayesianOptimization).

Обёртка KerasClassifier / KerasRegressor позволяет использовать модели Keras в sklearn-функциях поиска. Нужно определить функцию create_model(hidden_size=128, lr=0.001), возвращающую скомпилированную модель.

Кросс-валидация для нейронных сетей ресурсоёмка, но часто используется с небольшим числом фолдов (3–5).

TensorBoard — визуализация метрик, гистограмм весов, графов модели в реальном времени. Запускается через callback TensorBoard.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет Fashion-MNIST. Нормализовать пиксели. 2. Построить базовую модель: Flatten → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam и sparse_categorical_crossentropy. Обучить на 5 эпох для оценки. 4. Создать функцию build_model(hidden_size=128, activation='relu'), возвращающую скомпилированную модель. 5. Использовать GridSearchCV (через KerasClassifier) для подбора hidden_size (64, 128, 256) и activation ('relu', 'tanh'). 6. Вывести лучшие параметры и accuracy. 7. Обучить лучшую модель на 20 эпохах, построить графики обучения. 8. Сравнить accuracy на тесте с базовой моделью. 9. Визуализировать матрицу ошибок. 10. Сохранить лучшую модель.
2	1. Загрузить датасет MNIST. Нормализовать. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Эксперимент: обучить модель с разным количеством скрытых слоёв (1, 2, 3, 4) при фиксированном числе нейронов (256). 4. Для каждого варианта обучить на 10 эпох, зафиксировать accuracy на валидации. 5. Построить график зависимости accuracy от числа слоёв. 6. Для лучшей конфигурации обучить модель на 30 эпох с ReduceLROnPlateau. 7. Выбрать оптимальное число слоёв и обосновать. 8. Построить confusion matrix. 9. Сохранить лучшую модель. 10. Сделать вывод о влиянии глубины сети.
3	1. Загрузить датасет Fashion-MNIST. 2. Создать модель с архитектурой: Flatten → Dense(512, activation='relu') → Dense(512, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель с разными функциями активации в скрытых слоях: relu, tanh, sigmoid, leaky_relu (через LeakyReLU). 4. Для каждой активации обучить 10 эпох, сравнить accuracy и

	время сходимости. 5. Построить графики сравнения loss по эпохам. 6. Выбрать лучшую функцию активации. 7. Обучить модель с лучшей активацией на 30 эпох с EarlyStopping. 8. Оценить на тесте. 9. Визуализировать 10 неправильно классифицированных изображений. 10. Сохранить модель.
4	1. Загрузить датасет MNIST. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Эксперимент: обучить модель с разными значениями learning rate: 0.1, 0.01, 0.001, 0.0001 (оптимизатор SGD). 4. Для каждого lr обучить 15 эпох, построить график loss. 5. Выбрать оптимальный lr. 6. Сравнить с оптимизатором Adam (learning rate по умолчанию). 7. Построить график сравнения accuracy на валидации для SGD (лучший lr) и Adam. 8. Оценить лучшую модель на тесте. 9. Сохранить модель. 10. Сделать вывод о влиянии learning rate.
5	1. Загрузить датасет Fashion-MNIST. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Эксперимент: обучить модель с разными размерами батча: 16, 32, 64, 128, 256. 4. Для каждого батча замерить время одной эпохи и итоговую accuracy после 10 эпох. 5. Построить график зависимости времени и accuracy от batch size. 6. Выбрать оптимальный batch size. 7. Обучить модель с выбранным батчем на 30 эпох. 8. Сравнить с результатом при batch size=32. 9. Визуализировать кривые обучения. 10. Сохранить модель.
6	1. Загрузить датасет MNIST. 2. Построить модель: Flatten → Dense(512, activation='relu') → Dropout(0.5) → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Использовать RandomizedSearchCV с KerasClassifier для

	подбора dropout_rate (0.2, 0.3, 0.5, 0.7) и learning_rate (0.001, 0.01, 0.1). 4. Вывести лучшие параметры. 5. Обучить лучшую модель на 20 эпох. 6. Построить графики обучения. 7. Сравнить accuracy с моделью без Dropout. 8. Построить confusion matrix. 9. Сохранить модель. 10. Сделать вывод о влиянии dropout на переобучение.
7	1. Загрузить датасет Fashion-MNIST. 2. Построить модель: Flatten → Dense(128, activation='relu') → Dense(64, activation='relu') → Dense(10, activation='softmax'). 3. Использовать keras_tuner (RandomSearch) для подбора числа нейронов в первом слое (64–512), второго слоя (32–256) и learning rate (0.001, 0.01). 4. Вывести лучшую архитектуру. 5. Обучить лучшую модель на 30 эпох. 6. Построить графики обучения. 7. Оценить на тесте. 8. Сравнить с базовой моделью (128→64). 9. Визуализировать матрицу ошибок. 10. Сохранить лучшую модель.
8	1. Загрузить датасет MNIST. 2. Построить модель с архитектурой: Flatten → Dense(256, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Эксперимент: обучить модель с различными инициализаторами весов: glorot_uniform, he_normal, random_normal. 4. Для каждого инициализатора обучить 10 эпох, сравнить скорость сходимости. 5. Построить графики loss. 6. Выбрать лучший инициализатор. 7. Обучить модель с лучшим инициализатором на 30 эпох. 8. Оценить на тесте. 9. Сохранить модель. 10. Сделать вывод о влиянии инициализации.
9	1. Загрузить датасет Fashion-MNIST. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Использовать GridSearchCV для подбора оптимизатора ('adam',

	'rmsprop', 'sgd') и функции активации ('relu', 'tanh'). 4. Вывести лучшую комбинацию. 5. Обучить лучшую модель на 25 эпох. 6. Построить графики обучения. 7. Сравнить с базовой (adam + relu). 8. Построить confusion matrix. 9. Сохранить модель. 10. Сделать вывод о выборе оптимизатора и активации.
10	1. Загрузить датасет MNIST. 2. Создать модель с архитектурой: Flatten → Dense(128, activation='relu') → Dense(64, activation='relu') → Dense(10, activation='softmax'). 3. Эксперимент: обучить модель с различными коэффициентами L2-регуляризации (kernel_regularizer) для всех слоёв: 0, 0.0001, 0.001, 0.01. 4. Для каждого значения обучить 15 эпох, сравнить accuracy на валидации и на обучении. 5. Построить график зависимости разницы train-val accuracy от коэффициента регуляризации. 6. Выбрать оптимальный коэффициент. 7. Обучить модель с L2-регуляризацией на 30 эпох. 8. Сравнить с моделью без регуляризации. 9. Сохранить модель. 10. Сделать вывод о влиянии L2-регуляризации на переобучение.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Какие гиперпараметры нейронной сети влияют на её способность к обучению и обобщению?
2. Что такое скорость обучения (learning rate) и как она влияет на сходимость? Какие проблемы возникают при слишком большом или слишком малом значении?
3. Как размер батча (batch size) влияет на скорость обучения и качество модели?
4. Какие функции активации обычно используются в скрытых слоях? В чём преимущество ReLU перед sigmoid/tanh?
5. Что такое регуляризация L1 и L2 в нейронных сетях? Как она помогает бороться с переобучением?
6. Как влияет количество скрытых слоёв и число нейронов на способность сети моделировать сложные зависимости и на риск переобучения?
7. Какие методы подбора гиперпараметров существуют? В чём разница между GridSearchCV, RandomizedSearchCV и keras-tuner?
8. Как обернуть модель Keras для использования с GridSearchCV из sklearn? Что для этого нужно?
9. Что такое инициализация весов? Как выбор метода инициализации влияет на обучение глубоких сетей?
10. Как сравнить эффективность различных конфигураций гиперпараметров без переобучения на тестовой выборке? Какую роль играет кросс-валидация?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность использования методов подбора гиперпараметров. Применение GridSearchCV / RandomizedSearchCV / keras-tuner, обоснование выбора.	10
Анализ влияния гиперпараметров. Интерпретация графиков, сравнение конфигураций, выводы о чувствительности модели.	10
Ответы на контрольные вопросы (10 вопросов).	10

Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Обоснование выбора оптимальной конфигурации, сравнение с базовой моделью.	10
Визуализации (графики loss/accuracy для разных параметров). Качество, читаемость, подписи.	10

Лабораторная работа № 24

Регуляризация нейронных сетей: Dropout и Batch Normalization

Цель: Применить методы регуляризации для борьбы с переобучением.

Теоретическое обоснование

Переобучение в нейронных сетях проявляется в виде растущего разрыва между train и val loss/accuracy. Методы регуляризации уменьшают этот разрыв.

Dropout. Во время обучения случайным образом обнуляет (выключает) долю нейронов с вероятностью rate. Это предотвращает кооперативную адаптацию нейронов, заставляя сеть учиться более робастным признакам. На этапе тестирования dropout отключается, а веса умножаются на (1 - rate) для компенсации.

AlphaDropout — самомасштабируемый dropout, сохраняющий среднее и дисперсию активаций. Рекомендуется с активацией selu.

Batch Normalization. Нормализует активации внутри батча: вычитает среднее и делит на стандартное отклонение по батчу, затем масштабирует и сдвигает с помощью обучаемых параметров γ и β. Преимущества:

- Ускоряет сходимость (позволяет использовать более высокий learning rate).
- Снижает зависимость от инициализации.
- Оказывает регуляризующее действие (шум от статистики батча).
- Позволяет убрать Dropout или уменьшить его.

Порядок слоёв: обычно BatchNormalization ставят **перед** функцией активации: Dense() → BatchNormalization() → Activation('relu') → Dropout(). Однако в некоторых реализациях допускается после активации.

Сравнение эффектов: Dropout в основном борется с переобучением,

BatchNormalisation ускоряет обучение и также немного регуляризует. Лучшие результаты часто даёт комбинация обоих методов.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет Fashion-MNIST. Нормализовать пиксели. 2. Построить «переобучающуюся» модель с архитектурой: Flatten → Dense(1024, activation='relu') → Dense(512, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель на 30 эпох (без регуляризации), построить графики accuracy на обучении и валидации – показать переобучение. 4. Добавить Dropout(0.5) после каждого скрытого слоя. Обучить заново, сравнить графики. 5. Добавить BatchNormalization после каждого скрытого слоя (до активации или после – по желанию) вместе с Dropout. 6. Сравнить три модели: без регуляризации, только Dropout, Dropout + BatchNormalization. 7. Для лучшей модели вычислить accuracy на тесте, построить confusion matrix. 8. Эксперимент: изменить dropout_rate на 0.2 и 0.7 – сравнить влияние. 9. Сохранить лучшую модель. 10. Сделать вывод о вкладе каждого метода регуляризации.
2	1. Загрузить датасет MNIST. Нормализовать. 2. Создать глубокую модель: Flatten → Dense(512, activation='relu') → Dense(512, activation='relu') → Dense(512, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель на 20 эпох, зафиксировать разницу между train и val accuracy (переобучение). 4. Добавить BatchNormalization после каждого скрытого слоя (перед активацией). Обучить – сравнить скорость сходимости. 5. Добавить Dropout(0.3) после каждого BatchNormalization. 6. Построить графики accuracy и loss для всех трёх вариантов на одном поле. 7. Оценить модели на тесте (accuracy). 8. Выбрать лучшую модель, построить confusion matrix.

	9. Сохранить модель в формате .h5. 10. Сделать вывод о влиянии BatchNormalization на скорость обучения.
3	1. Загрузить датасет Fashion-MNIST. 2. Построить модель: Flatten → Dense(1024, activation='relu') → Dropout(0.5) → Dense(512, activation='relu') → Dropout(0.5) → Dense(10, activation='softmax'). 3. Обучить модель с разными значениями dropout_rate в первом и втором слоях (0.2/0.5, 0.5/0.5, 0.5/0.8, 0.8/0.8). 4. Для каждой комбинации обучить 15 эпох, сравнить accuracy на валидации. 5. Построить тепловую карту (heatmap) зависимости accuracy от dropout_rate. 6. Выбрать оптимальную комбинацию. 7. Добавить BatchNormalization перед Dropout, сравнить качество. 8. Построить confusion matrix для лучшей модели. 9. Сохранить модель. 10. Сделать вывод о том, как распределение dropout по слоям влияет на регуляризацию.
4	1. Загрузить датасет CIFAR-10 (через keras.datasets.cifar10.load_data()). Нормализовать пиксели. 2. Построить модель: Flatten → Dense(2048, activation='relu') → Dense(2048, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель без регуляризации (20 эпох) – наблюдать сильное переобучение. 4. Добавить Dropout(0.5) после каждого скрытого слоя – сравнить. 5. Добавить BatchNormalization после каждого скрытого слоя (перед активацией) – сравнить. 6. Добавить комбинацию BatchNormalization + Dropout. 7. Для каждой конфигурации построить графики val_accuracy. 8. Выбрать лучшую конфигурацию, оценить на тесте. 9. Визуализировать несколько неправильно классифицированных изображений. 10. Сохранить лучшую модель.
5	1. Загрузить датасет MNIST. 2. Построить модель с архитектурой: Flatten → Dense(1024, activation='relu') → Dense(1024, activation='relu') → Dense(10, activation='softmax').

	3. Обучить модель с EarlyStopping (patience=5) без регуляризации – определить эпоху остановки. 4. Добавить BatchNormalization после каждого скрытого слоя. Сравнить эпоху остановки и итоговую accuracy. 5. Добавить Dropout(0.3) после BatchNormalization – сравнить. 6. Построить графики val_loss для всех трёх случаев. 7. Оценить модели на тесте. 8. Построить confusion matrix для лучшей модели. 9. Сохранить модель. 10. Сделать вывод о влиянии регуляризации на сходимость и раннюю остановку.
6	1. Загрузить датасет Fashion-MNIST. 2. Создать модель с высоким риском переобучения: Flatten → Dense(1024, activation='relu') → Dense(1024, activation='relu') → Dense(1024, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель без регуляризации (30 эпох) – показать разрыв train/val. 4. Добавить SpatialDropout1D (для Flatten – не применим) – использовать обычный Dropout. 5. Добавить BatchNormalization и Dropout вместе. 6. Эксперимент: изменить позицию BatchNormalization (до активации и после активации) – сравнить качество. 7. Построить графики сравнения. 8. Для лучшей конфигурации вычислить accuracy на тесте. 9. Сохранить модель. 10. Сделать вывод о влиянии порядка слоёв (BN до или после активации).
7	1. Загрузить датасет MNIST. 2. Построить модель: Flatten → Dense(512, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель с разными вероятностями Dropout только на последнем скрытом слое: 0, 0.2, 0.5, 0.7. 4. Сравнить accuracy на валидации и разницу train-val accuracy. 5. Построить график зависимости степени переобучения от dropout_rate. 6. Выбрать оптимальный dropout_rate для последнего слоя. 7. Добавить BatchNormalization перед этим слоем – сравнить. 8. Оценить на тесте. 9. Сохранить модель. 10. Сделать вывод о том, какие слои важнее регуляризовать.
8	1. Загрузить датасет Fashion-MNIST.

	2. Построить модель с архитектурой: Flatten → Dense(1024, activation='relu') → BatchNormalization() → Dropout(0.5) → Dense(512, activation='relu') → BatchNormalization() → Dropout(0.5) → Dense(10, activation='softmax'). 3. Обучить модель на 40 эпох. 4. Сравнить с моделью без BatchNormalization (только Dropout). 5. Сравнить с моделью без Dropout (только BatchNormalization). 6. Построить все графики accuracy на одном поле. 7. Оценить на тесте. 8. Визуализировать 10 примеров, где модель сомневается (вероятность предсказания < 0.7). 9. Сохранить модель. 10. Сделать вывод о синергии Dropout и BatchNormalization.
9	1. Загрузить датасет CIFAR-10. Нормализовать. 2. Построить модель с несколькими скрытыми слоями (2048, 1024, 512). 3. Обучить модель с L2-регуляризацией (kernel_regularizer=l2(0.001)). 4. Обучить модель с Dropout(0.5). 5. Обучить модель с BatchNormalization. 6. Обучить модель со всеми тремя методами вместе. 7. Сравнить accuracy на валидации и тесте. 8. Построить столбчатую диаграмму сравнения. 9. Сохранить лучшую модель. 10. Сделать вывод о том, какой метод регуляризации наиболее эффективен для CIFAR-10.
10	1. Загрузить датасет MNIST. 2. Построить «переобучающуюся» модель: Flatten → Dense(1024, activation='relu') → Dense(1024, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель без регуляризации, сохранить историю. 4. Добавить Dropout(0.5) после каждого скрытого слоя. 5. Добавить BatchNormalization после каждого скрытого слоя (до активации) + Dropout(0.5). 6. Сравнить кривые обучения (train/val accuracy). 7. Эксперимент: добавить AlphaDropout (самомасштабируемый) вместо обычного Dropout – сравнить. 8. Выбрать лучшую модель и оценить на тесте. 9. Построить матрицу ошибок. 10. Сохранить модель и сделать вывод.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое переобучение (overfitting) в нейронных сетях? Как его распознать по кривым обучения?
2. Как работает Dropout? Почему он помогает предотвратить переобучение?
3. Какие параметры имеет Dropout слой? Что означает вероятность rate?
4. Как работает BatchNormalization? Какие параметры она нормализует?
5. Почему BatchNormalization позволяет использовать более высокие скорости обучения?
6. В каком порядке обычно располагают BatchNormalization, активацию и Dropout? Есть ли рекомендации?
7. Чем отличается AlphaDropout от обычного Dropout? Когда его используют?
8. Как регуляризация влияет на скорость сходимости и итоговое качество модели?
9. В каких случаях Dropout может ухудшить качество модели?
10. Можно ли использовать BatchNormalization и Dropout вместе? Если да, то в каком порядке?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40

Демонстрация переобучения. Наличие модели без регуляризации с явным разрывом train/val accuracy.	10
Корректное применение Dropout и BatchNormalization. Правильное добавление слоёв, обоснование выбора параметров.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ влияния регуляризации, сравнение методов, обоснование лучшей конфигурации.	10
Визуализации (сравнительные графики accuracy/loss). Качество, читаемость, наличие легенд.	10

Лабораторная работа № 25

Ранняя остановка и сохранение моделей

Цель: Научиться использовать early stopping и сохранять обученные модели.

Теоретическое обоснование

EarlyStopping — callback, останавливающий обучение, когда перестаёт улучшаться заданная метрика на валидации. Параметры:

- monitor — метрика для отслеживания (например, 'val_loss', 'val_accuracy').
- mode — 'min' для loss, 'max' для accuracy.
- patience — число эпох без улучшения после которого остановка.
- restore_best_weights — восстанавливать веса из лучшей эпохи (иначе останутся веса последней эпохи, которые могут быть хуже).
- min_delta — минимальное изменение, считающееся улучшением.

ModelCheckpoint — callback, сохраняющий модель в файл:

- filepath — путь (можно включать форматирование: model_{epoch:02d}_{val_accuracy:.2f}.h5).
- monitor, mode — аналогично EarlyStopping.

- save_best_only — сохранять только лучшую модель.
- save_weights_only — сохранять только веса (экономит место).
- save_freq — частота сохранения ('epoch' или число шагов).

Форматы сохранения:

- **HDF5 (.h5)** — единый файл, содержащий архитектуру, веса, оптимизатор. Загрузка: tf.keras.models.load_model().
- **SavedModel (.pb)** — папка с несколькими файлами, стандарт TensorFlow для сервинга. Загрузка: tf.keras.models.load_model('saved_model_dir').
- **Только веса (.weights.h5)** — model.save_weights() / model.load_weights().

Практический паттерн: использовать EarlyStopping вместе с ModelCheckpoint (сохраняющим лучшую модель). После обучения загрузить лучшую модель для оценки.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет Fashion-MNIST. Нормализовать пиксели. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Скомпилировать модель с оптимизатором adam и sparse_categorical_crossentropy. 4. Обучить модель на 100 эпох без EarlyStopping, построить график val_accuracy – показать ухудшение после определённого момента. 5. Добавить callback EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True). Обучить заново. 6. Вывести эпоху, на которой остановилось обучение. 7. Сравнить accuracy на тесте с моделью, обученной 100 эпох без ранней остановки. 8. Построить графики val_loss для обеих моделей на одном поле. 9. Сохранить модель с EarlyStopping в формате .h5. 10. Сделать вывод о преимуществах ранней остановки.
2	1. Загрузить датасет MNIST. Нормализовать. 2. Построить модель: Flatten → Dense(512, activation='relu') → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Использовать ModelCheckpoint(filepath='best_model.h5',

	monitor='val_accuracy', save_best_only=True). 4. Обучить модель на 50 эпох с валидацией (20%). 5. Загрузить сохранённую лучшую модель и оценить на тесте. 6. Сравнить accuracy лучшей модели с моделью после последней эпохи. 7. Добавить EarlyStopping(patience=10, monitor='val_loss') вместе с ModelCheckpoint. 8. Обучить заново, проверить, что сохраняется лучшая модель. 9. Построить график val_accuracy с отметкой эпохи остановки. 10. Сохранить историю обучения в CSV.
3	1. Загрузить датасет CIFAR-10. Нормализовать. 2. Построить модель: Flatten → Dense(1024, activation='relu') → Dense(512, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель с EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=False). 4. Обучить модель с restore_best_weights=True. Сравнить accuracy на тесте. 5. Объяснить разницу в результатах. 6. Использовать ModelCheckpoint с save_weights_only=True – сохранить только веса. 7. Создать новую модель с той же архитектурой, загрузить веса, проверить предсказания. 8. Построить confusion matrix для лучшей модели. 9. Сохранить модель в формате SavedModel (.pb). 10. Сделать вывод о важности restore_best_weights.
4	1. Загрузить датасет Fashion-MNIST. 2. Построить модель с тремя скрытыми слоями (512, 256, 128). 3. Использовать EarlyStopping(monitor='val_accuracy', mode='max', patience=5, restore_best_weights=True). 4. Обучить модель на 80 эпох. Зафиксировать эпоху остановки. 5. Построить график val_accuracy с вертикальной линией на эпохе остановки. 6. Эксперимент: изменить patience на 2, 5, 10 – для каждого обучить модель и сравнить итоговую accuracy. 7. Построить график зависимости accuracy от patience. 8. Выбрать оптимальное patience. 9. Сохранить модель с оптимальным patience. 10. Сделать вывод о выборе patience.
5	1. Загрузить датасет MNIST. 2. Построить модель: Flatten → Dense(1024, activation='relu') → Dropout(0.5) → Dense(10, activation='softmax').

	3. Обучить модель с ModelCheckpoint и EarlyStopping вместе. 4. В callback ModelCheckpoint указать save_freq='epoch'. 5. После обучения загрузить лучшую модель. 6. Сравнить accuracy на тесте с моделью, обученной без регуляризации. 7. Визуализировать сохранённые файлы модели (размер). 8. Эксперимент: изменить save_best_only=False – сколько файлов создалось? 9. Сохранить модель в формате .keras. 10. Сделать вывод о влиянии регуляризации на необходимость ранней остановки.
6	1. Загрузить датасет CIFAR-10. 2. Построить глубокую модель: Flatten → Dense(2048, activation='relu') → Dense(2048, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель на 100 эпох без ранней остановки, записывая лучшую модель через ModelCheckpoint. 4. Загрузить лучшую модель и оценить на тесте. 5. Обучить модель с EarlyStopping(patience=10). Сравнить время обучения. 6. Построить график val_loss для обеих стратегий. 7. Показать, сколько эпох сэкономила ранняя остановка. 8. Сохранить модель с ранней остановкой. 9. Загрузить модель и использовать для предсказания 5 случайных изображений. 10. Сделать вывод об эффективности ранней остановки во времени.
7	1. Загрузить датасет Fashion-MNIST. 2. Построить модель с архитектурой: Flatten → Dense(512, activation='relu') → BatchNormalization() → Dense(256, activation='relu') → Dense(10, activation='softmax'). 3. Использовать EarlyStopping(monitor='val_loss', patience=7, min_delta=0.001). 4. Обучить модель. 5. Эксперимент: изменить min_delta на 0.0001, 0.001, 0.01 – сравнить эпоху остановки. 6. Построить график зависимости эпохи остановки от min_delta. 7. Выбрать оптимальный min_delta. 8. Сохранить лучшую модель. 9. Оценить на тесте. 10. Сделать вывод о влиянии min_delta на чувствительность

	ранней остановки.
8	1. Загрузить датасет MNIST. 2. Построить модель: Flatten → Dense(256, activation='relu') → Dense(128, activation='relu') → Dense(10, activation='softmax'). 3. Обучить модель на 30 эпох, сохраняя модель после каждой эпохи (через ModelCheckpoint с именем model_epoch_{epoch:02d}.h5). 4. Загрузить модель с 10-й эпохи и модель с 30-й эпохи, сравнить accuracy на тесте. 5. Использовать EarlyStopping с patience=3 и обучить заново – определить лучшую эпоху. 6. Загрузить модель с этой эпохи из сохранённых. 7. Сравнить accuracy трёх подходов (эпоха 10, эпоха 30, ранняя остановка). 8. Построить график val_accuracy с отметкой лучшей эпохи. 9. Сохранить лучшую модель. 10. Сделать вывод о том, почему ранняя остановка может найти лучшую эпоху, чем фиксированное число.
9	1. Загрузить датасет Fashion-MNIST. 2. Построить модель с несколькими скрытыми слоями (1024, 512, 256). 3. Использовать комбинацию ReduceLROnPlateau (терпение 3, фактор 0.5) и EarlyStopping (терпение 10). 4. Обучить модель, построить графики lr и val_loss. 5. Сохранить модель через ModelCheckpoint с периодом 5 эпох (save_freq=5). 6. Загрузить последнюю сохранённую модель и оценить. 7. Сравнить с моделью, сохранённой save_best_only=True. 8. Эксперимент: сохранять модель только при улучшении val_accuracy. 9. Сохранить историю обучения в DataFrame. 10. Сделать вывод о комбинировании callback'ов.
10	1. Загрузить датасет CIFAR-10. 2. Построить модель: Flatten → Dense(1024, activation='relu') → Dense(512, activation='relu') → Dense(10, activation='softmax'). 3. Создать пользовательский callback для логирования времени каждой эпохи. 4. Обучить модель с EarlyStopping(patience=5) и ModelCheckpoint. 5. После обучения вывести: общее время, лучшую эпоху, лучшую val_accuracy.

	6. Загрузить лучшую модель и оценить на тесте. 7. Сохранить модель в двух форматах: .h5 и SavedModel. Сравнить размеры файлов. 8. Загрузить модель из SavedModel и убедиться, что предсказания совпадают. 9. Построить confusion matrix. 10. Сделать вывод о том, какой формат сохранения удобнее для развёртывания.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое ранняя остановка (Early Stopping) и для чего она используется?
2. Какие параметры callback EarlyStopping отвечают за выбор метрики, терпение и восстановление лучших весов?
3. В чём разница между restore_best_weights=True и False? Почему важно устанавливать True?
4. Какой callback используется для сохранения модели во время обучения? Какие параметры позволяют сохранять только лучшую модель?
5. В каких форматах можно сохранять модели Keras/TensorFlow? В чём разница между .h5 и SavedModel?
6. Как загрузить сохранённую модель из файла .h5? Как загрузить из SavedModel?

7. Как сохранить только веса модели (без архитектуры)? Как потом их загрузить в новую модель?
8. Как работает `ModelCheckpoint` с `save_freq='epoch'` и `save_freq=N` (например, каждые 5 эпох)?
9. Как можно комбинировать `EarlyStopping` и `ModelCheckpoint` для достижения наилучшего результата?
10. Почему ранняя остановка считается методом регуляризации? Как она предотвращает переобучение?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность использования EarlyStopping. Корректные параметры (monitor, patience, restore_best_weights), анализ остановки.	10
Правильность сохранения и загрузки моделей. Умение сохранять в разных форматах, загружать и восстанавливать предсказания.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Сравнение моделей с early stopping и без, обоснование выбора patience, анализ графиков.	10
Визуализации (графики с отметкой эпохи остановки). Качество, читаемость, подписи.	10

Лабораторная работа № 26

Функциональный API Keras. Модели с несколькими входами/выходами

Цель: Освоить функциональный API для создания более сложных архитектур.

Теоретическое обоснование

Sequential API подходит только для линейного стека слоёв (один вход, один выход). **Functional API** позволяет строить произвольные графы:

- Несколько входов.
- Несколько выходов.
- Общие слои (shared layers).
- Нелинейные топологии (остаточные связи, Inception и т.д.).

Синтаксис:

python

```
input1 = Input(shape=(dim1,))
x1 = Dense(32, activation='relu')(input1)
input2 = Input(shape=(dim2,))
x2 = Dense(32, activation='relu')(input2)
concatenated = Concatenate()([x1, x2])
output = Dense(num_classes, activation='softmax')(concatenated)
model = Model(inputs=[input1, input2], outputs=output)
```

Модели с несколькими выходами (многозадачное обучение):

python

```
output_class = Dense(3, activation='softmax', name='class')(x)
output_reg = Dense(1, activation='linear', name='reg')(x)
model = Model(inputs=input_, outputs=[output_class, output_reg])
model.compile(
    loss={'class': 'categorical_crossentropy', 'reg': 'mse'},
    loss_weights={'class': 0.5, 'reg': 0.5}
)
```

Общие слои (shared layers). Один и тот же слой может быть вызван для разных входов; его веса будут общими. Полезно для обработки похожих данных

(например, два изображения одного объекта).

Сценарии использования:

- Разные типы признаков (текст + категориальные + числовые) — разные ветви обработки.
- Многозадачность (одновременное предсказание нескольких связанных величин).
- Siamese сети (сравнение двух объектов).
- Рекуррентные модели с состоянием.

Визуализация: tf.keras.utils.plot_model(model, show_shapes=True, show_layer_names=True).

Functional API является стандартом для сложных архитектур глубокого обучения, включая ResNet, Inception, трансформеры.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет iris. Разделить признаки на две группы: первые два признака (sepal_length, sepal_width) и последние два (petal_length, petal_width). 2. Создать функциональную модель с двумя входами: Input(shape=(2,)) для первой группы и Input(shape=(2,)) для второй группы. 3. Каждый вход пропустить через свой скрытый слой (Dense 16, activation='relu'), затем объединить через Concatenate(). 4. Добавить выходной слой (Dense 3, activation='softmax'). Скомпилировать модель. 5. Обучить модель на 50 эпохах. 6. Построить графики accuracy и loss. 7. Сравнить accuracy с Sequential-моделью, использующей все 4 признака. 8. Визуализировать архитектуру модели с помощью plot_model. 9. Сохранить модель в формате .h5. 10. Сделать вывод о преимуществах разделения входов
2	1. Загрузить датасет wine. Разделить на числовые и категориальные признаки (создать искусственный категориальный признак по квантилю alcohol).

	2. Построить функциональную модель: один вход для числовых признаков (Dense 32), второй вход для категориального (Embedding + Flatten). 3. Объединить выходы через Concatenate, добавить Dense(16, 'relu'), затем выходной слой (Dense 3, 'softmax'). 4. Скомпилировать и обучить модель. 5. Сравнить с моделью, где категориальный признак закодирован one-hot и подан в тот же вход. 6. Построить confusion matrix. 7. Сохранить модель. 8. Загрузить модель и сделать предсказание для нового образца. 9. Визуализировать архитектуру. 10. Сделать вывод о гибкости функционального API.
3	1. Загрузить датасет titanic после предобработки. Создать два набора признаков: числовые (age, fare, sibsp, parch) и категориальные (pclass, sex, embarked – one-hot). 2. Построить функциональную модель: вход числовых → Dense(32, 'relu'); вход категориальных → Dense(32, 'relu'); объединить. 3. Добавить Dropout(0.3) после объединения, затем выход (Dense 1, 'sigmoid'). 4. Обучить модель на 50 эпох с EarlyStopping. 5. Построить ROC-кривую, вычислить AUC-ROC. 6. Сравнить с Sequential-моделью, где все признаки подаются одним входом. 7. Визуализировать графики обучения. 8. Сохранить лучшую модель. 9. Загрузить модель и проверить на тесте. 10. Сделать вывод о целесообразности разделения входов.
4	1. Загрузить датасет mnist (только 5000 образцов). Создать два входа: первый – полное изображение (784), второй – только центральная часть (14×14 = 196). 2. Построить модель: первый вход → Dense(128, 'relu'); второй вход → Dense(64, 'relu'); объединить → Dense(10, 'softmax'). 3. Обучить модель на 20 эпох. 4. Сравнить с моделью, использующей только полное изображение. 5. Эксперимент: изменить размер центральной части (7×7, 14×14, 21×21) – сравнить accuracy. 6. Построить график зависимости accuracy от размера второго входа.

	7. Визуализировать архитектуру. 8. Сохранить модель. 9. Сделать предсказание для одного изображения, используя оба входа. 10. Сделать вывод о том, как дополнительная информация (центр) влияет на качество.
5	1. Создать синтетический датасет для многозадачного обучения: признаки X (10 признаков), целевые переменные – классификация (3 класса) и регрессия (одно значение). 2. Построить функциональную модель с одним входом и двумя выходами: o Выход классификации: Dense(3, 'softmax', name='class') o Выход регрессии: Dense(1, 'linear', name='reg') 3. Скомпилировать модель с двумя функциями потерь (categorical_crossentropy и mse) и разными весами (loss_weights=[0.5, 0.5]). 4. Обучить модель на 50 эпох. 5. Построить графики обеих функций потерь. 6. Сделать предсказание для тестового образца – получить оба выхода. 7. Сравнить с двумя отдельными моделями (одна для классификации, одна для регрессии). 8. Сохранить модель. 9. Загрузить модель и проверить. 10. Сделать вывод о преимуществах многозадачного обучения.
6	1. Загрузить датасет california_housing. Создать два набора признаков: социально-экономические (MedInc, Population, HouseAge) и географические (Latitude, Longitude). 2. Построить функциональную модель с двумя входами: o Социально-экономические → Dense(32, 'relu') → Dense(16, 'relu') o Географические → Dense(16, 'relu') 3. Объединить → Dense(1, 'linear') для предсказания цены. 4. Обучить модель на 100 эпох. 5. Вычислить RMSE и R^2 на тесте. 6. Сравнить с моделью, где все признаки подаются одним входом. 7. Построить график предсказанных vs истинных значений. 8. Сохранить модель. 9. Эксперимент: добавить BatchNormalization после каждого слоя

	– сравнить качество. 10. Сделать вывод о пользе разделения признаков по смыслу.
7	1. Загрузить датасет fashion_mnist (2000 образцов). Создать два входа: исходное изображение (784) и его версия с шумом (добавить гауссов шум). 2. Построить модель с общими слоями (shared layers) для двух входов: o Вход1 → Dense(128, 'relu') o Вход2 → Dense(128, 'relu') – те же веса (shared) через использование одного и того же слоя. 3. Объединить через Concatenate или Add, затем выходной слой. 4. Обучить модель. 5. Сравнить с моделью без шума (один вход). 6. Построить confusion matrix. 7. Визуализировать архитектуру с shared слоями. 8. Сохранить модель. 9. Эксперимент: изменить уровень шума – как меняется accuracy? 10. Сделать вывод о робастности модели к шуму.
8	1. Загрузить датасет digits. Создать модель с двумя выходами: o Классификация цифры (10 классов) o Классификация чётности (чётная/нечётная) 2. Построить функциональную модель с одним входом (64 признака) и двумя выходами через общую базу (Dense(64, 'relu'), затем два отдельных Dense). 3. Скомпилировать модель с двумя функциями потерь (sparse_categorical_crossentropy для цифры, binary_crossentropy для чётности). 4. Обучить на 30 эпох. 5. Построить графики точности для каждой задачи. 6. Сравнить с двумя отдельными моделями (время обучения и качество). 7. Визуализировать архитектуру. 8. Сохранить модель. 9. Сделать предсказание и вывести оба выхода. 10. Сделать вывод о том, помогает ли многозадачность.
9	1. Создать синтетические данные: текстовые описания (эмбеддинги) и числовые признаки. Целевая переменная – бинарная. 2. Построить модель с двумя входами: o Текстовый вход (50 эмбеддингов) → LSTM(32)

	○ Числовой вход (5 признаков) → Dense(16, 'relu') 3. Объединить → Dense(1, 'sigmoid'). 4. Обучить модель на 30 эпох. 5. Построить ROC-кривую, вычислить AUC-ROC. 6. Сравнить с моделью, использующей только числовые признаки. 7. Визуализировать архитектуру. 8. Сохранить модель. 9. Загрузить и проверить. 10. Сделать вывод о пользе комбинирования разнородных данных.
10	1. Загрузить датасет boston_housing. Создать модель с двумя выходами: ○ Предсказание цены (регрессия) ○ Предсказание категории цены (низкая/средняя/высокая) – по терцилям. 2. Построить функциональную модель: вход → Dense(64, 'relu') → Dense(32, 'relu') → два выходных слоя (линейный и softmax). 3. Скомпилировать с loss_weights=[0.7, 0.3]. 4. Обучить на 100 эпох. 5. Вычислить MAE для регрессии и accuracy для классификации. 6. Сравнить с отдельными моделями. 7. Построить графики потерь. 8. Сохранить модель. 9. Загрузить и сделать предсказание. 10. Сделать вывод о том, как многозадачность влияет на качество каждой задачи.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. В чём основное различие между Sequential API и Functional API в Keras? Когда следует использовать Functional API?
2. Как создать модель с несколькими входами в Functional API? Приведите пример кода.
3. Как создать модель с несколькими выходами? Как задать разные функции потерь и веса для каждого выхода?
4. Что такое общие слои (shared layers) в Functional API? Как они экономят память и как реализуются?
5. Как объединить выходы нескольких слоёв? Какие слои для этого используются (Concatenate, Add, Multiply)?
6. Как визуализировать архитектуру функциональной модели с помощью plot_model?
7. В каких задачах полезно использовать модели с несколькими входами (приведите 2–3 примера)?
8. В каких задачах полезно использовать модели с несколькими выходами (многозадачное обучение)?
9. Как сохранить и загрузить функциональную модель? Есть ли отличия от Sequential?
10. Можно ли в функциональной модели создавать циклы и рекуррентные связи? Если да, для чего?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность построения функциональной модели. Корректное создание графа, использование Input, правильное объединение слоёв.	10
Работа с несколькими входами/выходами. Умение задать несколько входов, выходов, разные функции потерь и веса.	10
Ответы на контрольные вопросы (10 вопросов).	10

Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Сравнение с Sequential, обоснование выбора архитектуры, анализ многозадачности.	10
Визуализации (plot_model, графики обучения). Качество, читаемость, подписи.	10

Лабораторная работа № 27

Сверточные нейронные сети (CNN) для классификации изображений

Цель: Реализовать сверточную сеть для распознавания изображений.

Теоретическое обоснование

Сверточные нейронные сети (Convolutional Neural Networks, CNN) являются основным инструментом для анализа визуальных данных. В отличие от полносвязных (плотных) сетей, которые обрабатывают изображение как плоский вектор и теряют пространственную структуру, CNN сохраняют двумерную топологию пикселей и используют три ключевых принципа: **локальные рецептивные поля**, **разделяемые веса** (сверточные ядра) и **пространственное субдискретизирование** (пулинг).

Сверточный слой (Conv2D) — основной строительный блок. Он применяет набор обучаемых фильтров (ядер) к входному изображению. Каждый фильтр — это небольшое окно (обычно 3×3 или 5×5), которое скользит по всему изображению, вычисляя скалярное произведение своих весов с соответствующим фрагментом изображения. Результат — карта признаков (feature map), где каждый элемент показывает активацию фильтра в данной позиции.

Параметры сверточного слоя:

- filters — количество фильтров (глубина выходной карты признаков). Каждый фильтр обнаруживает определённый паттерн (края, текстуры, формы).
- kernel_size — размер окна фильтра (например, (3,3) или (5,5)).
- strides — шаг скольжения фильтра. По умолчанию (1,1). Больший шаг уменьшает пространственные размеры.
- padding — обработка границ. 'valid' — не добавляет отступы, выходное изображение уменьшается. 'same' — добавляет нулевые отступы так, чтобы выходной размер совпадал с входным (с учётом шага).

Функция активации — после свертки обычно применяется ReLU (Rectified Linear Unit): $f(x)=\max(0,x)$. Она вносит нелинейность и ускоряет обучение, решая проблему исчезающего градиента.

Пулинговый слой (MaxPooling2D) уменьшает пространственные размеры карт признаков, сохраняя наиболее сильные активации. MaxPooling берёт окно (обычно 2×2) и оставляет максимальное значение, что делает представление

инвариантным к малым сдвигам и уменьшает количество параметров.

Архитектура CNN обычно следует схеме: несколько блоков [Conv2D → ReLU → MaxPooling2D], затем слой Flatten (превращает многомерные карты в вектор) и один или несколько полносвязных слоев (Dense) с функцией активации softmax для классификации.

Преимущества CNN перед полносвязной сетью:

- Значительно меньшее количество параметров (благодаря разделяемым весам).
- Способность обнаруживать пространственные иерархии признаков (от простых краёв к сложным формам).
- Инвариантность к трансляциям и локальным деформациям.

Визуализация фильтров и карт признаков. Фильтры первых слоев визуализируются как небольшие цветные или градационные паттерны — они реагируют на простые структуры (горизонтальные/вертикальные границы, цветовые пятна). Карты признаков промежуточных слоев показывают, какие части изображения активируют определённые фильтры. Это помогает понять, что именно «видит» сеть.

Датасет CIFAR-10 — 60 000 цветных изображений 32×32 пикселя, разделённых на 10 классов (самолёты, автомобили, птицы, кошки и т.д.). 50 000 тренировочных, 10 000 тестовых. CIFAR-10 сложнее MNIST из-за цвета, низкого разрешения и внутриклассовой вариативности.

Сравнение с полносвязной сетью: полносвязная сеть, обрабатывающая 32×32×3 = 3072 пикселя, имеет миллионы параметров, что приводит к сильному переобучению. CNN, напротив, эффективно обобщает и достигает точности 70–80% на CIFAR-10 даже с простой архитектурой.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет CIFAR-10 через keras.datasets.cifar10.load_data(). 2. Нормализовать пиксели (разделить на 255.0). Преобразовать метки в one-hot encoding. 3. Построить модель Sequential со следующей архитектурой:

	○ Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(32,32,3)) ○ MaxPooling2D((2,2)) ○ Conv2D(64, (3,3), activation='relu') ○ MaxPooling2D((2,2)) ○ Flatten() ○ Dense(128, activation='relu') ○ Dense(10, activation='softmax') 4. Скомпилировать модель с оптимизатором adam, функцией потерь categorical_crossentropy и метрикой accuracy. 5. Обучить модель на 10 эпохах с размером батча 64, используя 20% обучающей выборки как валидационную. 6. Построить графики точности и функции потерь на обучении и валидации по эпохам. 7. Оценить модель на тестовой выборке: вывести accuracy и loss. 8. Построить confusion matrix в виде тепловой карты для тестовой выборки. 9. Визуализировать первые 16 фильтров первого сверточного слоя (веса модели) с помощью matplotlib. 10. Сохранить модель в формате .h5 и загрузить обратно, проверив предсказания на первых 5 тестовых изображениях.
2	1. Загрузить CIFAR-10, нормализовать, преобразовать метки в one-hot. 2. Построить модель с архитектурой: ○ Conv2D(32, (3,3), activation='relu') ○ Conv2D(32, (3,3), activation='relu') ○ MaxPooling2D((2,2)) ○ Conv2D(64, (3,3), activation='relu') ○ Conv2D(64, (3,3), activation='relu') ○ MaxPooling2D((2,2)) ○ Flatten() ○ Dense(256, activation='relu') ○ Dropout(0.5) ○ Dense(10, activation='softmax') 3. Скомпилировать модель с оптимизатором adam, функцией потерь categorical_crossentropy. 4. Обучить модель на 15 эпохах. 5. Построить графики обучения. 6. Вывести количество параметров модели (model.summary()). 7. Сравнить точность с моделью из варианта 1 (сделать таблицу). 8. Построить confusion matrix.

	9. Эксперимент: изменить kernel_size во всех сверточных слоях на (5,5) и обучить модель заново (5 эпох). Сравнить точность. 10. Сохранить лучшую модель (без эксперимента) в формате SavedModel.
3	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Построить модель с использованием BatchNormalization: - Conv2D(32, (3,3), padding='same') → BatchNormalization() → Activation('relu') → MaxPooling2D((2,2)) - Conv2D(64, (3,3)) → BatchNormalization() → Activation('relu') → MaxPooling2D((2,2)) - Flatten() - Dense(128) → BatchNormalization() → Activation('relu') - Dense(10, 'softmax') 3. Скомпилировать с adam. 4. Обучить на 20 эпохах. 5. Построить графики обучения. 6. Сравнить с моделью без BatchNormalization (из варианта 1) по графикам сходимости и итоговой точности. 7. Визуализировать карты признаков (выход после первого MaxPooling) для одного тестового изображения. 8. Эксперимент: изменить padding='valid' в первом сверточном слое – как изменился размер карт признаков? Вывести размеры до и после. 9. Сохранить модель в .h5. 10. Сделать вывод о влиянии BatchNormalization на скорость обучения и точность.
4	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Построить полносвязную (плотную) сеть для сравнения: - Flatten(input_shape=(32,32,3)) - Dense(1024, activation='relu') - Dense(512, activation='relu') - Dense(10, activation='softmax') 3. Обучить на 20 эпох, зафиксировать accuracy. 4. Построить CNN с аналогичным числом параметров (подобрать архитектуру, например, Conv2D(16) → MaxPool → Conv2D(32) → Flatten → Dense(128) → Dense(10)). 5. Обучить CNN и сравнить accuracy и время обучения. 6. Построить confusion matrix для CNN. 7. Визуализировать карты признаков после второго сверточного слоя для 4 разных тестовых изображений. 8. Эксперимент: добавить strides=2 в первый Conv2D вместо

	MaxPooling – обучить модель и сравнить точность. 9. Сохранить лучшую модель (CNN). 10. Сделать вывод о преимуществах CNN перед полносвязной сетью.
5	1. Загрузить датасет Fashion-MNIST (28×28, ч/б). Добавить размерность канала: X_train = X_train.reshape(-1,28,28,1). 2. Нормализовать пиксели. Метки – целые числа (sparse). 3. Построить CNN: o Conv2D(16, (3,3), activation='relu', input_shape=(28,28,1)) o MaxPooling2D((2,2)) o Conv2D(32, (3,3), activation='relu') o MaxPooling2D((2,2)) o Flatten() o Dense(64, activation='relu') o Dense(10, activation='softmax') 4. Скомпилировать с adam, функцией потерь sparse_categorical_crossentropy. 5. Обучить на 10 эпохах. 6. Оценить accuracy на тесте. 7. Визуализировать фильтры первого сверточного слоя. 8. Визуализировать карты признаков после первого MaxPooling для одного изображения. 9. Эксперимент: увеличить число фильтров в 2 раза (32 и 64) – обучить и сравнить точность и время. 10. Сохранить модель.
6	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Использовать аугментацию данных с ImageDataGenerator: o rotation_range=15 o width_shift_range=0.1 o height_shift_range=0.1 o horizontal_flip=True o zoom_range=0.1 3. Построить CNN (любая архитектура из вариантов 1–3). 4. Обучить модель с аугментацией на 20 эпох (используя datagen.flow()). 5. Обучить такую же модель без аугментации. 6. Построить графики точности на валидации для обеих моделей на одном поле. 7. Сравнить тестовую accuracy. 8. Визуализировать 10 аугментированных изображений из генератора.

	9. Сохранить модель с аугментацией. 10. Сделать вывод о влиянии аугментации на переобучение.
7	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Построить CNN с архитектурой, имитирующей VGG-подобную: ○ Блок 1: Conv2D(32,3,3) → Conv2D(32,3,3) → MaxPooling2D ○ Блок 2: Conv2D(64,3,3) → Conv2D(64,3,3) → MaxPooling2D ○ Блок 3: Conv2D(128,3,3) → Conv2D(128,3,3) → MaxPooling2D ○ Flatten → Dense(256, 'relu') → Dropout(0.5) → Dense(10, 'softmax') 3. Обучить на 15 эпох. 4. Вывести количество параметров. 5. Построить confusion matrix. 6. Визуализировать 10 тестовых изображений, на которых модель ошиблась, с указанием истинного и предсказанного класса. 7. Эксперимент: уменьшить число фильтров вдвое (16,32,64) – обучить и сравнить точность. 8. Использовать plot_model для визуализации архитектуры (сохранить рисунок). 9. Сохранить модель. 10. Сделать вывод о зависимости качества от глубины.
8	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Построить CNN с GlobalAveragePooling2D вместо Flatten: ○ Conv2D(32,3,3, activation='relu') → MaxPooling2D ○ Conv2D(64,3,3, activation='relu') → MaxPooling2D ○ Conv2D(128,3,3, activation='relu') → GlobalAveragePooling2D() ○ Dense(10, 'softmax') 3. Обучить на 15 эпох. 4. Сравнить количество параметров с моделью, использующей Flatten + Dense(256) (из варианта 1). 5. Сравнить accuracy. 6. Визуализировать карты признаков перед GlobalAveragePooling для одного изображения. 7. Эксперимент: добавить Dropout(0.3) перед выходным слоем – сравнить. 8. Построить графики обучения.

	9. Сохранить модель. 10. Сделать вывод о преимуществах GlobalAveragePooling.
9	1. Загрузить CIFAR-10, нормализовать, one-hot. 2. Построить CNN с использованием SeparableConv2D (глубинно-разделимая свертка) вместо обычной: o SeparableConv2D(32,3,3, activation='relu') → MaxPooling2D o SeparableConv2D(64,3,3, activation='relu') → MaxPooling2D o Flatten → Dense(128, 'relu') → Dense(10, 'softmax') 3. Вычислить количество параметров и сравнить с обычной CNN (аналогичной архитектуры). 4. Обучить на 15 эпох. 5. Сравнить accuracy и время обучения с обычной CNN. 6. Визуализировать фильтры SeparableConv2D (они имеют разную структуру). 7. Построить confusion matrix. 8. Эксперимент: изменить kernel_size на (5,5) – сравнить точность. 9. Сохранить модель. 10. Сделать вывод о эффективности SeparableConv2D.
10	1. Загрузить датасет CIFAR-100 (100 классов). 2. Нормализовать пиксели, преобразовать метки в one-hot. 3. Построить CNN: o Conv2D(64,3,3, activation='relu') → MaxPooling2D o Conv2D(128,3,3, activation='relu') → MaxPooling2D o Conv2D(256,3,3, activation='relu') → GlobalAveragePooling2D o Dense(256, 'relu') → Dropout(0.5) o Dense(100, 'softmax') 4. Обучить на 20 эпох. 5. Вывести тестовую accuracy (ожидается 30–40%). 6. Построить confusion matrix для топ-10 самых частых классов (уменьшенную). 7. Визуализировать карты признаков после второго сверточного слоя. 8. Сравнить с полносвязной сетью (если позволяет память) – обучить 5 эпох и сравнить. 9. Сохранить модель. 10. Сделать вывод о сложности CIFAR-100 и применимости CNN.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Чем сверточный слой отличается от полносвязного (плотного)? Какие преимущества это даёт для обработки изображений?
2. Объясните параметры сверточного слоя: `filters`, `kernel_size`, `strides`, `padding`. Как они влияют на размер выходной карты признаков?
3. Какую функцию активации обычно используют после сверточных слоёв и почему?
4. Для чего нужен слой MaxPooling2D? Чем он отличается от AveragePooling2D?
5. Как визуализировать фильтры обученной CNN? Что показывают фильтры первого слоя?
6. Что такое карты признаков (feature maps)? Как визуализировать активации для конкретного входного изображения?
7. Почему CNN лучше подходят для задач классификации изображений, чем полносвязные сети?
8. Что такое `GlobalAveragePooling2D` и в каких случаях его применяют вместо `Flatten`?
9. Как рассчитать количество обучаемых параметров в сверточном слое?
10. Какие методы регуляризации (кроме Dropout) применяются в CNN для борьбы с переобучением?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность архитектуры CNN. Правильное использование слоёв, параметров, соответствие заданию.	10
Качество визуализаций (графики, confusion matrix, фильтры, карты признаков). Наличие подписей, заголовков, читаемость, соответствие требованиям.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ полученных результатов, сравнение с ожиданиями, обоснование выводов.	10
Сравнение с альтернативами (если по варианту). Для вариантов, где требуется сравнение с полносвязной сетью, изменением параметров и т.д. – наличие таблиц и анализа.	10

Лабораторная работа № 28

Аугментация данных для изображений

Цель: Применить аугментацию данных для улучшения обобщающей способности модели.

Теоретическое обоснование

Аугментация данных — это метод искусственного увеличения обучающей выборки путём применения случайных трансформаций к исходным изображениям. Цель — повысить обобщающую способность модели и уменьшить переобучение, особенно когда размер обучающей выборки ограничен.

ImageDataGenerator в Keras предоставляет набор стандартных аугментаций:

- **Повороты** (rotation_range) — случайный поворот в градусах (например, 20°).
- **Сдвиги** (width_shift_range, height_shift_range) — горизонтальное/вертикальное смещение в долях от ширины/высоты.
- **Отражения** (horizontal_flip, vertical_flip) — зеркальное отражение.
- **Масштабирование** (zoom_range) — случайное приближение/отдаление.
- **Изменение яркости** (brightness_range) — случайное изменение яркости.
- **Искажение каналов** (channel_shift_range) — сдвиг цветовых каналов.
- **Нормализация** (rescale) — приведение значений пикселей к диапазону [0,1] или [-1,1].

Принцип работы. ImageDataGenerator не хранит аугментированные изображения в памяти, а генерирует их на лету во время обучения. На каждой эпохе каждое изображение случайным образом трансформируется, поэтому модель видит бесконечно разнообразные вариации исходных данных.

Влияние аугментации на переобучение. Без аугментации модель может запомнить шум и специфические особенности тренировочных изображений, что приводит к низкой точности на тестовой выборке. Аугментация добавляет естественную вариативность (повороты, сдвиги, отражения), заставляя модель учиться инвариантности к этим преобразованиям. Это снижает разрыв между тренировочной и валидационной точностью.

Выбор аугментаций. Не все аугментации подходят для любой задачи. Например, для распознавания цифр вертикальное отражение недопустимо (6 и 9 перепутаются). Для медицинских изображений повороты могут быть

бессмысленными. Аугментации должны сохранять семантическую метку класса.

Визуализация аугментированных изображений. Полезно визуализировать несколько примеров с разными аугментациями, чтобы убедиться, что они разумны и не разрушают содержимое.

Сравнение качества. Экспериментально показывается, что модель, обученная с аугментацией, достигает более высокой тестовой точности при том же или меньшем переобучении (сужение разрыва train/val).

Современные альтернативы: Augmentor, Albumentations, imgaug — библиотеки с более широким набором трансформаций и более высокой производительностью.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет CIFAR-10, нормализовать пиксели, преобразовать метки в one-hot encoding. 2. Создать базовую CNN модель (например, Conv2D(32,3,3) → MaxPooling → Conv2D(64,3,3) → MaxPooling → Flatten → Dense(128) → Dense(10)). 3. Обучить базовую модель без аугментации на 15 эпох, записать итоговую accuracy на тесте и графики обучения. 4. Создать ImageDataGenerator с параметрами: rotation_range=20, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True. 5. Обучить такую же модель с использованием аугментации (datagen.flow()) на 15 эпох. 6. Построить графики точности на валидации для обеих моделей на одном поле. 7. Сравнить тестовую accuracy (таблица). 8. Визуализировать 10 аугментированных изображений из генератора (сетка 2×5). 9. Проанализировать разницу в переобучении (разрыв train/val accuracy). 10. Сохранить модель, обученную с аугментацией.
2	1. Загрузить датасет Fashion-MNIST, добавить размерность канала (28,28,1), нормализовать.

	2. Построить CNN: Conv2D(32,3,3, activation='relu') → MaxPooling2D → Conv2D(64,3,3, activation='relu') → MaxPooling2D → Flatten → Dense(64, 'relu') → Dense(10, 'softmax'). 3. Обучить без аугментации (10 эпох), зафиксировать accuracy. 4. Создать ImageDataGenerator с zoom_range=0.2, shear_range=0.2, rotation_range=10. 5. Обучить с аугментацией (10 эпох). 6. Сравнить графики loss. 7. Визуализировать 10 аугментированных изображений. 8. Эксперимент: добавить brightness_range=[0.8,1.2] и повторить обучение. Сравнить с предыдущим результатом. 9. Сохранить лучшую модель. 10. Сделать вывод о влиянии яркости на качество.
3	1. Загрузить CIFAR-10. 2. Построить CNN с архитектурой: Conv2D(32,3,3) → BatchNormalization → Activation('relu') → MaxPooling2D → Conv2D(64,3,3) → BatchNormalization → Activation('relu') → MaxPooling2D → Flatten → Dense(256) → Dropout(0.5) → Dense(10). 3. Обучить без аугментации (20 эпох). 4. Использовать ImageDataGenerator только с horizontal_flip=True и обучить заново. 5. Сравнить тестовую accuracy и кривые обучения. 6. Добавить rotation_range=15 и zoom_range=0.1 – снова обучить. 7. Визуализировать примеры аугментаций для одного изображения (оригинал + 5 вариантов). 8. Построить таблицу сравнения трёх подходов (без аугментации, только flip, полная аугментация). 9. Сохранить модель с полной аугментацией. 10. Сделать вывод о том, какая аугментация дала наибольший прирост.
4	1. Загрузить CIFAR-10. 2. Создать модель VGG-подобную: два блока (Conv2D(32,3,3) → Conv2D(32,3,3) → MaxPooling2D) и (Conv2D(64,3,3) → Conv2D(64,3,3) → MaxPooling2D) → Flatten → Dense(256) → Dropout(0.5) → Dense(10). 3. Обучить без аугментации (15 эпох). 4. Создать ImageDataGenerator с width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True. 5. Обучить с аугментацией. 6. Построить confusion matrix для модели с аугментацией.

	7. Визуализировать 10 аугментированных изображений. 8. Эксперимент: изменить fill_mode='nearest' на 'reflect' – сравнить визуально. 9. Сохранить модель. 10. Сделать вывод о влиянии сдвигов.
5	1. Загрузить датасет MNIST (28,28,1), нормализовать. 2. Построить простую CNN. 3. Обучить без аугментации (5 эпох) – получить baseline. 4. Создать ImageDataGenerator с rotation_range=30 и zoom_range=0.2. 5. Обучить с аугментацией (5 эпох). 6. Визуализировать аугментированные изображения – есть ли недопустимые трансформации (например, 6 похожа на 9)? 7. Ограничить rotation_range=10 и повторить. Сравнить accuracy. 8. Добавить shear_range=0.1 – проверить, не портятся ли цифры. 9. Сделать вывод о том, какие аугментации не подходят для MNIST. 10. Сохранить лучшую модель.
6	1. Загрузить CIFAR-10. 2. Использовать предобученную модель (например, MobileNetV2) с замороженными слоями и добавить свои классификационные слои. 3. Обучить без аугментации (10 эпох). 4. Обучить с аугментацией (горизонтальное отражение + поворот на 15°). 5. Сравнить accuracy. 6. Визуализировать аугментированные изображения. 7. Эксперимент: добавить channel_shift_range=0.1 (изменение цвета) – сравнить. 8. Построить графики сравнения. 9. Сохранить модель с лучшей аугментацией. 10. Сделать вывод о важности аугментации при трансферном обучении.
7	1. Загрузить CIFAR-10. 2. Создать CNN с несколькими Dropout слоями. 3. Обучить без аугментации (20 эпох). 4. Обучить с аугментацией (rotation_range=20, horizontal_flip=True, zoom_range=0.1). 5. Построить графики разницы train-val accuracy для обеих моделей. 6. Сравнить тестовую accuracy. 7. Визуализировать 20 аугментированных изображений в сетке 4×5. 8. Эксперимент: обучить модель с аугментацией, но без Dropout – сравнить. 9. Сохранить модель.

	10. Сделать вывод о совместном использовании аугментации и Dropout.
8	1. Загрузить датасет cats_vs_dogs (из tensorflow_datasets или малую выборку). 2. Создать CNN для бинарной классификации. 3. Обучить без аугментации (10 эпох). 4. Использовать ImageDataGenerator с horizontal_flip=True, rotation_range=20, brightness_range=[0.8,1.2]. 5. Обучить с аугментацией. 6. Сравнить accuracy и графики. 7. Визуализировать аугментированные изображения кошек/собак. 8. Эксперимент: добавить shear_range=0.2 – улучшило ли качество? 9. Сохранить модель. 10. Сделать вывод о применимости аугментации для бинарной классификации.
9	1. Загрузить CIFAR-10. 2. Создать CNN с GlobalAveragePooling2D. 3. Обучить без аугментации (15 эпох). 4. Создать два генератора: один с аугментацией (повороты, сдвиги, отражения), второй – только с нормализацией. 5. Обучить модель с аугментацией и сравнить. 6. Визуализировать карты признаков для аугментированного и неаугментированного изображения. 7. Построить таблицу сравнения времени обучения. 8. Эксперимент: увеличить интенсивность аугментации (rotation_range=40, zoom_range=0.3) – сравнить. 9. Сохранить модель. 10. Сделать вывод о пределе полезной интенсивности аугментации.
10	1. Загрузить CIFAR-10. 2. Создать CNN с архитектурой из варианта 1. 3. Обучить модель на уменьшенной выборке (только 5000 изображений из train) без аугментации. 4. Обучить модель на той же уменьшенной выборке с интенсивной аугментацией (повороты, сдвиги, отражения, изменение яркости). 5. Сравнить accuracy – насколько аугментация помогает при малом объёме данных. 6. Визуализировать аугментированные изображения. 7. Построить графики обучения. 8. Обучить модель на полной выборке без аугментации для сравнения. 9. Сохранить модель с аугментацией (малая выборка).

	10. Сделать вывод о критической важности аугментации при дефиците данных.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое аугментация данных и с какой целью она применяется в задачах компьютерного зрения?
2. Какие трансформации можно реализовать с помощью ImageDataGenerator в Keras? Приведите 5 примеров.
3. Как аугментация помогает бороться с переобучением? Объясните механизм.
4. Почему для датасета MNIST не следует использовать horizontal_flip? Приведите пример недопустимой аугментации.
5. Что делает параметр fill_mode в ImageDataGenerator? Какие значения он принимает и как они влияют на результат?
6. Как работает zoom_range? Что означают значения (0.8, 1.2)?
7. Как правильно использовать ImageDataGenerator с flow() или flow_from_directory() для обучения модели?
8. Как визуализировать аугментированные изображения, чтобы убедиться в корректности трансформаций?
9. Влияет ли аугментация на время обучения модели? Если да, то как?

10.Может ли слишком сильная аугментация ухудшить качество модели? Почему?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность настройки аугментации. Правильный выбор параметров ImageDataGenerator для данного датасета.	10
Качество визуализаций (сравнительные графики, аугментированные изображения). Наличие подписей, легенд, читаемость.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Глубокий анализ влияния аугментации, сравнение с baseline.	10
Сравнение разных стратегий аугментации (если по варианту). Таблицы, графики, обоснование выбора.	10

Лабораторная работа № 29

Трансферное обучение с предобученными CNN

Цель: Использовать предобученную модель для решения задачи классификации на новом датасете.

Теоретическое обоснование

Трансферное обучение (Transfer Learning) — это метод, при котором модель, предварительно обученная на большом наборе данных (например, ImageNet с 1.2 миллионами изображений и 1000 классов), адаптируется к новой задаче с ограниченным количеством данных. Это позволяет достичь высокой точности, не требуя огромных вычислительных ресурсов и данных.

Предобученные модели. Наиболее популярные архитектуры:

- **VGG16/VGG19** — глубокая сеть с однородными блоками (3×3 свертки), много параметров (138 млн).
- **ResNet50 (Residual Network)** — содержит «остаточные связи» (skip connections), позволяющие обучать очень глубокие сети (50, 101, 152 слоя) без исчезающего градиента.
- **Inception (GoogLeNet)** — использует модули с разными размерами ядер (1×1, 3×3, 5×5) для захвата признаков разных масштабов.
- **EfficientNet** — современная семейство, балансирующая глубину, ширину и разрешение; достигает высокого качества при малом числе параметров.

include_top=False. При загрузке предобученной модели мы отбрасываем исходный классификационный верх (полносвязные слои), оставляя только свёрточную базу (feature extractor). Это позволяет адаптировать сеть к новому числу классов.

Заморозка весов. Заморозка (layer.trainable = False) означает, что веса предобученной свёрточной базы не обновляются во время обучения. Таким образом, модель использует общие низкоуровневые признаки (края, текстуры), извлечённые из ImageNet, и настраивает только новые классификационные слои.

Добавление новых слоёв. Поверх свёрточной базы добавляются:

- GlobalAveragePooling2D или Flatten — для преобразования карт признаков в вектор.
- Один или несколько полносвязных слоев (например, Dense(256,

activation='relu')).

- Выходной слой (Dense с softmax или sigmoid для нового числа классов).

Обучение. Сначала обучаются только добавленные слои (свёрточная база заморожена). После достижения хорошего качества можно **дообучить (fine-tune)** — разморозить верхние слои свёрточной базы и продолжить обучение с очень маленькой скоростью обучения, чтобы адаптировать высокоуровневые признаки к новой задаче.

Сравнение с обучением с нуля. Обучение CNN с нуля на малом датасете (например, 1000 изображений кошек и собак) приводит к сильному переобучению. Трансферное обучение позволяет достичь точности 95–99% даже с небольшим количеством данных, так как предобученная модель уже умеет выделять универсальные признаки.

Когда использовать трансферное обучение: всегда, когда новый датасет похож на исходный (например, фотографии реальных объектов), а размер выборки ограничен.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет CIFAR-10, нормализовать пиксели, преобразовать метки в one-hot encoding. 2. Загрузить предобученную модель VGG16 с весами ImageNet, include_top=False, input_shape=(32,32,3). 3. Заморозить все слои VGG16 (layer.trainable = False). 4. Добавить поверх VGG16: GlobalAveragePooling2D(), Dense(256, activation='relu'), Dropout(0.5), Dense(10, activation='softmax'). 5. Скомпилировать модель с оптимизатором adam, функцией потерь categorical_crossentropy. 6. Обучить только добавленные слои на 10 эпох. 7. Оценить точность на тестовой выборке. 8. Разморозить последние 2 блока VGG16 (последние 4 слоя) и выполнить fine-tuning с очень маленькой скоростью обучения (например, 1e-5) на 5 эпох. 9. Сравнить точность до и после fine-tuning (таблица).

	10. Сохранить финальную модель.
2	1. Загрузить CIFAR-10. 2. Загрузить ResNet50 с include_top=False, input_shape=(32,32,3). 3. Заморозить все слои ResNet50. 4. Добавить: GlobalAveragePooling2D(), Dense(512, activation='relu'), BatchNormalization(), Dropout(0.5), Dense(10, activation='softmax'). 5. Обучить на 15 эпох. 6. Оценить тестовую accuracy. 7. Разморозить все слои ResNet50, установить очень маленький learning rate (1e-5) и выполнить fine-tuning на 5 эпох. 8. Сравнить результаты. 9. Построить confusion matrix после fine-tuning. 10. Сохранить модель.
3	1. Загрузить датасет cats_vs_dogs (из tensorflow_datasets или малая выборка из 2000 изображений). 2. Использовать EfficientNetB0 с include_top=False. 3. Заморозить базовую модель. 4. Добавить: GlobalAveragePooling2D(), Dense(128, activation='relu'), Dropout(0.3), Dense(1, activation='sigmoid'). 5. Обучить на 10 эпох. 6. Оценить accuracy и AUC-ROC. 7. Разморозить последние 3 блока EfficientNet и выполнить fine-tuning с learning_rate=1e-5 на 5 эпох. 8. Сравнить точность. 9. Визуализировать несколько изображений с предсказаниями (кошка/собака). 10. Сохранить лучшую модель.
4	1. Загрузить CIFAR-10. 2. Загрузить MobileNetV2 с include_top=False. 3. Заморозить все слои. 4. Добавить: GlobalMaxPooling2D(), Dense(128, activation='relu'), Dense(10, activation='softmax'). 5. Обучить на 10 эпох. 6. Оценить точность. 7. Эксперимент: вместо заморозки всех слоёв, заморозить только первые 50% слоёв (по номеру) – обучить. Сравнить. 8. Для лучшей конфигурации построить confusion matrix. 9. Визуализировать архитектуру модели с помощью plot_model. 10. Сохранить модель.
5	1. Загрузить датасет цветов (Flowers, 5 классов) – например,

	из tensorflow_datasets. 2. Использовать предобученную VGG16. 3. Заморозить VGG16. 4. Добавить: Flatten() (вместо GAP), Dense(1024, activation='relu'), Dropout(0.5), Dense(5, activation='softmax'). 5. Обучить на 20 эпох. 6. Оценить accuracy. 7. Выполнить fine-tuning последнего блока VGG16 на 10 эпохах с lr=1e-5. 8. Сравнить точность. 9. Построить графики обучения. 10. Сохранить модель.
6	1. Загрузить CIFAR-100. 2. Использовать ResNet50. 3. Заморозить ResNet50. 4. Добавить: GlobalAveragePooling2D(), Dense(512, activation='relu'), BatchNormalization(), Dropout(0.5), Dense(100, activation='softmax'). 5. Обучить на 20 эпох. 6. Вычислить top-5 accuracy. 7. Разморозить все слои ResNet50 и выполнить fine-tuning с lr=1e-5 на 10 эпох. 8. Сравнить top-5 accuracy. 9. Построить confusion matrix для топ-10 классов. 10. Сохранить модель.
7	1. Загрузить датасет chest_xray (пневмония/норма) – малая выборка. 2. Использовать InceptionV3 с include_top=False. 3. Заморозить InceptionV3. 4. Добавить: GlobalAveragePooling2D(), Dense(128, activation='relu'), Dropout(0.5), Dense(1, activation='sigmoid'). 5. Обучить на 15 эпох. 6. Оценить accuracy, precision, recall, F1. 7. Выполнить fine-tuning последних 3 слоёв InceptionV3 на 5 эпохах. 8. Сравнить метрики. 9. Построить ROC-кривую. 10. Сохранить модель.
8	1. Загрузить CIFAR-10. 2. Загрузить две предобученные модели: VGG16 и ResNet50. 3. Для каждой построить классификатор (замороженная база +

	GAP + Dense(256) + Dense(10)). 4. Обучить каждую на 10 эпох. 5. Сравнить точность, количество параметров, время обучения. 6. Выбрать лучшую модель. 7. Для лучшей модели выполнить fine-tuning (разморозить последние 2 блока) на 5 эпохах. 8. Построить confusion matrix. 9. Сохранить лучшую модель. 10. Сделать вывод о выборе архитектуры для CIFAR-10.
9	1. Загрузить датасет horses_or_humans (бинарная классификация). 2. Использовать MobileNetV2. 3. Заморозить MobileNetV2. 4. Добавить: GlobalAveragePooling2D(), Dense(64, activation='relu'), Dense(1, activation='sigmoid'). 5. Обучить на 10 эпох. 6. Оценить accuracy. 7. Применить аугментацию (горизонтальное отражение, поворот на 10°) и повторить обучение. 8. Сравнить с моделью без аугментации. 9. Визуализировать предсказания на тестовых изображениях. 10. Сохранить модель с аугментацией.
10	1. Загрузить CIFAR-10. 2. Загрузить предобученную модель (любую) и заморозить её. 3. Добавить новые слои и обучить. 4. Эксперимент: обучить такую же модель «с нуля» (без предобученных весов) на 20 эпох. 5. Сравнить точность и скорость сходимости (графики loss). 6. Построить таблицу сравнения. 7. Для модели с трансферным обучением выполнить fine-tuning. 8. Визуализировать карты признаков из предобученной модели для одного изображения. 9. Сохранить обе модели (трансфер и с нуля). 10. Сделать вывод о преимуществах трансферного обучения.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания

с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.

4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое трансферное обучение? Почему оно эффективно для задач компьютерного зрения?
2. Что означает параметр include_top=False при загрузке предобученной модели?
3. Зачем замораживать веса предобученной модели? Как это сделать в Keras?
4. Что такое fine-tuning? Когда его следует применять и почему используют маленькую скорость обучения?
5. Какие предобученные архитектуры CNN наиболее популярны? Назовите их особенности (VGG, ResNet, MobileNet, EfficientNet).
6. Как добавить новые классификационные слои поверх предобученной свёрточной базы?
7. В чём разница между GlobalAveragePooling2D и Flatten при построении головы классификатора?
8. Как выбрать, сколько слоёв размораживать при fine-tuning?
9. Можно ли использовать трансферное обучение, если новый датасет сильно отличается от ImageNet (например, медицинские снимки)?
10. Как оценить, что fine-tuning улучшил модель, а не ухудшил?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность загрузки и настройки предобученной модели. Правильное использование include_top, заморозка, добавление слоёв.	10

Проведение fine-tuning. Корректное размораживание, выбор маленького learning rate, сравнение.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ прироста качества, сравнение с обучением с нуля, обоснование.	10
Визуализации (графики, confusion matrix). Качество, подписи.	10

Лабораторная работа № 30

Обнаружение объектов с YOLO

Цель: Применить готовую модель YOLO для обнаружения объектов на изображениях и видео.

Теоретическое обоснование

Обнаружение объектов — задача компьютерного зрения, заключающаяся в локализации (выделение bounding box) и классификации объектов на изображении. YOLO (You Only Look Once) — семейство одноэтапных детекторов, отличающихся высокой скоростью и хорошей точностью. В отличие от двухэтапных методов (Faster R-CNN), YOLO выполняет предсказание за один проход сети.

Принцип работы YOLO (обобщённо). Входное изображение делится на сетку ячеек (например, S×S). Каждая ячейка отвечает за предсказание:

- B bounding box (координаты центра, ширина, высота, уверенность).
- Условные вероятности классов для объектов, центр которых попадает в ячейку.

Итоговое предсказание — набор bounding box с классами и оценкой уверенности. После подавления немаксимумов (Non-Maximum Suppression, NMS) остаются только наиболее вероятные объекты.

YOLOv8 (Ultralytics) — современная версия с улучшенной архитектурой (CSPDarknet, PANet для многомасштабных признаков, анкор-фри детекция). Поддерживает обнаружение, сегментацию, классификацию и трекинг.

Основные возможности YOLOv8:

- Детекция на изображениях, видео, в реальном времени.
- Вывод результатов: bounding boxes (координаты), классы, уверенность (confidence).
- Предобученные модели на COCO (80 классов).
- Простой API: model = YOLO('yolov8n.pt'); results = model('image.jpg').

Визуализация результатов. Модель возвращает аннотированные изображения с нарисованными рамками и подписями классов. Можно также извлечь сырые данные для дальнейшей обработки.

Тонкая настройка (fine-tuning). Для детекции объектов на пользовательском датасете (например, распознавание конкретных деталей на производстве) необходимо:

1. Подготовить датасет в формате YOLO (изображения и текстовые файлы с аннотациями: класс, x_center, y_center, width, height в нормированных координатах).
2. Использовать предобученную модель и дообучить её на новом датасете с малым learning rate и количеством эпох.

Ограничения YOLO: не очень хорошо работает с мелкими объектами или объектами, сильно перекрывающимися; может давать ложные срабатывания при низком пороге уверенности.

Применение: системы видеонаблюдения, беспилотные автомобили, распознавание товаров на полках, медицинская визуализация.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Установить библиотеку Ultralytics (pip install ultralytics). 2. Загрузить предобученную модель yolov8n.pt (nano) с помощью YOLO('yolov8n.pt'). 3. Выполнить детекцию на тестовом изображении (например, https://ultralytics.com/images/bus.jpg). 4. Визуализировать результат — сохранить изображение с нарисованными bounding boxes.

	5. Извлечь и вывести в консоль для каждого обнаруженного объекта: координаты рамки, класс, уверенность (confidence). 6. Отфильтровать результаты по порогу уверенности 0.5 — вывести только объекты с confidence ≥ 0.5. 7. Выполнить детекцию на локальном видеофайле (первые 30 кадров) и сохранить результат. 8. Использовать модель yolov8s.pt (small) на том же изображении и сравнить количество обнаруженных объектов и время выполнения. 9. Ознакомительно: подготовить пользовательский датасет из 5 изображений в формате YOLO (аннотации вручную) и запустить тренировку на 1 эпоху (демонстрация процесса). 10. Сохранить результаты детекции для изображения в формате JSON (список объектов с классами и координатами).
2	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Запустить детекцию на веб-камере (source=0) на 50 кадров, отображая результат в реальном времени. 3. Для каждого кадра вывести количество обнаруженных объектов. 4. Изменить порог уверенности на 0.7 и повторить детекцию на том же видеофайле — сравнить количество обнаруженных объектов. 5. Использовать модель yolov8m.pt (medium) на тестовом изображении, замерить время выполнения и сравнить с yolov8n.pt. 6. Выполнить детекцию на группе изображений из папки, сохранить все результаты. 7. Извлечь координаты bounding box для самого уверенного объекта на изображении. 8. Визуализировать результат детекции, подписав на изображении класс и уверенность. 9. Ознакомительно: скачать пользовательский датасет с Roboflow в формате YOLOv8 и запустить тренировку на 5 эпох. 10. Сохранить обученную модель в формате ONNX (model.export(format='onnx')).
3	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Выполнить детекцию на удалённом видео по URL. 3. Сохранить видео с нарисованными bounding boxes. 4. Вывести для каждого кадра список обнаруженных объектов (класс, уверенность). 5. Использовать модель yolov8x.pt (extra large) на том же видео

	(первые 20 кадров) — сравнить точность и скорость. 6. Отфильтровать объекты по классам (например, только 'person' и 'car') — вывести только их. 7. Визуализировать результат детекции в виде сетки из 4 изображений (2×2). 8. Извлечь и вывести в консоль confidence score для самого ненадёжного обнаруженного объекта (минимальная уверенность). 9. Ознакомительно: подготовить data.yaml для пользовательского датасета (указать пути и классы). 10. Выполнить экспорт модели в TensorRT формат (model.export(format='engine')).
4	1. Установить Ultralytics, загрузить yolov8n-seg.pt (модель для сегментации). 2. Выполнить сегментацию на тестовом изображении. 3. Визуализировать результат (маски + bounding boxes). 4. Извлечь маски сегментации для каждого обнаруженного объекта. 5. Вывести площадь (количество пикселей) каждой маски. 6. Сравнить количество параметров моделей yolov8n.pt и yolov8n-seg.pt. 7. Выполнить сегментацию на видеофайле, сохранить результат. 8. Отфильтровать объекты по порогу уверенности 0.6. 9. Ознакомительно: запустить тренировку модели сегментации на пользовательском датасете (1 эпоха). 10. Сохранить модель в формате CoreML (model.export(format='coreml')).
5	1. Установить Ultralytics, загрузить yolov8n-pose.pt (модель для оценки позы). 2. Выполнить детекцию ключевых точек на изображении с людьми. 3. Визуализировать результат — нарисовать скелеты. 4. Извлечь координаты ключевых точек для первого обнаруженного человека. 5. Вывести количество обнаруженных людей на изображении. 6. Использовать модель yolov8n.pt для обычной детекции на том же изображении — сравнить, сколько людей обнаружено. 7. Выполнить оценку позы на видео, сохранить результат. 8. Отфильтровать людей с уверенностью < 0.5. 9. Ознакомительно: подготовить пользовательский датасет для оценки позы и запустить тренировку на 1 эпоху.

	10. Сохранить модель в формате OpenVINO (model.export(format='openvino')).
6	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Выполнить детекцию на скриншоте экрана (сохранить скриншот программно). 3. Визуализировать результат. 4. Вывести общее количество обнаруженных объектов всех классов. 5. Создать функцию, которая принимает путь к изображению и возвращает список объектов с координатами и уверенностью. 6. Протестировать функцию на 5 разных изображениях, вывести сводную таблицу. 7. Изменить размер входного изображения (imgsz=320) и сравнить время детекции с imgsz=640. 8. Использовать Non-Maximum Suppression (NMS) с порогом iou=0.3 — как изменилось количество обнаруженных объектов? 9. Ознакомительно: скачать датасет COCO128 и запустить тренировку на нём (3 эпохи) для знакомства с процессом. 10. Сохранить обученную модель в формате TorchScript (model.export(format='torchscript')).
7	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Выполнить детекцию на изображении, содержащем перекрывающиеся объекты. 3. Проанализировать, как NMS обрабатывает перекрытия — вывести количество объектов до и после NMS. 4. Изменить порог NMS (iou=0.1) и сравнить результат. 5. Выполнить детекцию на изображении с мелкими объектами — оценить качество. 6. Использовать модель yolov8n.pt с увеличенным разрешением (imgsz=1280) — сравнить точность и время. 7. Визуализировать confidence scores в виде гистограммы для всех обнаруженных объектов на изображении. 8. Эксперимент: применить аугментацию (поворот, отражение) к тестовому изображению и сравнить результаты детекции. 9. Ознакомительно: обучить модель на пользовательском датасете с использованием трансферного обучения (5 эпох, загрузка предобученных весов). 10. Сохранить модель в формате TFLite (model.export(format='tflite')).
8	1. Установить Ultralytics, загрузить yolov8n.pt.

	2. Выполнить детекцию на видео с движущимися объектами. 3. Для каждого кадра вывести количество обнаруженных объектов. 4. Построить график изменения количества объектов по кадрам (первые 100 кадров). 5. Извлечь координаты bounding box объекта с максимальной уверенностью в каждом кадре. 6. Выполнить детекцию с разными моделями (yolov8n.pt, yolov8s.pt, yolov8m.pt) на одном изображении — сравнить FPS (frames per second). 7. Отфильтровать объекты, выходящие за пределы определённой области на изображении (ROI — region of interest). 8. Визуализировать результат детекции с цветовой кодировкой классов. 9. Ознакомительно: подготовить аннотации для 20 изображений в формате YOLO с помощью LabelImg. 10. Сохранить модель в формате PaddlePaddle (model.export(format='paddle')).
9	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Выполнить детекцию на изображении, содержащем объекты разных размеров (крупные и мелкие). 3. Проанализировать, для каких размеров объектов модель работает лучше. 4. Использовать модель yolov8x.pt — сравнить точность детекции мелких объектов. 5. Выполнить детекцию на ночном изображении (с низкой освещённостью) — оценить качество. 6. Применить предобработку (увеличение яркости, контраста) к тестовому изображению и сравнить результаты детекции. 7. Визуализировать confidence scores на самом изображении (подписать каждую рамку). 8. Эксперимент: изменить порог уверенности с 0.25 на 0.75 и оценить, как изменились precision и recall. 9. Ознакомительно: обучить модель на датасете с дисбалансом классов (например, 90% одного класса, 10% другого). 10. Сохранить модель в формате NVIDIA Triton (model.export(format='triton')).
10	1. Установить Ultralytics, загрузить yolov8n.pt. 2. Выполнить детекцию на изображении с панорамой (широкий угол). 3. Проанализировать искажения bounding boxes по краям

	изображения. 4. Использовать модель yolov8n.pt с параметром half=True (FP16 precision) — сравнить скорость и точность. 5. Выполнить детекцию на видео с изменяющимся освещением — оценить робастность. 6. Извлечь и сохранить в CSV все обнаруженные объекты для видеофайла (кадр, класс, координаты, уверенность). 7. Визуализировать распределение классов на тестовом изображении (круговая диаграмма). 8. Эксперимент: добавить шум (гауссовский) к тестовому изображению и сравнить результаты детекции. 9. Ознакомительно: обучить модель на пользовательском датасете с использованием многомасштабного обучения (multi-scale training). 10. Сохранить модель в формате WebNN (model.export(format='webnn')).

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое задача обнаружения объектов (object detection)? Чем она отличается от классификации изображений и семантической сегментации?
2. Каков принцип работы YOLO (You Only Look Once)? Почему алгоритм получил такое название?
3. Что такое bounding box? Как он представляется в YOLO (формат координат x_center, y_center, width, height)?

4. Что означает confidence score в YOLO? Как он интерпретируется?
5. Как работает Non-Maximum Suppression (NMS) и зачем он нужен в задачах детекции?
6. Какие типы моделей существуют в экосистеме YOLOv8 (detection, segmentation, pose, classification)? Для каких задач каждая предназначена?
7. Как загрузить предобученную модель YOLO и выполнить инференс на изображении с помощью библиотеки Ultralytics?
8. Как настроить порог уверенности и порог NMS при инференсе? Как эти параметры влияют на результаты детекции?
9. Что такое fine-tuning (дообучение) модели YOLO на пользовательском датасете? В каких случаях это применяется?
10. В каких форматах можно экспортировать обученную модель YOLO для последующего развертывания (ONNX, TensorRT, CoreML и др.)?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Правильность использования YOLO для детекции. Корректная загрузка модели, выполнение инференса, извлечение результатов.	10
Анализ результатов и эксперименты с параметрами. Сравнение порогов уверенности, NMS, размеров модели, интерпретация изменений.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Глубокий анализ качества детекции, обоснование выбора параметров.	10
Визуализации (скриншоты с bounding boxes, графики). Наличие подписей, читаемость, соответствие требованиям.	10

Лабораторная работа № 31

Предобработка текста. Bag-of-Words и TF-IDF

Цель: Преобразовать текстовые данные в числовые признаки.

Теоретическое обоснование

Машинное обучение работает с числами, поэтому текст необходимо преобразовать в числовые векторы. Классические методы (до появления глубокого обучения) основаны на подсчёте частот слов.

Предобработка текста с NLTK/spaCy.

- **Токенизация** — разбиение текста на слова, знаки препинания, числа. Например, "Я люблю ML" → ["Я", "люблю", "ML"].
- **Приведение к нижнему регистру** — унификация регистра ("ML" → "ml").
- **Удаление стоп-слов** — удаление частых, но малоинформативных слов (предлогов, союзов, местоимений). Списки стоп-слов есть для многих языков.
- **Стемминг** — усечение слов до основы (например, "люблю", "любил", "любить" → "люб"). Грубый, быстрый алгоритм (PorterStemmer, SnowballStemmer).
- **Лемматизация** — приведение к нормальной форме (словарной) с учётом части речи ("люблю" → "любить"). Требует словаря, точнее, но медленнее.

Bag-of-Words (BoW). Модель мешка слов. Создаётся словарь всех уникальных слов в корпусе. Каждый документ представляется вектором, где элемент i — количество вхождений i-го слова. Параметры CountVectorizer:

- max_features — ограничение размера словаря (топ N частотных слов).
- min_df — минимальная частота документа для включения слова.
- max_df — максимальная частота (игнорировать слишком общие слова).
- ngram_range — учёт n-грамм (например, (1,2) — униграммы и биграммы).

TF-IDF (Term Frequency — Inverse Document Frequency). Улучшение BoW. TF-IDF взвешивает слово по важности:

- **TF** (частота в документе) — отношение количества вхождений слова в документе к общему числу слов в документе.
- **IDF** (обратная частота документа) — log((N+1) / (df+1)) + 1, где N — число документов, df — число документов, содержащих слово. Редкие слова

получают больший вес.

- TF-IDF = TF × IDF.

TF-IDF уменьшает вес слишком частых слов (например, «и», «в») и увеличивает вес специфических, информативных слов.

Применение к классификации текстов. Полученная матрица признаков (документы × термины) подаётся на вход классификатору (логистическая регрессия, SVM, Random Forest). Для анализа тональности отзывов (положительные/отрицательные) эти методы дают хороший базовый уровень.

Ограничения BoW и TF-IDF: не учитывают порядок слов, теряют семантику, создают разреженные и высокоразмерные вектора. Не работают с омонимией и синонимией.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет IMDB для анализа тональности (через keras.datasets.imdb или load_files). Вывести количество положительных и отрицательных отзывов. 2. Выполнить предобработку текста: привести к нижнему регистру, удалить пунктуацию и цифры с помощью регулярных выражений. Вывести первые 3 документа после очистки. 3. Выполнить токенизацию с помощью NLTK (nltk.tokenize.word_tokenize) и подсчитать общее количество уникальных токенов. 4. Удалить стоп-слова (английские) с использованием nltk.corpus.stopwords. Вывести размер словаря до и после удаления стоп-слов. 5. Применить стемминг с помощью PorterStemmer или SnowballStemmer. Сравнить результат для слова "running" и "ran". 6. Создать матрицу Bag-of-Words с CountVectorizer(max_features=5000). Вывести размерность полученной матрицы. 7. Преобразовать тексты в TF-IDF матрицу с помощью TfidfVectorizer(max_features=5000). Объяснить, почему TF-IDF лучше учитывает важность слов.

	8. Разделить данные на обучающую (80%) и тестовую (20%) выборки. 9. Обучить логистическую регрессию на TF-IDF признаках. Вычислить accuracy, precision, recall, F1-score. 10. Визуализировать confusion matrix и вывести топ-10 наиболее важных слов (по коэффициентам модели).
2	1. Загрузить датасет новостей (например, fetch_20newsgroups с 3 категориями). Вывести названия категорий и количество документов в каждой. 2. Выполнить очистку текста: удаление заголовков писем (строк, начинающихся с "From:"), чисел, пунктуации. 3. Применить лемматизацию с помощью spaCy (модель en_core_web_sm). Сравнить результат лемматизации со стеммингом для слова "better". 4. Использовать CountVectorizer с параметрами ngram_range=(1,2), max_df=0.8, min_df=5. Вывести количество признаков. 5. Визуализировать 20 наиболее частотных униграмм и биграмм для одного из классов в виде горизонтальных bar chart. 6. Преобразовать тексты в TF-IDF матрицу. Вычислить среднее TF-IDF значение для слова "и" (англ. "and") – почему оно низкое? 7. Разделить данные на train/test (75/25) со стратификацией. 8. Обучить наивный байесовский классификатор (MultinomialNB) на Bag-of-Words и TF-IDF. Сравнить точность. 9. Построить confusion matrix и classification report. 10. Эксперимент: изменить max_features (1000, 5000, 10000) и сравнить accuracy. Построить график зависимости.
3	1. Загрузить датасет SMS Spam Collection (определение спама). Вывести процент спам-сообщений. 2. Очистить текст: привести к нижнему регистру, удалить лишние пробелы и знаки препинания. 3. Токенизировать текст с помощью spaCy. Вывести топ-10 самых частых токенов до удаления стоп-слов. 4. Удалить стоп-слова с помощью spaCy (token.is_stop). Сравнить топ-10 токенов после удаления. 5. Применить лемматизацию через spaCy. Вывести леммы для предложения: "They were running faster than the cars". 6. Создать Bag-of-Words с CountVectorizer(max_features=3000, binary=True). Что делает параметр binary=True?

	7. Создать TF-IDF матрицу с TfidfVectorizer(max_features=3000, use_idf=True, smooth_idf=True). 8. Разделить данные на train/test (80/20). 9. Обучить логистическую регрессию с регуляризацией (C=1). Вычислить accuracy, precision, recall для класса "spam". 10. Построить ROC-кривую и вычислить AUC-ROC. Сравнить с моделью без удаления стоп-слов.
4	1. Загрузить датасет bbc-news (5 категорий). Вывести распределение категорий. 2. Очистить текст: удалить числа, пунктуацию, привести к нижнему регистру. Использовать регулярные выражения. 3. Токенизировать текст с помощью nltk.word_tokenize. Вывести среднюю длину документа в токенах. 4. Применить стемминг с помощью SnowballStemmer('english'). Сравнить слова "connecting", "connected", "connection" после стемминга. 5. Создать Bag-of-Words с CountVectorizer(max_features=10000, min_df=3, max_df=0.7). Вывести размерность. 6. Преобразовать в TF-IDF. Использовать TfidfVectorizer(sublinear_tf=True) – что делает sublinear_tf? 7. Разделить данные на train/test (70/30). 8. Обучить SVM (SVC) на TF-IDF признаках. Вычислить accuracy и macro F1. 9. Построить confusion matrix в виде тепловой карты. 10. Эксперимент: добавить ngram_range=(1,3) и сравнить точность.
5	1. Загрузить русскоязычный датасет отзывов (например, с кинотеатров). Вывести несколько примеров. 2. Очистить текст от эмодзи и URL. Привести к нижнему регистру, удалить пунктуацию. 3. Токенизировать с помощью nltk.tokenize.word_tokenize (русская модель). Вывести количество уникальных токенов. 4. Удалить русские стоп-слова из NLTK (или stop-words). Вывести список первых 10 стоп-слов. 5. Применить стемминг для русского языка (PyMorphy2 или SnowballStemmer). Сравнить слова "нога", "ногой", "ногами". 6. Создать Bag-of-Words с CountVectorizer(max_features=8000). Вывести 10 самых частых слов. 7. Преобразовать в TF-IDF с TfidfVectorizer(max_features=8000,

	norm='l2'). 8. Разделить данные на train/test (80/20) со стратификацией. 9. Обучить логистическую регрессию. Вычислить accuracy и F1-weighted. 10. Эксперимент: сравнить качество модели со стеммингом и без него (только лемматизация).
6	1. Загрузить датасет Twitter Sentiment Analysis (положительные/отрицательные твиты). Вывести примеры с длиной > 10 слов. 2. Очистить текст: удалить упоминания (@username), хэштеги (оставить текст без #), ссылки. 3. Токенизировать с сохранением эмодзи (перевести в текст). Использовать библиотеку emoji. 4. Удалить стоп-слова с помощью spaCy. Добавить в стоп-слова слова "RT", "http", "https". 5. Применить лемматизацию через spaCy. Сравнить леммы для "am", "are", "is". 6. Создать Bag-of-Words с CountVectorizer(analyzer='word', token_pattern=r'\b[a-zA-Z]+\b'). Что делает token_pattern? 7. Преобразовать в TF-IDF. Использовать TfidfVectorizer(stop_words='english'). 8. Разделить данные на train/test (75/25). 9. Обучить градиентный бустинг (XGBoost) на TF-IDF. Вычислить accuracy и AUC-ROC. 10. Визуализировать 15 самых важных слов для положительных и отрицательных твитов.
7	1. Загрузить датасет «Toxic Comments» (токсичные/нетоксичные). Вывести баланс классов. 2. Очистить текст: удалить пунктуацию, цифры, привести к нижнему регистру. Заменить аббревиатуры (например, "don't" -> "do not"). 3. Токенизировать с помощью nltk.tokenize.word_tokenize. Построить гистограмму распределения длины документов. 4. Удалить стоп-слова. Добавить свои стоп-слова: "im", "ive", "youre", "theyre". 5. Применить стемминг. Сравнить стемминг и лемматизацию для слова "studies". 6. Создать Bag-of-Words с CountVectorizer(max_features=10000, binary=False). Вывести размер матрицы. 7. Преобразовать в TF-IDF. Использовать TfidfVectorizer(sublinear_tf=True, max_df=0.5).

	8. Разделить данные на train/test (70/30) со стратификацией. 9. Обучить логистическую регрессию с подбором C через GridSearchCV. 10. Построить кривые обучения (learning curves) для лучшей модели.
8	1. Загрузить датасет «News Category» (предсказание категории новости). Вывести уникальные категории. 2. Очистить текст: удалить заголовки, числа, пунктуацию. Использовать re.sub. 3. Токенизировать с помощью spaCy. Подсчитать количество токенов в каждом документе. 4. Удалить стоп-слова и короткие слова (< 3 символов). Объяснить, почему короткие слова часто удаляют. 5. Применить лемматизацию через spaCy. Сравнить леммы для "goose", "geese", "gooses". 6. Создать Bag-of-Words с CountVectorizer(max_features=15000, ngram_range=(1,2)). 7. Преобразовать в TF-IDF. Использовать TfidfVectorizer(smooth_idf=True, norm='l2'). 8. Разделить данные на train/test (80/20). 9. Обучить Random Forest на TF-IDF. Вычислить accuracy и weighted F1. 10. Эксперимент: использовать HashingVectorizer вместо CountVectorizer и сравнить время векторизации и качество.
9	1. Загрузить датасет «Amazon Reviews» (оценки 1-5). Преобразовать в бинарную классификацию (оценки 1-2 -> отрицательные, 4-5 -> положительные). 2. Очистить текст: удалить HTML-теги, привести к нижнему регистру, удалить пунктуацию. 3. Токенизировать с помощью nltk.word_tokenize. Сохранить слова, начинающиеся с буквы. 4. Удалить стоп-слова из NLTK. Сравнить размер словаря до и после. 5. Применить лемматизацию с помощью WordNetLemmatizer. Сравнить леммы для "better", "best", "good". 6. Создать Bag-of-Words с CountVectorizer(max_features=20000, min_df=5, max_df=0.8). 7. Преобразовать в TF-IDF. Использовать TfidfTransformer() после CountVectorizer. 8. Разделить данные на train/test (75/25).

	9. Обучить наивный байесовский классификатор. Вычислить precision, recall, F1 для положительного класса. 10. Эксперимент: изменить min_df (1, 3, 5, 10) и построить график зависимости accuracy.
10	1. Загрузить датасет «Spam Assassin» (спам/нормальные письма). Вывести процент писем со спамом. 2. Очистить текст: удалить заголовки писем, номера, пунктуацию. Привести к нижнему регистру. 3. Токенизировать с помощью spaCy. Вывести количество уникальных токенов. 4. Удалить стоп-слова с помощью spaCy. Добавить в стоп-слова слова с частотой > 90% в корпусе. 5. Применить стемминг с помощью PorterStemmer. Сравнить слова "studying", "studied", "studies". 6. Создать Bag-of-Words с CountVectorizer(max_features=5000, binary=False). Вывести размер матрицы. 7. Преобразовать в TF-IDF с TfidfVectorizer(use_idf=True, norm='l1'). Что делает norm='l1'? 8. Разделить данные на train/test (80/20). 9. Обучить логистическую регрессию с L1-регуляризацией (penalty='l1', solver='saga'). Сколько признаков обнулилось? 10. Визуализировать важность признаков (коэффициенты модели) в виде bar chart.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое токенизация? Какие методы токенизации существуют в NLTK и spaCy?
2. Зачем удаляют стоп-слова при обработке текста? Приведите примеры стоп-слов для английского и русского языков.
3. В чём разница между стеммингом и лемматизацией? Когда какой метод предпочтительнее?
4. Что такое Bag-of-Words (BoW)? Как он преобразует текст в числовой вектор? Каковы его недостатки?
5. Что такое TF-IDF? Как вычисляется TF (term frequency) и IDF (inverse document frequency)? Запишите формулу.
6. В чём преимущество TF-IDF перед обычным Bag-of-Words для классификации текстов?
7. Какие параметры CountVectorizer позволяют ограничить размер словаря? Объясните max_features, min_df, max_df.
8. Что такое n-граммы? Зачем использовать биграммы и триграммы вместо униграмм?
9. Как оценить качество модели классификации текстов? Какие метрики используют при дисбалансе классов?
10. Почему важно применять векторизатор (CountVectorizer/TfidfVectorizer) только на обучающих данных, а затем преобразовывать тестовые?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Качество предобработки текста. Правильное применение токенизации, удаления стоп-слов, стемминга/лемматизации.	10
Корректность векторизации (BoW, TF-IDF). Правильный выбор параметров, понимание различий.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов,	10

комментарии, чистота кода.	
Выводы и интерпретация. Анализ влияния предобработки и векторизации, сравнение методов.	10
Визуализации (графики, confusion matrix). Качество, подписи, читаемость.	10

Лабораторная работа № 32

Word Embeddings и классификация текстов

Цель: Использовать векторные представления слов для классификации текстов.

Теоретическое обоснование

Word Embeddings — плотные векторные представления слов, где семантически близкие слова расположены близко в векторном пространстве. В отличие от разреженных BoW/TF-IDF, эмбеддинги имеют размерность 100–300 и кодируют смысловые отношения (например, король — мужчина + женщина ≈ королева).

Предобученные эмбеддинги.

- **Word2Vec** (Google) — использует архитектуры CBOW (предсказание слова по контексту) и Skip-gram (предсказание контекста по слову). Обучается на больших корпусах (например, Google News).
- **GloVe** (Stanford) — основан на глобальной матрице совместных встречаемостей слов.
- **FastText** (Facebook) — учитывает подсловные n-граммы, что позволяет обрабатывать редкие и незнакомые слова.

Embedding слой в Keras. Слой Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len) создаёт матрицу весов (словарь × эмбеддинги). Во время обучения эта матрица может быть:

- **Зафиксирована** (trainable=False) — используются предобученные векторы, не обновляются.
- **Дообучаема** (trainable=True) — векторы настраиваются под конкретную задачу.

Архитектура для классификации текстов. После Embedding слоя следуют:

- GlobalAveragePooling1D или GlobalMaxPooling1D — усреднение/максимум по временной оси (токенам). Это преобразует последовательность переменной длины в вектор фиксированной размерности.
- Затем один или несколько Dense слоев и выходной слой (softmax/sigmoid).

Рекуррентные слои (SimpleRNN, LSTM, GRU). В отличие от пулинга, они сохраняют порядок слов:

- **SimpleRNN** — базовая рекуррентная ячейка, страдает от исчезающего/взрывающегося градиента.
- **LSTM (Long Short-Term Memory)** — имеет механизмы забывания, входа и выхода, позволяет запоминать долгосрочные зависимости.
- **GRU (Gated Recurrent Unit)** — упрощённая версия LSTM, быстрее, с меньшим числом параметров.

Для классификации обычно используют LSTM/GRU, а затем глобальный пулинг или последний выход.

Сравнение с TF-IDF. Word Embeddings + LSTM/GRU учитывают контекст и порядок слов, что позволяет различать, например, «не очень хороший» и «очень не хороший». Однако требуют больше данных и вычислительных ресурсов. На больших датасетах эмбеддинги превосходят TF-IDF.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Загрузить датасет IMDB для анализа тональности через keras.datasets.imdb.load_data(num_words=10000). 2. Выровнять последовательности до одинаковой длины с помощью pad_sequences(maxlen=200). 3. Построить модель: Embedding(10000, 128, input_length=200) → GlobalAveragePooling1D() → Dense(1, activation='sigmoid'). 4. Скомпилировать модель с adam, функцией потерь binary_crossentropy, метрикой accuracy. 5. Обучить модель на 5 эпох, оценить accuracy на тесте.

	6. Загрузить предобученные эмбеддинги GloVe (например, glove.6B.100d.txt) для 10000 слов. 7. Создать матрицу весов Embedding слоя и инициализировать ею слой (установить trainable=False). 8. Повторить обучение с замороженными эмбеддингами, сравнить accuracy. 9. Добавить слой SimpleRNN(64) вместо GlobalAveragePooling1D и обучить с GloVe. 10. Сохранить лучшую модель и построить confusion matrix.
2	1. Загрузить датасет IMDB, ограничить словарь 5000 слов, максимальная длина последовательности 100. 2. Построить модель: Embedding(5000, 64) → LSTM(64) → Dense(1, 'sigmoid'). 3. Обучить модель на 3 эпохи (из-за LSTM долго). 4. Использовать Embedding с trainable=False и случайными весами. 5. Загрузить предобученные эмбеддинги FastText (или GloVe), повторить обучение. 6. Сравнить accuracy и время обучения для LSTM и простой модели с GlobalAveragePooling. 7. Добавить второй LSTM слой (return_sequences=True) и сравнить. 8. Визуализировать t-SNE проекцию первых 500 эмбеддингов слов. 9. Эксперимент: изменить максимальную длину последовательности на 300 и сравнить. 10. Сохранить модель в формате .h5.
3	1. Загрузить датасет новостей (20 Newsgroups) с 3 категориями. 2. Токенизировать тексты с помощью Tokenizer(num_words=10000), преобразовать в последовательности, выровнять. 3. Построить модель с Embedding → GRU(128) → Dense(3, 'softmax'). 4. Обучить модель на 5 эпох, оценить accuracy. 5. Загрузить предобученные эмбеддинги Word2Vec (Google News) для 10000 слов. 6. Инициализировать Embedding слой, заморозить. 7. Обучить заново, сравнить качество. 8. Разморозить эмбеддинги (trainable=True) и дообучить на 2 эпохи. 9. Построить confusion matrix и classification report. 10. Сделать вывод о влиянии дообучения эмбеддингов.
4	1. Загрузить датасет SMS Spam Collection. Преобразовать тексты в последовательности. 2. Построить модель с Embedding(8000, 50) → Bidirectional(LSTM(64)) → Dense(1, 'sigmoid').

	3. Обучить модель на 10 эпох, зафиксировать accuracy и AUC-ROC. 4. Заменить LSTM на GRU и сравнить качество. 5. Использовать предобученные эмбеддинги GloVe (50d). 6. Добавить слой Dropout(0.5) после LSTM. 7. Сравнить модель с предобученными эмбеддингами и модель со случайными. 8. Визуализировать кривые обучения. 9. Построить ROC-кривую для лучшей модели. 10. Сохранить модель и токенизатор.
5	1. Загрузить датасет sst2 (Stanford Sentiment Treebank) через datasets. 2. Выполнить токенизацию и выравнивание последовательностей. 3. Построить модель с Embedding → GlobalMaxPooling1D() → Dense(128, 'relu') → Dense(1, 'sigmoid'). 4. Обучить модель на 5 эпох. 5. Использовать предобученные эмбеддинги FastText (300d). 6. Применить слой BatchNormalization после pooling. 7. Сравнить результаты с TF-IDF + логистической регрессией. 8. Визуализировать 10 самых близких слов к слову "good" в векторном пространстве (по косинусному расстоянию). 9. Эксперимент: изменить размер эмбеддингов (50, 100, 300) и сравнить accuracy. 10. Сохранить таблицу сравнения.
6	1. Загрузить датасет «Toxic Comments» (бинарная классификация). 2. Токенизировать, ограничить словарь 20000 слов, максимальная длина 150. 3. Построить модель: Embedding → Conv1D(128, 5, activation='relu') → GlobalMaxPooling1D → Dense(1, 'sigmoid'). 4. Обучить модель на 10 эпох. 5. Загрузить предобученные эмбеддинги GloVe (100d). 6. Добавить второй сверточный слой с kernel_size=3. 7. Сравнить CNN-модель с LSTM (оба с GloVe). 8. Построить confusion matrix. 9. Проанализировать самые частые ошибки модели (вывести примеры). 10. Сохранить модель.
7	1. Загрузить русскоязычный датасет отзывов. 2. Токенизировать с помощью Tokenizer (словарь 15000). 3. Загрузить предобученные русские эмбеддинги (например, ruwikiruscorpora или RusVectōrēs). 4. Создать модель с Embedding → LSTM(128) → Dense(1, 'sigmoid').

	5. Обучить с замороженными эмбеддингами. 6. Обучить такую же модель со случайными эмбеддингами. 7. Сравнить accuracy и время обучения. 8. Добавить слой Dropout(0.3) и RecurrentDropout(0.3) в LSTM. 9. Визуализировать t-SNE для 1000 слов. 10. Сохранить модель.
8	1. Загрузить датасет ag_news (4 категории новостей). 2. Токенизировать, максимальная длина 80. 3. Построить модель: Embedding → Bidirectional(GRU(64)) → GlobalAveragePooling1D → Dense(4, 'softmax'). 4. Обучить с предобученными эмбеддингами GloVe. 5. Эксперимент: изменить тип рекуррентного слоя (SimpleRNN, LSTM, GRU) и сравнить. 6. Добавить Attention слой (самописный) после GRU. 7. Сравнить с моделью без внимания. 8. Построить confusion matrix. 9. Вывести accuracy для каждой категории отдельно. 10. Сохранить лучшую модель.
9	1. Загрузить датасет yelp_reviews (бинарная классификация). 2. Токенизировать, максимальная длина 200. 3. Построить модель с Embedding → SpatialDropout1D(0.2) → LSTM(64, return_sequences=True) → LSTM(32) → Dense(1, 'sigmoid'). 4. Обучить модель на 5 эпох с валидацией. 5. Загрузить предобученные эмбеддинги FastText. 6. Применить GlobalMaxPooling1D вместо второго LSTM – сравнить. 7. Визуализировать кривые обучения. 8. Построить ROC-кривую. 9. Эксперимент: увеличить dropout до 0.5 и сравнить переобучение. 10. Сохранить модель и токенизатор.
10	1. Загрузить датасет dbpedia (14 классов). Использовать подвыборку 5000 образцов. 2. Токенизировать, максимальная длина 150. 3. Построить модель: Embedding → Conv1D(128, 3, activation='relu') → GlobalMaxPooling1D → Dense(128, 'relu') → Dropout(0.5) → Dense(14, 'softmax'). 4. Обучить модель с предобученными эмбеддингами (GloVe). 5. Обучить модель без предобученных эмбеддингов (случайные). 6. Сравнить accuracy и macro F1.

	7. Добавить второй сверточный слой (Conv1D(64, 5)) и сравнить. 8. Построить confusion matrix (можно уменьшенную для топ-5 классов). 9. Эксперимент: изменить размер ядра с 3 на 5 – сравнить. 10. Сохранить лучшую модель.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое word embedding? В чём отличие от one-hot векторов и от Bag-of-Words?
2. Какие предобученные эмбеддинги наиболее популярны (Word2Vec, GloVe, FastText)? Их особенности.
3. Как работает слой Embedding в Keras? Какие параметры он принимает?
4. Как загрузить предобученные эмбеддинги в слой Embedding? Что нужно сделать с весами?
5. Когда следует замораживать эмбеддинги (trainable=False), а когда дообучать?
6. В чём разница между GlobalAveragePooling1D и GlobalMaxPooling1D при обработке последовательностей эмбеддингов?
7. Что такое рекуррентные слои (RNN, LSTM, GRU)? В чём их преимущество перед свёрточными для текста?
8. Зачем использовать Bidirectional обёртку для LSTM/GRU?

9. Как оценить качество модели классификации текстов? Какие метрики важны для многоклассовой задачи?
10. Как визуализировать эмбеддинги слов с помощью t-SNE? Что можно увидеть на такой визуализации?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность использования Embedding слоя. Правильная инициализация предобученными весами, понимание параметров.	10
Выбор и настройка рекуррентной архитектуры. Обоснование выбора RNN/LSTM/GRU, корректное применение Bidirectional.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ сравнения моделей, влияние предобученных эмбеддингов.	10
Визуализации (кривые обучения, confusion matrix, t-SNE). Качество, подписи, читаемость.	10

Лабораторная работа № 33

Трансформеры с Hugging Face

Цель: Использовать предобученные трансформеры для задач NLP.

Теоретическое обоснование

Трансформеры — архитектура, основанная на механизме **самовнимания** (self-attention), которая стала стандартом в обработке естественного языка (NLP). В отличие от рекуррентных сетей, трансформеры обрабатывают всю последовательность параллельно, что позволяет эффективно обучать очень глубокие модели на огромных корпусах.

Ключевые понятия:

- **Self-attention** — каждый токен в последовательности вычисляет веса взаимодействия со всеми другими токенами. Результат — взвешенная сумма представлений всех токенов. Это позволяет модели улавливать контекстные зависимости любой длины.
- **Позиционное кодирование** — поскольку self-attention не имеет понятия порядка, к входным эмбеддингам добавляются синусоидальные коды, кодирующие позицию токена.
- **Многоголовое внимание** (Multi-Head Attention) — несколько механизмов внимания параллельно, каждый захватывает разные типы взаимосвязей.

Предобученные модели трансформеров.

- **BERT** (Bidirectional Encoder Representations from Transformers) — обучается на задачах маскирования слов (MLM) и предсказания следующего предложения (NSP). Понимает контекст с обеих сторон.
- **RoBERTa** — улучшенный BERT (больше данных, удалена задача NSP, большие батчи).
- **DistilBERT** — облегчённая версия BERT (в 2 раза быстрее, на 40% меньше параметров, 97% точности).
- **GPT** (Generative Pre-trained Transformer) — авторегрессионный, предсказывает следующий токен (хорош для генерации).

Hugging Face Transformers — библиотека с единым интерфейсом для сотен предобученных моделей. Основные компоненты:

- AutoTokenizer — преобразует текст в входные ID, attention mask, типы токенов.

- AutoModelForSequenceClassification — модель с классификационной головой.
- pipeline — высокоуровневый API для распространённых задач (классификация, NER, вопрос-ответ).

Fine-tuning с Trainer API. Для дообучения на своём датасете:

1. Загружаем токенизатор и модель.
2. Подготавливаем Dataset (токенизируем текст).
3. Определяем аргументы обучения (TrainingArguments): число эпох, размер батча, скорость обучения.
4. Запускаем Trainer.

Преимущества трансформеров:

- Учитывают контекст (в отличие от Bag-of-Words).
- Обучаются на огромных корпусах, затем дообучаются на малых датасетах (transfer learning).
- Достигают SOTA на большинстве NLP задач.

Недостатки: большие требования к памяти и вычислительным ресурсам, сложность интерпретации.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	**Задание**
1	1. Установить библиотеки transformers, datasets, torch. 2. Загрузить датасет IMDB для анализа тональности через load_dataset('imdb'). Вывести размер обучающей и тестовой выборок. 3. Загрузить токенизатор distilbert-base-uncased с помощью AutoTokenizer.from_pretrained(). 4. Написать функцию токенизации, которая принимает текст и возвращает input_ids и attention_mask. Применить её к датасету через .map(). 5. Загрузить модель DistilBertForSequenceClassification с 2 метками с помощью AutoModelForSequenceClassification.from_pretrained(). 6. Создать TrainingArguments с

	параметрами: output_dir='./results', evaluation_strategy='epoch', num_train_epochs=2, per_device_train_batch_size=16, save_total_limit=2. 7. Инициализировать Trainer с моделью, аргументами, токенизатором, обучающим и валидационным датасетами. 8. Обучить модель через trainer.train(). 9. Оценить модель на тестовой выборке через trainer.evaluate(). 10. Использовать pipeline('sentiment-analysis') с загруженной моделью для классификации нового текста («This movie was amazing!»).
2	1. Загрузить датасет ag_news (4 категории новостей) через load_dataset('ag_news'). 2. Загрузить токенизатор bert-base-uncased. 3. Токенизировать датасет с padding='max_length', truncation=True, max_length=128. 4. Загрузить модель BertForSequenceClassification с num_labels=4. 5. Настроить TrainingArguments с learning_rate=2e-5, per_device_train_batch_size=32, num_train_epochs=3. 6. Обучить модель, добавив compute_metrics для вычисления accuracy. 7. Сохранить модель и токенизатор в папку ./my_bert_model. 8. Загрузить сохранённую модель через AutoModelForSequenceClassification.from_pretrained(). 9. Создать pipeline для классификации текста на основе загруженной модели. 10. Протестировать на 5 случайных предложениях из тестовой выборки, вывести истинную и предсказанную метки.
3	1. Загрузить датасет sst2 (Stanford Sentiment Treebank) через load_dataset('glue', 'sst2'). 2. Загрузить токенизатор roberta-base. 3. Токенизировать данные, добавив truncation=True, padding=True. 4. Загрузить модель RobertaForSequenceClassification с 2 метками. 5. Использовать DataCollatorWithPadding для динамического выравнивания последовательностей. 6. Обучить модель на 3 эпохах с evaluation_strategy='steps', eval_steps=500. 7. В процессе обучения логировать loss и accuracy. 8. Оценить модель на тестовой выборке, вывести F1-score. 9. Эксперимент: изменить learning_rate на 3e-5 и сравнить результат с 2e-5. 10. Сохранить лучшую модель на основе валидационной accuracy.
4	1. Загрузить датасет twitter_complaints (классификация жалоб) через load_dataset('twitter_complaints'). 2. Загрузить токенизатор albert-base-v2.

	3. Выполнить токенизацию, добавив return_tensors='pt'. 4. Загрузить модель AlbertForSequenceClassification с 2 метками. 5. Создать TrainingArguments с weight_decay=0.01, warmup_steps=500. 6. Обучить модель на 5 эпохах с использованием Trainer. 7. Построить confusion matrix для предсказаний на тестовой выборке. 8. Эксперимент: применить EarlyStoppingCallback(early_stopping_patience=2). 9. Сравнить результат с моделью без ранней остановки. 10. Загрузить модель на Hugging Face Hub (в приватный репозиторий) с помощью model.push_to_hub().
5	1. Загрузить датасет emotion (6 классов эмоций) через load_dataset('emotion'). 2. Загрузить токенизатор distilroberta-base. 3. Токенизировать датасет с max_length=64. 4. Загрузить модель DistilRobertaForSequenceClassification с 6 метками. 5. Создать TrainingArguments с save_strategy='epoch', load_best_model_at_end=True, metric_for_best_model='f1'. 6. Определить функцию compute_metrics, возвращающую accuracy и f1. 7. Обучить модель на 4 эпохах. 8. Построить график изменения loss и accuracy по эпохам. 9. Эксперимент: заменить модель на roberta-base и сравнить точность и время обучения. 10. Сохранить модель в формате ONNX через transformers.onnx.export().
6	1. Загрузить русскоязычный датасет (например, ru_sentiment из Hugging Face Hub). 2. Загрузить токенизатор DeepPavlov/rubert-base-cased. 3. Токенизировать датасет с padding=True, truncation=True. 4. Загрузить модель AutoModelForSequenceClassification для русского языка. 5. Настроить TrainingArguments с logging_dir='./logs', logging_steps=100. 6. Обучить модель на 3 эпохах. 7. Использовать TensorBoard для визуализации метрик (создать коллбэк). 8. Оценить модель на тестовой выборке, вывести classification report. 9. Создать pipeline('sentiment-analysis') для русского языка на основе обученной модели. 10. Протестировать на 5 русских фразах разной тональности.
7	1. Загрузить датасет financial_phrasebank (3 класса тональности финансовых новостей).

	2. Загрузить токенизатор finbert из модели ProsusAI/finbert. 3. Токенизировать датасет, добавив return_attention_mask=True. 4. Загрузить модель BertForSequenceClassification с 3 метками, используя предобученные веса ProsusAI/finbert. 5. Заморозить нижние 6 слоёв BERT (for param in model.bert.encoder.layer[:6].parameters(): param.requires_grad = False). 6. Обучить только верхние слои на 5 эпохах с learning_rate=3e-5. 7. Сравнить качество с моделью, обученной без заморозки. 8. Построить confusion matrix для лучшей модели. 9. Эксперимент: разморозить все слои и выполнить fine-tuning с маленьким lr=1e-5 на 3 эпохи. 10. Сохранить модель и токенизатор в одну папку.
8	1. Загрузить датасет dair-ai/emotion с 6 классами эмоций. 2. Использовать train_test_split для создания валидационной выборки (10% от train). 3. Загрузить токенизатор microsoft/deberta-v3-base. 4. Токенизировать датасет, добавив truncation=True, max_length=128. 5. Загрузить модель DebertaForSequenceClassification с 6 метками. 6. Создать TrainingArguments с eval_strategy='epoch', save_strategy='epoch', metric_for_best_model='f1'. 7. Обучить модель на 4 эпохах. 8. Построить график зависимости f1 от эпохи. 9. Эксперимент: добавить параметр fp16=True в TrainingArguments и сравнить скорость обучения. 10. Сохранить лучшую модель.
9	1. Загрузить датасет amazon_polarity (2 класса). Использовать подвыборку 5000 образцов для ускорения. 2. Загрузить токенизатор google/electra-small-discriminator. 3. Токенизировать датасет с padding='max_length', max_length=256. 4. Загрузить модель ElectraForSequenceClassification с 2 метками. 5. Создать TrainingArguments с per_device_train_batch_size=32, per_device_eval_batch_size=64. 6. Обучить модель на 2 эпохах. 7. Использовать trainer.predict() для получения предсказаний на тестовой выборке. 8. Построить ROC-кривую и вычислить AUC-ROC. 9. Эксперимент: заменить модель на google/electra-base-discriminator и сравнить точность. 10. Сохранить модель в формате PyTorch.
10	1. Загрузить датасет yahoo_answers (10 категорий). Использовать подвыборку 2000 образцов.

	2. Загрузить токенизатор xlm-roberta-base. 3. Токенизировать датасет с truncation=True, max_length=200. 4. Загрузить модель XLMRobertaForSequenceClassification с 10 метками. 5. Создать TrainingArguments с report_to='wandb' (для логирования в Weights & Biases). 6. Обучить модель на 3 эпохах. 7. Визуализировать матрицу ошибок (confusion matrix) для всех 10 классов. 8. Проанализировать, какие классы чаще всего путаются. 9. Эксперимент: добавить label_smoothing_factor=0.1 и сравнить качество. 10. Загрузить модель на Hugging Face Hub в публичный репозиторий.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое библиотека Hugging Face Transformers? Какие основные классы она предоставляет?
2. Что делает AutoTokenizer.from_pretrained()? Почему важно использовать автоматические классы?
3. Что такое Trainer API? Какие преимущества он даёт при fine-tuning моделей?
4. Как загрузить датасет с помощью библиотеки datasets? Как применить токенизацию ко всему датасету с помощью .map()?

5. Что такое TrainingArguments? Какие основные параметры вы знаете? Объясните evaluation_strategy, save_strategy, logging_steps.
6. Как загрузить предобученную модель для классификации текста? Что делает параметр num_labels?
7. Как использовать pipeline для инференса с загруженной моделью? Приведите пример для анализа тональности.
8. В чём разница между BERT, DistilBERT и RoBERTa? Каковы их преимущества и недостатки?
9. Как сохранить и загрузить fine-tuned модель? Какие файлы создаются?
10. Что такое DataCollatorWithPadding? Зачем он нужен при работе с последовательностями разной длины?

Критерии оценивания

Параметр	**Баллы**
Выполнение индивидуального задания (10 пунктов).	40
Корректность загрузки и использования токенизатора. Правильное применение AutoTokenizer, обработка последовательностей.	10
Правильная настройка Trainer и TrainingArguments. Обоснованный выбор параметров, корректное обучение и оценка.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ качества, сравнение моделей (если было), обсуждение результатов.	10
Визуализации (графики loss/accuracy, confusion matrix). Качество, подписи, читаемость.	10

Лабораторная работа № 34

Интерпретация моделей с LIME и SHAP

Цель: Освоить методы объяснимого ИИ для интерпретации предсказаний модели.

Теоретическое обоснование

Сложные модели (случайный лес, XGBoost, нейронные сети) часто работают как «чёрный ящик» — трудно понять, почему принято то или иное решение. Методы объяснимого ИИ (XAI — Explainable Artificial Intelligence) помогают интерпретировать предсказания.

LIME (Local Interpretable Model-agnostic Explanations) — локальный метод, объясняющий отдельное предсказание. Принцип работы:

1. Вокруг объясняемого объекта создаётся множество синтетических примеров путём небольших изменений признаков.
2. Для этих примеров получаем предсказания сложной модели.
3. Обучаем простую, интерпретируемую модель (линейную регрессию или дерево решений) на этих синтетических данных, взвешивая примеры по близости к исходному объекту.
4. Веса признаков в простой модели показывают, какие признаки больше всего повлияли на предсказание.

LIME хорош для локального объяснения, но результаты могут быть нестабильны и зависеть от параметров.

SHAP (SHapley Additive exPlanations) — основан на теории кооперативных игр (значения Шепли). Значение Шепли для признака i — это средний вклад этого признака в предсказание по всем возможным коалициям признаков.

SHAP values обладают свойствами:

- **Аддитивность** — сумма вкладов всех признаков плюс базовое значение даёт предсказание.
- **Консистентность** — если модель меняется так, что вклад признака увеличивается, его SHAP value не уменьшится.

Визуализации SHAP:

- **Summary plot** — для всех объектов и всех признаков показывает

распределение SHAP values. Каждый признак — горизонтальная полоса, цвет отражает величину признака. Позволяет увидеть, какие признаки в целом важны и как их значения влияют на предсказание (положительно или отрицательно).

- **Dependence plot** — для одного признака показывает зависимость SHAP value от значения признака. Может выявить нелинейные эффекты.
- **Waterfall plot** — для одного объекта показывает, как каждый признак толкает предсказание от базового значения к финальному.

Применение LIME и SHAP. Используются для доверия к модели, обнаружения смещений (bias), отладки неверных предсказаний, соответствия регуляторным требованиям (например, «право на объяснение» в GDPR).

Ограничения: SHAP может быть медленным для больших данных, требует калибровки.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Обучить случайный лес (RandomForestClassifier) на датасете Iris. Разделить на train/test (80/20). 2. Установить библиотеку lime. Создать LimeTabularExplainer на обучающих данных (используя mode='classification'). 3. Для одного тестового примера объяснить предсказание с помощью explainer.explain_instance(). 4. Визуализировать объяснение в виде bar plot (сохранить скриншот). 5. Установить библиотеку shap. Вычислить SHAP values с помощью shap.TreeExplainer для случайного леса. 6. Построить summary plot для всех признаков (глобальная интерпретация). 7. Построить dependence plot для самого важного признака. 8. Для того же тестового примера, что и в LIME, визуализировать waterfall plot. 9. Сравнить объяснения LIME и SHAP (совпадают ли важные признаки, одинаков ли знак влияния). 10. Сделать вывод о согласованности двух методов.
2	1. Обучить XGBoost классификатор на датасете Breast Cancer.

	2. Использовать shap.TreeExplainer для XGBoost. 3. Построить summary plot (bar plot) – глобальная важность признаков по SHAP. 4. Построить beeswarm plot (shap.plots.beeswarm). 5. Использовать LIME для одного тестового примера (с наибольшей ошибкой). 6. Визуализировать LIME объяснение в виде таблицы вкладов признаков. 7. Построить SHAP waterfall plot для того же примера. 8. Сравнить, какие признаки в LIME и SHAP считаются положительными/отрицательными. 9. Эксперимент: объяснить пример, где модель ошибается. Проанализировать причины. 10. Сохранить SHAP объяснения в файл (JSON).
3	1. Обучить логистическую регрессию на датасете Titanic (предварительно закодировав категориальные признаки). 2. Использовать shap.LinearExplainer для линейной модели. 3. Построить summary plot. 4. Сравнить SHAP важность с абсолютными значениями коэффициентов модели. 5. Использовать LIME (LimeTabularExplainer) для объяснения предсказания для пассажира, который выжил, но модель предсказала смерть. 6. Визуализировать LIME объяснение. 7. Построить SHAP waterfall plot для того же пассажира. 8. Определить, какие признаки «перевесили» правильное предсказание. 9. Эксперимент: изменить значения признаков (например, возраст) и посмотреть, как изменится объяснение SHAP. 10. Сделать вывод о полезности LIME и SHAP для диагностики ошибок.
4	1. Обучить нейронную сеть (Keras) на датасете MNIST (несколько классов). 2. Использовать shap.DeepExplainer для модели. 3. Для одного тестового изображения (цифра 3) вычислить SHAP values для каждого пикселя. 4. Визуализировать SHAP значения в виде тепловой карты, наложенной на изображение. 5. Использовать LIME для изображений (lime.lime_image.LimeImageExplainer). 6. Объяснить предсказание для того же изображения,

	визуализировать суперпиксели, которые влияют на предсказание. 7. Сравнить визуализации SHAP и LIME – какие области изображения считаются важными. 8. Эксперимент: взять изображение, где модель ошибается (например, 5 распознала как 3). Проанализировать, какие пиксели ввели в заблуждение. 9. Сохранить визуализации SHAP и LIME. 10. Сделать вывод о применимости методов для изображений.
5	1. Обучить случайный лес на датасете California Housing (регрессия). 2. Использовать shap.TreeExplainer для регрессии. 3. Построить summary plot для всех признаков. 4. Построить dependence plot для признака MedInc (медианный доход). Проанализировать нелинейность. 5. Использовать LIME для регрессии (LimeTabularExplainer(mode='regression')). 6. Для одного тестового дома объяснить предсказание (цену) с помощью LIME. 7. Сравнить SHAP и LIME: какой признак самый важный по SHAP и по LIME для этого дома. 8. Построить SHAP waterfall plot для этого дома. 9. Эксперимент: изменить значение MedInc на большее и посмотреть на изменение SHAP. 10. Сохранить SHAP values для всех тестовых объектов в CSV.
6	1. Обучить XGBoost классификатор на датасете Adult Income (предсказание дохода >50K). 2. Использовать shap.TreeExplainer. 3. Построить глобальную важность признаков (summary plot). 4. Определить топ-3 признака, увеличивающих шанс высокого дохода. 5. Использовать shap.force_plot для одного примера (визуализация в виде стрелок). 6. Для того же примера построить LIME объяснение. 7. Сравнить, какой метод даёт более интуитивно понятное объяснение. 8. Эксперимент: взять пример с доходом <50K, который модель предсказала как >50K. Найти признак, который ввёл в заблуждение. 9. Построить групповой force plot для 20 случайных примеров. 10. Сохранить SHAP explainer в файл.

7	1. Обучить градиентный бустинг (LightGBM) на датасете Credit Card Fraud (дисбаланс классов). 2. Использовать shap.TreeExplainer. 3. Построить summary plot. 4. Вычислить SHAP values для мошеннической транзакции. 5. Проанализировать, какие признаки указывают на мошенничество. 6. Использовать LIME для объяснения той же транзакции. 7. Сравнить объяснения – совпадают ли признаки. 8. Эксперимент: применить shap.force_plot для мошеннической и обычной транзакции. 9. Построить dependence plot для признака V14 (анонимный). 10. Сделать вывод о том, какие методы лучше подходят для выявления аномалий.
8	1. Обучить случайный лес на датасете Heart Disease (бинарная классификация). 2. Использовать shap.TreeExplainer. 3. Построить waterfall plot для пациента с высоким риском. 4. Использовать LIME для того же пациента. 5. Сравнить визуализации и сделать вывод о том, какой метод более детальный. 6. Эксперимент: изменить возраст пациента с 60 на 30 и посмотреть на изменение SHAP. 7. Построить глобальный summary plot и сравнить с важностью признаков из модели (feature_importances_). 8. Определить, какой признак больше всего влияет на риск (по SHAP). 9. Сохранить SHAP значения для всех пациентов. 10. Сделать вывод о практической ценности SHAP для медицинской диагностики.
9	1. Обучить модель на текстовых данных (IMDB) с использованием TF-IDF и логистической регрессии. 2. Использовать LIME для текстов (lime.lime_text.LimeTextExplainer). 3. Для одного отзыва объяснить, почему он классифицирован как положительный. 4. Визуализировать объяснение (слова, увеличивающие и уменьшающие вероятность). 5. Использовать SHAP для текстов – сложнее, но можно применить shap.Explainer с функцией предсказания. 6. Сравнить объяснения LIME и SHAP (если SHAP применим).

	7. Эксперимент: изменить отзыв, удалив положительные слова, и посмотреть на изменение предсказания и объяснения. 8. Визуализировать топ-10 слов, влияющих на предсказание. 9. Сохранить объяснение LIME в HTML файл (explainer.save_to_file()). 10. Сделать вывод о применимости LIME для текстов.
10	1. Обучить модель на датасете Iris, но с добавлением шумовых признаков (случайные числа). 2. Использовать SHAP (TreeExplainer) для случайного леса. 3. Построить summary plot – убедиться, что шумовые признаки имеют низкую важность. 4. Построить dependence plot для одного из шумовых признаков – значение SHAP должно быть около 0. 5. Использовать LIME для одного тестового примера – проверить, что шумовые признаки не влияют на объяснение. 6. Эксперимент: увеличить масштаб шумового признака (умножить на 100) – изменится ли его SHAP важность? 7. Построить waterfall plot для примера, где модель ошибается из-за шума. 8. Сравнить стабильность LIME и SHAP при добавлении шума. 9. Сохранить SHAP explainer. 10. Сделать вывод о робастности методов к шумовым признакам.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое объяснимый ИИ (XAI) и почему он важен для сложных моделей машинного обучения?
2. Как работает LIME? Опишите основные шаги алгоритма (локальная аппроксимация).
3. Что такое SHAP? На чём основаны значения Шепли из теории кооперативных игр?
4. В чём разница между локальным и глобальным объяснением модели?
5. Какие визуализации SHAP вы знаете? Что показывает summary plot, dependence plot, waterfall plot?
6. Какие эксплейнеры существуют в SHAP для разных типов моделей (TreeExplainer, LinearExplainer, DeepExplainer)?
7. В чём преимущества SHAP перед LIME? Каковы недостатки SHAP?
8. Можно ли использовать SHAP для регрессионных задач? Если да, то как интерпретировать значения?
9. Как LIME объясняет предсказания для текстовых данных? Как представляется «локальная окрестность» для текста?
10. Какие проблемы могут возникнуть при интерпретации моделей с помощью LIME и SHAP (стабильность, вычислительная сложность, корректность)?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность использования LIME. Правильное создание эксплейнера, выбор режима, визуализация.	10
Корректность использования SHAP. Правильный выбор эксплейнера для типа модели, вычисление SHAP values, построение графиков.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10

Выводы и интерпретация. Сравнение LIME и SHAP, анализ результатов, обоснование выводов.	10
Визуализации (LIME bar plot, SHAP summary/dependence/waterfall). Качество, подписи, читаемость.	10

Лабораторная работа № 35

Развертывание модели с REST API (Flask/FastAPI)

Цель: Разработать REST API для serving обученной модели.

Теоретическое обоснование

После обучения модели её необходимо интегрировать в реальное приложение. Наиболее распространённый способ — развернуть модель как веб-сервис с REST API (Representational State Transfer Application Programming Interface). Клиент отправляет HTTP-запрос (обычно POST) с данными, а сервер возвращает предсказание в формате JSON.

Сохранение модели. Модель должна быть сохранена на диск после обучения:

- Для sklearn — joblib.dump(model, 'model.pkl') или pickle.dump().
- Для XGBoost/LightGBM — model.save_model('model.json').
- Для Keras/TensorFlow — model.save('model.h5') или model.save('saved_model/').
- ONNX (Open Neural Network Exchange) — универсальный формат для обмена моделями между фреймворками.

Flask — микрофреймворк для создания веб-приложений на Python. Простой, лёгкий, хорошо подходит для небольших сервисов. Структура:

python

```python
from flask import Flask, request, jsonify
app = Flask(__name__)
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    # предобработка, предсказание
    return jsonify({'prediction': pred})
```

FastAPI — современный фреймворк, асинхронный, с автоматической генерацией документации (Swagger), валидацией типов (Pydantic), более высокая производительность. Структура:

python

```python
from fastapi import FastAPI
from pydantic import BaseModel
app = FastAPI()
class InputData(BaseModel):
    feature1: float
    feature2: int
@app.post('/predict')
def predict(data: InputData):
    # предсказание
    return {'prediction': pred}
```

return {'prediction': pred} **Тестирование API.** Используются инструменты:

- **Postman** — графический клиент для отправки HTTP-запросов, проверки ответов, автоматизации тестов.
- **curl** — командная строка: curl -X POST -H "Content-Type: application/json" -d '{"feat":1.2}' http://localhost:5000/predict.

Контейнеризация с Docker. Docker упаковывает приложение со всеми зависимостями в контейнер, что обеспечивает воспроизводимость и упрощает деплой. Создаётся Dockerfile, описывающий окружение (Python, установка пакетов, копирование кода, запуск сервера). Контейнер можно развернуть на любом сервере с поддержкой Docker.

Безопасность. В production необходимо добавить аутентификацию (API-ключи), ограничение частоты запросов (rate limiting), логирование, обработку ошибок.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Обучить логистическую регрессию на датасете Iris и сохранить модель с помощью joblib.dump(). 2. Создать Flask-приложение с эндпоинтом /health (GET), возвращающим {"status": "ok"}. 3. Создать эндпоинт /predict (POST), принимающий JSON с признаками (sepal_length, sepal_width, petal_length, petal_width). 4. Реализовать загрузку модели при старте приложения (глобальная переменная).

	5. В эндпоинте /predict получить данные из запроса, преобразовать в массив, выполнить предсказание. 6. Вернуть ответ в формате {"prediction": int, "probabilities": list}. 7. Добавить обработку ошибок (неверный формат JSON, отсутствие полей) с возвратом HTTP 400. 8. Запустить сервер локально на порту 5000. 9. Протестировать эндпоинт с помощью curl или Postman (привести пример запроса и ответа). 10. Написать Dockerfile для приложения, собрать образ и запустить контейнер, проверить работоспособность.
2	1. Обучить модель случайного леса на датасете Boston Housing (регрессия), сохранить в .pkl. 2. Создать FastAPI-приложение. 3. Определить Pydantic-модель InputData с полями для всех признаков (с валидацией типов). 4. Создать эндпоинт /predict (POST), принимающий InputData. 5. Загрузить модель при старте (использовать lru_cache или глобальную переменную). 6. Выполнить предсказание и вернуть {"prediction": float}. 7. Добавить эндпоинт /ping (GET) для проверки доступности. 8. Автоматически сгенерировать документацию Swagger (FastAPI предоставляет /docs). 9. Протестировать API через Swagger UI. 10. Сохранить Dockerfile и requirements.txt, запустить контейнер, проверить эндпоинты.
3	1. Обучить XGBoost классификатор на датасете Titanic (предварительно обработав данные). Сохранить модель в формате JSON (model.save_model()). 2. Создать Flask-приложение. 3. Реализовать эндпоинт /predict, принимающий JSON с признаками пассажира (pclass, sex, age, sibsp, parch, fare, embarked). 4. Выполнить предобработку входных данных (one-hot кодирование) внутри эндпоинта. 5. Вернуть предсказание (0/1) и вероятность. 6. Добавить логирование запросов (дата, IP, входные данные, предсказание) в файл. 7. Создать эндпоинт /logs (GET), возвращающий последние 10 записей лога. 8. Защитить эндпоинт /logs простым API-ключом (проверка заголовка X-API-Key).

	9. Протестировать с помощью curl. 10. Упаковать приложение в Docker, передать API-ключ через переменную окружения.
4	1. Обучить нейронную сеть (Keras) на MNIST, сохранить модель в формате .h5. 2. Создать FastAPI-приложение. 3. Реализовать эндпоинт /predict_image (POST), принимающий изображение в виде base64-строки. 4. Декодировать base64 в изображение, преобразовать в массив 28×28, нормализовать. 5. Выполнить предсказание (цифра от 0 до 9) и вернуть JSON с цифрой и массивом вероятностей. 6. Добавить ограничение на размер загружаемого файла (макс 1 МБ). 7. Реализовать кэширование предсказаний (если такое же изображение уже было) для ускорения. 8. Протестировать с помощью Postman (отправить base64). 9. Создать Dockerfile с multi-stage сборкой (уменьшить размер образа). 10. Запустить контейнер и проверить latency предсказания.
5	1. Обучить модель на датасете California Housing (регрессия) с использованием StandardScaler. Сохранить модель и scaler в один файл (например, с помощью pickle или joblib). 2. Создать Flask-приложение с эндпоинтом /predict для одного объекта. 3. Добавить эндпоинт /predict_batch (POST), принимающий список объектов (batch prediction). 4. Реализовать векторизованное предсказание для батча (преобразовать список в массив, масштабировать, предсказать). 5. Измерить время обработки батча из 100 объектов и одного объекта 100 раз – сравнить. 6. Добавить эндпоинт /metrics (GET), возвращающий количество запросов и среднее время обработки. 7. Использовать gunicorn как WSGI-сервер для Flask. 8. Написать docker-compose.yml для запуска приложения и Nginx в качестве reverse proxy. 9. Протестировать нагрузку с помощью ab (Apache Bench) или locust. 10. Сохранить результаты нагрузочного тестирования (RPS, latency).

6	1. Обучить модель на датасете Wine (классификация на 3 класса). Сохранить модель и список имён признаков. 2. Создать FastAPI-приложение. 3. Реализовать эндпоинт /predict с валидацией входных данных через Pydantic. 4. Добавить эндпоинт /model_info (GET), возвращающий информацию о модели: метрики (accuracy), дата обучения, версия модели. 5. Хранить метаинформацию в отдельном JSON-файле, загружать при старте. 6. Реализовать версионирование модели: эндпоинт /v1/predict и /v2/predict (разные версии). 7. Добавить middleware для логирования времени выполнения каждого запроса. 8. Протестировать оба эндпоинта. 9. Собрать Docker-образ с тегом версии. 10. Запустить контейнер и проверить логи.
7	1. Обучить модель LightGBM на датасете Customer Churn. Сохранить модель в формате .txt. 2. Создать Flask-приложение с эндпоинтом /predict, принимающим JSON. 3. Реализовать предобработку: заполнение пропусков медианой (параметры предобработки сохранить в JSON). 4. Добавить эндпоинт /preprocess_example (POST), который возвращает предобработанные данные без предсказания (для отладки). 5. Использовать библиотеку pydantic для валидации входных данных (если Flask, то через webargs или ручную проверку). 6. Добавить кэширование модели через functools.lru_cache. 7. Написать тесты с использованием pytest для эндпоинта /predict (позитивные и негативные сценарии). 8. Интегрировать pytest в Docker-сборку (запуск тестов при сборке). 9. Настроить CI/CD через GitHub Actions (проверка тестов, сборка образа, публикация в Docker Hub). 10. Сохранить все файлы (код, модель, тесты, Dockerfile, workflow) в репозиторий.
8	1. Обучить модель на датасете Spam SMS (TfidfVectorizer + LogisticRegression). Сохранить векторизатор и модель. 2. Создать FastAPI-приложение. 3. Реализовать эндпоинт /predict_sms (POST),

	принимающий {"text": "..."}. 4. Внутри эндпоинта применить векторизатор (transform) и модель. 5. Вернуть {"is_spam": bool, "confidence": float}. 6. Добавить эндпоинт /stats (GET), возвращающий общее количество запросов и количество спам/не спам предсказаний. 7. Использовать глобальные счётчики с потокобезопасностью (multiprocessing.Value или threading.Lock). 8. Протестировать с помощью requests в отдельном скрипте (100 запросов). 9. Упаковать в Docker. 10. Развернуть на бесплатном хостинге (Render.com или Railway) и предоставить публичный URL.
9	1. Обучить модель Random Forest на датасете Heart Disease. Сохранить модель и метаинформацию. 2. Создать Flask-приложение с эндпоинтом /predict и /explain. 3. В эндпоинте /explain использовать LIME для объяснения предсказания (вернуть важность признаков в JSON). 4. Для LIME предварительно обучить объяснитель на обучающих данных. 5. Ограничить число запросов до 10 в минуту (rate limiting) с помощью Flask-Limiter. 6. Добавить авторизацию по JWT-токену (простая реализация). 7. Протестировать эндпоинт с токеном и без него. 8. Написать Dockerfile. 9. Запустить контейнер и проверить, что rate limiting работает. 10. Сохранить логи авторизации и запросов.
10	1. Обудить модель Keras на Fashion-MNIST, сохранить в формате SavedModel. 2. Создать FastAPI-приложение с эндпоинтом /predict для одного изображения (base64) и /predict_batch для списка изображений. 3. Использовать асинхронное выполнение предсказаний (в FastAPI это async def). 4. Для асинхронной обработки батча использовать asyncio.gather. 5. Добавить эндпоинт /models (GET) – список доступных моделей (можно несколько версий). 6. Реализовать возможность выбора модели через параметр model_version в запросе. 7. Загружать модели при старте в словарь. 8. Написать docker-compose.yml для сервиса и Redis (для кэширования).

	9. Кэшировать предсказания в Redis (ключ – хэш изображения). 10. Протестировать производительность с кэшем и без него.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое REST API? Какие методы HTTP обычно используются для получения и отправки данных?
2. Как сохранить обученную модель на диск? Какие форматы и библиотеки вы знаете (joblib, pickle, ONNX, SavedModel)?
3. В чём разница между Flask и FastAPI? Какие преимущества даёт FastAPI (асинхронность, валидация, документация)?
4. Как загрузить модель при старте приложения и сделать её доступной в эндпоинтах?
5. Что такое Docker? Зачем контейнеризировать ML-приложение?
6. Как написать Dockerfile для Python-приложения? Какие инструкции обязательны (FROM, COPY, RUN, CMD)?
7. Как протестировать API с помощью curl или Postman? Приведите пример команды.
8. Что такое переменные окружения и как их использовать для хранения конфиденциальных данных (API-ключи, пароли)?

9. Как обрабатывать ошибки в эндпоинтах (неверный JSON, отсутствие полей, ошибка модели) и возвращать корректные HTTP-статусы?
10. Что такое Swagger/OpenAPI и как его автоматически сгенерировать в FastAPI?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Корректность реализации API. Правильная обработка запросов, валидация, возврат JSON, HTTP-статусы.	10
Работа с моделью (загрузка, предсказание). Модель загружается один раз, предсказания выполняются корректно.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, комментарии, чистота кода.	10
Выводы и интерпретация. Анализ выбора фреймворка, оценка производительности, обоснование решений.	10
Docker-контейнеризация. Корректный Dockerfile, сборка образа, запуск контейнера, проверка работы.	10

Лабораторная работа № 36

Интеграция модели в веб-приложение (Streamlit)

Цель: Разработать полноценное веб-приложение с UI для взаимодействия с моделью.

Теоретическое обоснование

Streamlit — фреймворк на Python для быстрого создания интерактивных веб-приложений для машинного обучения и анализа данных. В отличие от Flask/FastAPI, Streamlit предоставляет готовые UI-компоненты и не требует написания HTML/CSS/JavaScript.

Основные компоненты Streamlit:

- st.title(), st.header(), st.write() — текст и заголовки.

- st.file_uploader() — загрузка файлов (изображений, CSV, текстов).
- st.text_input(), st.text_area() — поля ввода текста.
- st.slider(), st.number_input(), st.selectbox() — выбор числовых и категориальных параметров.
- st.button() — кнопка для запуска предсказания.
- st.image(), st.dataframe(), st.plotly_chart() — визуализация результатов.
- st.cache_data, st.cache_resource — кэширование загрузки моделей и данных для повышения производительности.

Интеграция модели. Загрузка модели происходит однократно при запуске приложения с использованием кэширования. При взаимодействии пользователя (нажатие кнопки, изменение слайдера) модель применяется к введённым данным и результат отображается.

Пример потока для классификации изображений:

1. Пользователь загружает изображение через st.file_uploader.
2. Приложение отображает загруженное изображение.
3. По нажатию кнопки «Распознать» загружается предобученная модель CNN.
4. Изображение предобрабатывается (ресайз, нормализация).
5. Модель делает предсказание, выводится класс и уверенность.
6. Результат отображается в виде метрики или диаграммы.

Деплой. Streamlit приложения можно легко развернуть:

- **Streamlit Cloud** — бесплатный хостинг для публичных репозиториев GitHub. Поддерживает автоматический деплой при пуше в репозиторий.
- **Hugging Face Spaces** — платформа для демонстрации ML-приложений с поддержкой Streamlit, Gradio, Jupyter. Интегрирована с моделями Hugging Face.
- **Self-hosted** — запуск на своём сервере через streamlit run app.py.

Преимущества Streamlit:

- Минимальный код — приложение создаётся быстро (десятки строк).
- Интерактивность «из коробки» — не нужно писать JS.
- Встроенная поддержка кэширования, прогресс-баров, виджетов.

Недостатки: меньше гибкости, чем у полноценных веб-фреймворков; для production требуется дополнительная настройка (аутентификация,

масштабирование).

Использование в образовании и прототипировании: Streamlit идеален для быстрого демо модели, проверки гипотез, создания дашбордов.

Методика и порядок выполнения работы

Задание 1. Индивидуальные задания к лабораторной работе:

Вариант	Задание
1	1. Обучить модель логистической регрессии на датасете Iris. Сохранить модель в формате pkl. 2. Создать Streamlit-приложение с заголовком «Классификация ирисов Фишера». 3. Добавить в боковую панель (st.sidebar) слайдеры для ввода 4 признаков: sepal_length, sepal_width, petal_length, petal_width. 4. Загрузить модель при старте приложения с использованием @st.cache_resource. 5. По нажатию кнопки «Предсказать» выполнить предсказание и вывести название класса. 6. Дополнительно вывести вероятности для всех трёх классов в виде bar chart. 7. Добавить изображение цветка ириса (любое) с помощью st.image(). 8. Создать файл requirements.txt со всеми зависимостями. 9. Разместить код на GitHub в публичном репозитории. 10. Развернуть приложение на Streamlit Cloud и предоставить ссылку.
2	1. Обучить модель Random Forest на датасете Boston Housing (регрессия). Сохранить модель. 2. Создать Streamlit-приложение с заголовком «Предсказание стоимости дома». 3. Добавить поля ввода (числовые) для всех признаков (например, st.number_input). 4. Загрузить модель и scaler (если использовался). 5. При изменении любого поля автоматически обновлять предсказание (без кнопки). 6. Отобразить предсказанную стоимость с форматированием (знак доллара). 7. Добавить слайдер для выбора количества деревьев (если модель позволяет – переобучить? лучше не надо, просто демонстрация).

	8. Добавить информацию о модели (R^2, MAE) в расширяемой секции (st.expander). 9. Сохранить приложение в репозиторий. 10. Развернуть на Streamlit Cloud и проверить.
3	1. Обучить сверточную нейронную сеть на MNIST (или взять предобученную). Сохранить модель в .h5. 2. Создать Streamlit-приложение для распознавания рукописных цифр. 3. Добавить виджет загрузки изображения (st.file_uploader), принимающий PNG/JPG. 4. Отобразить загруженное изображение. 5. Предобработать изображение: преобразовать в градации серого, ресайз до 28×28, нормализовать. 6. Выполнить предсказание и вывести цифру крупным шрифтом. 7. Показать bar chart вероятностей для всех цифр (0–9). 8. Добавить примеры изображений (кнопка «Загрузить пример»). 9. Сохранить приложение в GitHub. 10. Развернуть на Hugging Face Spaces (с поддержкой Streamlit).
4	1. Обучить модель на датасете Titanic (классификация выживших). Сохранить модель и препроцессор (OneHotEncoder, StandardScaler). 2. Создать Streamlit-приложение с формой ввода данных пассажира: o Pclass (selectbox), Sex (radio), Age (slider), SibSp (number), Parch (number), Fare (number), Embarked (selectbox). 3. Применить препроцессор к введённым данным. 4. Выполнить предсказание и вывести результат («Выживет» или «Не выживет») с цветовым выделением (зелёный/красный). 5. Добавить вывод вероятности выживания в процентах. 6. Использовать st.session_state для хранения истории предсказаний (последние 5). 7. Добавить кнопку «Очистить историю». 8. Визуализировать важность признаков (из модели) в виде bar chart. 9. Сохранить приложение. 10. Развернуть на Streamlit Cloud.
5	1. Обучить модель на датасете California Housing (регрессия). Сохранить модель и scaler. 2. Создать Streamlit-приложение для предсказания стоимости дома. 3. Добавить числовые поля для всех признаков (8 признаков).

	4. Добавить карту (Folium или st.map) для отображения предсказанной стоимости в выбранных координатах (дополнительные поля latitude, longitude). 5. При изменении координат отображать маркер на карте. 6. Выполнять предсказание и отображать стоимость. 7. Добавить слайдер для изменения медианного дохода (MedInc) и показывать, как меняется предсказание. 8. Сохранить результаты предсказаний в CSV (кнопка «Скачать историю»). 9. Развернуть приложение. 10. Предоставить ссылку.
6	1. Обучить модель на датасете SMS Spam (TfidfVectorizer + LogisticRegression). Сохранить векторизатор и модель. 2. Создать Streamlit-приложение для определения спама. 3. Добавить текстовое поле (st.text_area) для ввода сообщения. 4. При нажатии кнопки «Проверить» выполнить предсказание. 5. Вывести результат: «Спам» (красный) или «Не спам» (зелёный). 6. Добавить подсветку слов, которые больше всего повлияли на предсказание (использовать LIME или просто вывести топ-5 слов с весами). 7. Добавить примеры сообщений (кнопки «Загрузить пример спама» и «Загрузить пример не спама»). 8. Сохранить историю проверок в сессии. 9. Развернуть приложение. 10. Предоставить ссылку.
7	1. Обучить модель на датасете Fashion-MNIST (10 классов). Сохранить модель. 2. Создать Streamlit-приложение для классификации одежды. 3. Добавить загрузку изображения или выбор из предустановленных примеров (selectbox). 4. Отобразить изображение и предсказанный класс. 5. Вывести уверенность модели (probability). 6. Показать top-3 наиболее вероятных класса. 7. Добавить возможность переключения между разными моделями (например, CNN vs полносвязная) – загрузить обе. 8. Визуализировать карту активаций (Grad-CAM) для CNN – упрощённо (опционально). 9. Сохранить приложение. 10. Развернуть на Hugging Face Spaces.
8	1. Обучить модель на датасете Wine (3 класса). Сохранить

	модель и scaler. 2. Создать Streamlit-приложение с многомерным вводом всех 13 признаков. 3. Использовать st.columns для компактного расположения полей. 4. Добавить кнопку «Случайный пример» (выбирает случайный объект из тестовой выборки). 5. Выполнить предсказание и показать название сорта вина. 6. Добавить radar chart для сравнения введённых признаков со средними значениями по классам. 7. Сохранить сессию пользователя (введённые параметры) в st.session_state. 8. Добавить экспорт в PDF отчёта о предсказании. 9. Развернуть приложение. 10. Предоставить ссылку.
9	1. Обучить модель на датасете Heart Disease (бинарная классификация). Сохранить модель. 2. Создать Streamlit-приложение «Калькулятор риска сердечных заболеваний». 3. Добавить поля ввода: возраст, пол, тип боли в груди, давление, холестерин, максимальный пульс и др. 4. Использовать st.slider и st.selectbox. 5. Выполнять предсказание автоматически при изменении любого поля. 6. Отображать результат в виде индикатора риска (проценты) и цветного круга (progress bar). 7. Добавить рекомендации по снижению риска (простые текстовые советы). 8. Сохранять результаты в CSV. 9. Развернуть приложение. 10. Предоставить ссылку.
10	1. Обучить модель на датасете CIFAR-10 (сверточная сеть). Сохранить модель. 2. Создать Streamlit-приложение для классификации изображений. 3. Добавить загрузку изображения, поддержку JPG/PNG. 4. Отобразить загруженное изображение. 5. Предобработать: ресайз до 32×32, нормализация. 6. Выполнить предсказание и вывести класс (самолёт, автомобиль, птица и т.д.). 7. Показать bar chart вероятностей для всех 10 классов.

	8. Добавить кнопку «Загрузить случайное изображение из тестового набора». 9. Сохранить предсказания с изображениями в сессии. 10. Развернуть на Streamlit Cloud.

Содержание отчета и его форма

1) Титульный лист (название работы, ФИО, группа).
2) Цель работы (формулировка из задания).
3) Ход выполнения работы – описание каждого пункта индивидуального задания с указанием, что именно делалось и какой результат получен (с листингом кода). Результаты оформляются в виде таблиц, скриншотов графиков, вывода числовых значений.
4) Выводы по работе (достигнута ли цель, что нового освоено, какие трудности, анализ результатов).
5) Ответы на контрольные вопросы (письменно, развёрнуто).

Вопросы для защиты работы

1. Что такое Streamlit и для каких задач он наиболее полезен?
2. Как вывести текст, заголовки и изображения в Streamlit? Приведите примеры.
3. Какие виджеты ввода данных существуют в Streamlit (слайдер, текстовое поле, селектор, кнопка, загрузка файла)?
4. Как закэшировать загрузку модели с помощью `@st.cache_resource` и почему это важно?
5. Что такое `st.session_state` и как его использовать для хранения состояния между перезапусками приложения?
6. Как создать боковую панель (`st.sidebar`) и добавить в неё элементы?
7. Как развернуть Streamlit-приложение на Streamlit Cloud? Какие файлы необходимы (requirements.txt, .streamlit/config.toml)?
8. Как развернуть приложение на Hugging Face Spaces? В чём отличие от Streamlit Cloud?
9. Как обрабатывать загрузку изображения и преобразовывать его в формат, подходящий для модели (например, для MNIST)?

10.Как в Streamlit создать многостраничное приложение (мультистраничность)?

Критерии оценивания

Параметр	Баллы
Выполнение индивидуального задания (10 пунктов).	40
Качество интерфейса и UX. Удобство использования, логичность расположения виджетов, отзывчивость.	10
Правильность интеграции модели. Корректная загрузка модели, предобработка ввода, вывод предсказаний.	10
Ответы на контрольные вопросы (10 вопросов).	10
Структура и оформление отчёта. Наличие всех разделов, аккуратность, наличие скриншотов.	10
Выводы и интерпретация. Анализ выбора инструментов, оценка сложности разработки, перспективы использования.	10
Деплой приложения (публичная ссылка). Приложение доступно по ссылке, работает корректно.	10

СПСИОК ЛИТЕРАТУРЫ

1. Николенко, С. И. Глубокое обучение: погружение в мир нейронных сетей / С. И. Николенко, А. А. Кадурин, Е. В. Архангельская. — Санкт-Петербург : Питер, 2025. — 476 с. — (Библиотека программиста). — ISBN 978-5-4461-1537-2.

2. Джон, Д. Машинное обучение на Python: учебник / Д. Джон. — Москва : КноРус, 2025. — 431 с. — ISBN 978-5-406-14728-3.

3. Главачик, Д. В. Python для аналитики данных: учебное пособие / Д. В. Главачик. — Москва : ИНФРА-М, 2025. — 318 с.

4. Лейн, Х. Обработка естественного языка в действии / Х. Лейн, Х. Ханнес, К. Ховард. — 2-е изд. — Санкт-Петербург : Питер, 2026. — 496 с. — ISBN 978-5-4461-2037-6.

5. Сацюк, А. В. Компьютерное зрение и нейронные сети: практика : учебное пособие / А. В. Сацюк. — Москва ; Вологда : Инфра-Инженерия, 2025. — 220 с. — ISBN 978-5-9729-1635-9.

6. Николенко, С. И. Глубокое обучение: погружение в мир нейронных сетей / С. И. Николенко, А. А. Кадурин, Е. В. Архангельская. — Санкт-Петербург : Питер, 2025. — 476 с. — (Библиотека программиста). — ISBN 978-5-4461-1537-2.

7. Джон, Д. Машинное обучение на Python: учебник / Д. Джон. — Москва : КноРус, 2025. — 431 с. — ISBN 978-5-406-14728-3.

8. Главачик, Д. В. Python для аналитики данных: учебное пособие / Д. В. Главачик. — Москва : ИНФРА-М, 2025. — 318 с.

9. Лейн, Х. Обработка естественного языка в действии / Х. Лейн, Х. Ханнес, К. Ховард. — 2-е изд. — Санкт-Петербург : Питер, 2026. — 496 с. — ISBN 978-5-4461-2037-6.

10. Сацюк, А. В. Компьютерное зрение и нейронные сети: практика : учебное пособие / А. В. Сацюк. — Москва ; Вологда : Инфра-Инженерия, 2025. — 220 с. — ISBN 978-5-9729-1635-9.

11. Николенко, С. И. Глубокое обучение: погружение в мир нейронных сетей / С. И. Николенко, А. А. Кадурин, Е. В. Архангельская. — Санкт-Петербург : Питер, 2025. — 476 с. — (Библиотека программиста). — ISBN 978-5-4461-1537-2.

12. Джон, Д. Машинное обучение на Python: учебник / Д. Джон. — Москва : КноРус,

2025. — 431 с. — ISBN 978-5-406-14728-3.

13. Главачик, Д. В. Python для аналитики данных: учебное пособие / Д. В. Главачик. — Москва : ИНФРА-М, 2025. — 318 с.

14. Лейн, Х. Обработка естественного языка в действии / Х. Лейн, Х. Ханнес, К. Ховард. — 2-е изд. — Санкт-Петербург : Питер, 2026. — 496 с. — ISBN 978-5-4461-2037-6.

15. Сацюк, А. В. Компьютерное зрение и нейронные сети: практика : учебное пособие / А. В. Сацюк. — Москва ; Вологда : Инфра-Инженерия, 2025. — 220 с. — ISBN 978-5-9729-1635-9.