Автономная некоммерческая профессиональная образовательная организация
«МЕЖДУНАРОДНЫЙ ВОСТОЧНО-ЕВРОПЕЙСКИЙ КОЛЛЕДЖ»
Пушкинская ул., д. 268, 426008, г. Ижевск. Тел.: (3412) 77-68-24. E-mail: mveu@mveu.ru, www. mveu.ru
ИНН 1831200089. ОГРН 1201800020641

20.02.2026

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по выполнению практических работ

профессионального модуля

ПМ.01 РАЗРАБОТКА КОДА ДЛЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

по специальности

09.02.13 Интеграция решений с применением технологий искусственного интеллекта

2026

ДОКУМЕНТ ПОДПИСАН
ЭЛЕКТРОННОЙ ПОДПИСЬЮ

Сертификат: 0217109D0063B3FC884CC579B7E3A15ED9
Владелец: Новикова Варвара Валентиновна
Действителен: с 25.09.2025 до 25.12.2026

Практическая работа – небольшой научный отчет, обобщающий проведенную учащимся работу, которую представляют для защиты преподавателю.

В процессе практического занятия учащиеся выполняют одну или несколько практических работ (заданий) под руководством преподавателя в соответствии с изучаемым содержанием учебного материала.

Состав и содержание практических занятий направлены на реализацию Государственных требований.

Наряду с формированием умений и навыков в процессе практических занятий обобщаются, систематизируются, углубляются и конкретизируются теоретические знания, вырабатывается способность и готовность использовать теоретические знания на практике, развиваются интеллектуальные умения.

Практические занятия проводятся в форме практической подготовки в виде работ, связанных с будущей профессиональной деятельностью.

К практическим работам предъявляется ряд требований, основным из которых является полное, исчерпывающее описание всей проделанной работы, позволяющее судить о полученных результатах, степени выполнения заданий и профессиональной подготовке учащихся.

I. Практические работы:

МДК 01.01. Разработка программных модулей в системах искусственного интеллекта

Тема практической работы №1. Анализ примеров успешных решений на основе ИИ, объем часов 8

У1. Анализировать технические задания и выявлять требования к алгоритмам.
Применять методы алгоритмизации для решения задач программирования.
Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

Цель практической работы:

Научиться самостоятельно анализировать примеры успешных решений на основе ИИ, выявлять используемые алгоритмы, оценивать чистоту кода и формировать структурированный отчёт.

Задание(я):

1. Выбрать три открытых проекта с реализацией ИИ-моделей (например, классификатор на датасете Iris, распознавание цифр MNIST, рекомендационная система на MovieLens). Сохранить список выбранных репозиториев в файл projects.txt.

2. Для каждого проекта изучить структуру кода, определить применяемые алгоритмы (поиск, жадный, динамическое программирование и т.д.) и используемые библиотеки. Сформировать таблицу сравнения в формате CSV (columns: Project, Algorithm, Library, Main Module). Сохранить как comparison.csv.

3. Оценить качество кода с помощью инструмента PyLint. Установить PyLint (pip install pylint). Запустить проверку для каждого проекта, сохранить отчёт PyLint в виде текста и скриншот графика количества предупреждений. Сохранить отчёты как pylint_project1.txt, pylint_project2.txt, pylint_project3.txt и скриншоты pylint_project1.png, pylint_project2.png, pylint_project3.png.

4. Подготовить итоговый отчёт, включив титульный лист, описание целей, методику анализа, таблицу сравнения, результаты PyLint, выводы и скриншоты. Сохранить отчёт в формате ODT (или PDF) с именем report.odt.

Методические указания по ходу выполнения работы:

В задании 1 используйте поисковые системы и официальные каталоги GitHub, выбирая проекты, где реализованы базовые ИИ-задачи, написанные на Python. Запишите ссылки в файл projects.txt.

В задании 2 откройте каждый проект в IDE, найдите файлы с основным кодом, определите используемые алгоритмы и библиотеки, занесите данные в таблицу comparison.csv, следуя указанным колонкам.

В задании 3 установите PyLint командой pip install pylint. Для каждого проекта выполните проверку PyLint, перенаправив вывод в текстовый файл. Сохраните скриншоты окна терминала с результатами в PNG-файлы. Поместите все файлы в отдельную папку analysis_results.

В задании 4 соберите все полученные материалы в один документ. Оформите титульный лист (название работы, ФИО, группа, дата). В разделе

«Методика анализа» опишите последовательность действий, в разделе «Результаты» включите таблицу comparison.csv (в виде изображения) и скриншоты PyLint. В разделе «Выводы» сформулируйте основные наблюдения о применяемых алгоритмах и качестве кода.

Формат сдачи: создайте ZIP-архив, содержащий папку с исходными файлами (projects.txt, comparison.csv, pylint_*.txt, pylint_*.png) и итоговый отчёт report.odt (или PDF).

Отчёт должен быть в формате ODT или PDF. Структура: титульный лист; цель и задачи работы; методика анализа; таблица сравнения (в виде изображения); результаты PyLint (текст и скриншоты); выводы и рекомендации; список использованных источников.

Сдача: упаковать все файлы в архив ZIP и отправить по электронной почте преподавателю до конца занятия.

Тема практической работы №2. Создание базовой модели ИИ для классификации данных, объем часов 10

У1. Анализировать технические задания и выявлять требования к алгоритмам.
Применять методы алгоритмизации для решения задач программирования.
Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

Цель практической работы:

Научиться самостоятельно разрабатывать, обучать и оценивать базовую модель машинного обучения для задачи классификации, оформлять результаты в виде отчёта и сохранять артефакты проекта.

Задание(я):

1. Подготовка рабочей среды, установка необходимых библиотек и загрузка датасета Iris.

2. Выполнение предварительного анализа данных: просмотр статистики, построение простых графиков распределения признаков.

3. Разделение данных на обучающую и тестовую выборки, обучение модели Logistic Regression.

4. Оценка качества модели: расчёт метрик точности, построение графика зависимости точности от параметра C, сохранение графика.

5. Сохранение обученной модели, экспорт предсказаний на тестовой выборке в CSV, сохранение исходного кода в файл main.py.

6. Оформление отчёта: титульный лист, описание выполненных шагов, скриншоты результатов, выводы; подготовка финального архива.

Методические указания по ходу выполнения работы:

В задании 1 создайте виртуальное окружение, установите пакеты scikit-learn, pandas, matplotlib, joblib (pip install scikit-learn pandas matplotlib joblib). Сохраните код в файл main.py.

В задании 2 загрузите встроенный датасет Iris, преобразуйте его в DataFrame, выполните описательную статистику и постройте гистограммы/диаграммы рассеяния. Сохраните каждый график в отдельный файл PNG (plt.savefig('feature_hist.png')).

В задании 3 разделите набор на 70 % обучающих и 30 % тестовых образцов, обучите модель Logistic Regression с параметром C=1.0. Сохраните обученную модель в файл model.pkl (joblib.dump).

В задании 4 вычислите метрики точности, precision, recall и f1-score на тестовой выборке. Постройте график зависимости точности от разных значений C (например, 0.1, 1, 10) и сохраните его как accuracy_vs_C.png.

В задании 5 примените обученную модель к тестовым данным, экспортируйте таблицу с реальными и предсказанными метками в файл predictions.csv. Убедитесь, что весь исходный код (main.py) и артефакты (model.pkl, *.png, predictions.csv) находятся в одной папке.

В задании 6 подготовьте отчёт в формате ODT или PDF: титульный лист, краткое описание каждого задания, вставьте скриншоты графиков и таблицу предсказаний, сформулируйте выводы о качестве модели. Сохраните отчёт как report.odt (или report.pdf). Скомпонуйте все файлы (main.py, model.pkl, *.png, predictions.csv, report.odt) в один ZIP-архив.

Формат отчёта: ODT или PDF. Структура: титульный лист, цель работы, описание каждого задания, скриншоты/графики, таблица предсказаний, выводы и список использованных библиотек.

Сдача: упаковать все файлы проекта в архив ZIP и отправить по email преподавателя не позднее конца занятия.

Тема практической работы №3. Сбор данных с использованием веб-скрапинга и API, объем часов 10

У1. Анализировать технические задания и выявлять требования к алгоритмам. Применять методы алгоритмизации для решения задач программирования. Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.
У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

Цель практической работы:

Научиться самостоятельно собирать данные из открытых веб-источников и API, выполнять их предобработку, сохранять в удобных форматах и оформлять результаты для дальнейшего использования в проектах ИИ.

Задание(я):

1. Подготовка рабочего окружения (установить Python, необходимые библиотеки, создать структуру проекта).

2. Выбор открытого API (например, OpenWeatherMap, GitHub Jobs) и изучение его документации.

3. Получение данных через API и сохранение ответа в файл JSON.

4. Преобразование полученного JSON в таблицу и экспорт в CSV.

5. Выбор простого сайта для скрапинга (например, список книг на openlibrary.org).

6. Сбор HTML-страниц с помощью запросов и сохранение их локально.

7. Парсинг сохранённых HTML-файлов, извлечение нужных полей (название, автор, дата) и запись в CSV.

8. Объединение CSV-файлов из пунктов 4 и 7 в один общий набор данных, удаление дублирующихся строк.

9. Добавление простого логирования (запись времени начала/окончания, количество записей) и сохранение лога в файл .log.

10. Подготовка отчёта: описание выполненных шагов, скриншоты кода, выводов, графики (если есть) и упаковка всех материалов в ZIP-архив.

Методические указания по ходу выполнения работы:

В задании 1 создайте папку проекта, внутри подпапки src, data, logs и report. Установите библиотеки: pip install requests beautifulsoup4 pandas. Сохраните скрипты в src/*.py.

В задании 2 выберите публичный API, изучите параметры запроса (ключ, endpoint, параметры). Оформите запрос в отдельном .py файле и запишите полученный JSON в data/api_response.json.

В задании 3 откройте файл JSON, преобразуйте его в DataFrame (pandas) и сохраните как data/api_data.csv с помощью метода to_csv (без индекса).

В задании 4 подготовьте CSV-файл с колонками, соответствующими выбранным полям, проверьте корректность данных (отсутствие NaN, типы).

В задании 5 определите URL-адрес страницы со списком элементов, убедитесь, что доступ к странице не ограничен (robots.txt).

В задании 6 выполните запрос GET к странице, сохраните полученный HTML в data/raw_page.html. При необходимости сделайте запрос к нескольким страницам и сохраните каждый в отдельный файл.

В задании 7 используя BeautifulSoup, распарсите сохранённые HTML-файлы, извлеките интересующие атрибуты (например, title, author, year) и сформируйте DataFrame, который экспортируйте в data/scraped_data.csv.

В задании 8 загрузите оба CSV-файла, выполните объединение (concat) по общим колонкам, удалите дубли с помощью drop_duplicates и сохраните результат в data/combined_dataset.csv.

В задании 9 настройте модуль logging: создайте файл logs/collection.log, запишите в него время начала и окончания каждого этапа, количество полученных записей и сообщения об ошибках (если они возникли).

В задании 10 подготовьте отчёт в LibreOffice: титульный лист, краткое описание каждого задания, скриншоты содержимого файлов (Python-скрипт, CSV-файлы, лог), выводы о качестве полученных данных. Сохраните отчёт как report.odt (или pdf). Упакуйте в один zip-архив корневую папку проекта.

Отчёт оформляется в ODT или PDF, содержит титульный лист, список выполненных заданий, описание подходов, скриншоты кода и файлов, таблицы с результатами, лог-файл и выводы о качестве данных.

Сдача: упаковать весь проект (папка с src, data, logs, report) в архив ZIP и отправить по электронной почте преподавателю до конца занятия.

Тема практической работы №4. Предобработка данных для машинного обучения: очистка, нормализация, кодирование, объем часов 10

У1. Анализировать технические задания и выявлять требования к алгоритмам. Применять методы алгоритмизации для решения задач программирования. Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.
У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

Цель практической работы:

Освоить навыки самостоятельной предобработки данных для машинного обучения: очистка, нормализация, кодирование категориальных признаков и оформление результатов.

Задание(я):

1. Ознакомление с набором данных Iris (CSV). Сохранить описательную статистику в файл stats.csv.

2. Очистка данных: выявить и обработать пропуски и выбросы. Сохранить очищенный набор в clean.csv.

3. Нормализация числовых признаков (Min-Max). Сохранить нормализованный набор в normalized.csv.

4. Кодирование категориального признака Species (One-Hot). Сохранить закодированный набор в encoded.csv.

5. Визуализация распределения признаков до и после предобработки. Сохранить графики как before.png и after.png, включить их в отчёт.

6. Организация кода в модули: создать файл preprocess.py с функциями load_data, clean_data, normalize, encode.

7. Добавление логирования процесса предобработки (модуль logging). Сохранить журнал в preprocess.log.

8. Отладка скрипта с использованием точек останова в PyCharm, зафиксировать скриншот окна отладчика и сохранить как debug.png.

Методические указания по ходу выполнения работы:

В задании 1 загрузите CSV-файл Iris, используя pandas, выполните первичный анализ (head, describe, info) и запишите полученные метрики в stats.csv.

В задании 2 проверьте наличие NaN, замените их медианой или удалите строки с пропусками; выявите выбросы с помощью межквартильного диапазона и удалите их. Сохраните результат в clean.csv.

В задании 3 примените Min-Max масштабирование к числовым столбцам (exclude target). Сохраните масштабированные данные в normalized.csv.

В задании 4 выполните One-Hot кодирование столбца Species, удалив оригинальный столбец. Сохраните полученный набор в encoded.csv.

В задании 5 постройте гистограммы и box-plot для каждого числового признака до и после предобработки, используя matplotlib. Сохраните графики в before.png и after.png, включите их в отчёт.

В задании 6 разбейте весь процесс на функции, разместите их в файле preprocess.py, убедитесь, что скрипт можно импортировать и последовательно выполнять.

В задании 7 добавьте в preprocess.py конфигурацию logging, выводите сообщения о начале и завершении каждой стадии предобработки, сохраняйте журнал в preprocess.log.

В задании 8 откройте preprocess.py в PyCharm, установите точки останова на каждой функции, выполните пошаговое выполнение, зафиксируйте скриншот окна отладчика (debug.png).

Отчёт оформляется в ODT или PDF, содержит: титульный лист (название работы, ФИО, группа), краткое описание цели, отдельный раздел для каждого задания с текстовым описанием, таблицами (CSV-файлы), скриншотами (debug.png) и графиками (before.png, after.png), выводы о проведённой предобработке.

Сдача: упаковать все файлы (stats.csv, clean.csv, normalized.csv, encoded.csv, before.png, after.png, preprocess.py, preprocess.log, debug.png) и отчёт в один ZIP-архив, отправить по e-mail преподавателю до конца занятия.

Тема практической работы №5. Реализация линейной регрессии на реальных данных, объем часов 10

У1. Анализировать технические задания и выявлять требования к алгоритмам.
Применять методы алгоритмизации для решения задач программирования.
Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.
У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных.
Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя.

Цель практической работы:

Научиться самостоятельно реализовывать линейную регрессию на реальных данных, проводить её оценку, оформлять результаты и писать автоматизированные тесты.

Задание(я):

1. Подготовка рабочего окружения (установить Python, необходимые библиотеки, создать проект).

2. Загрузка и предварительная обработка открытого датасета (например, датасет "Boston Housing"). Сохранить очищенные данные в CSV.

3. Реализация и обучение модели линейной регрессии с использованием scikit-learn. Сохранить обученную модель в файл (joblib).

4. Оценка модели: вычислить метрики MAE, MSE, R^2; построить график реальных vs предсказанных значений и сохранить его в PNG.

5. Написание наборов unit-тестов для функций загрузки данных, предобработки и предсказания модели; оформление отчёта и упаковка всех артефактов.

Методические указания по ходу выполнения работы:

Для задания 1 установите Python версии 3.8+, затем выполните pip install pandas scikit-learn matplotlib joblib pytest. Создайте папку проекта и внутри неё подпапки data, models, plots, tests, report.

В задании 2 загрузите выбранный датасет с помощью pandas, выполните базовую очистку (удаление пропусков, приведение типов). Сохраните полученный DataFrame в файл data/cleaned.csv через метод to_csv.

В задании 3 напишите скрипт (например, train.py) который читает data/cleaned.csv, разбивает данные на обучающую и тестовую выборки, обучает LinearRegression и сохраняет модель в models/linear_regression.joblib при помощи joblib.dump.

В задании 4 создайте скрипт (evaluate.py) который загружает модель, делает предсказания на тестовой части, вычисляет MAE, MSE и R^2, выводит их в консоль и сохраняет график сравнения реальных и предсказанных значений в plots/comparison.png через plt.savefig.

В задании 5 разработайте набор тестов (tests/test_pipeline.py) с использованием pytest: проверка наличия файлов data/cleaned.csv, корректности формы массивов, успешной загрузки модели, соответствия предсказаний ожидаемому типу. Оформите отчёт в документе report/report.odt (или PDF) со следующей структурой: титульный лист, цель, описание каждого задания, скриншоты кода и результатов (таблицы метрик, график), выводы. Упакуйте всё в архив ZIP: code (все .py файлы), data, models, plots, tests, report.

Все графики сохраняйте через plt.savefig('plots/имя.png'), модели – через joblib.dump(..., 'models/имя.joblib'), а экспортированные данные – через DataFrame.to_csv('data/имя.csv').

Отчёт в формате ODT или PDF. Структура: 1) Титульный лист (название работы, ФИО, группа, дата). 2) Цель практики. 3) Описание выполненных заданий. 4) Результаты (таблицы метрик, скриншоты кода, график comparison.png). 5) Выводы. 6) Приложения (список файлов, краткое содержание тестов).

Сдача: собрать все папки и файлы в один архив ZIP, назвать его <фамилия>_PM01_PW5.zip и отправить по электронной почте преподавателю не позднее окончания занятия.

Тема практической работы №6. Применение кластеризации для сегментации данных, объем часов 10

У1. Анализировать технические задания и выявлять требования к алгоритмам. Применять методы алгоритмизации для решения задач программирования. Разрабатывать оптимальные алгоритмы для решения задач в области ИИ.
У2. Реализовывать программные модули на основе требований технического задания.
Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.
У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных.
Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя.

Цель практической работы:

Освоить самостоятельное выполнение полного цикла кластеризации данных: подготовка данных, построение и оценка моделей k-means и иерархической кластеризации, визуализация результатов и автоматизированное тестирование кода.

Задание(я):

1. Подготовка окружения и загрузка датасета Iris; сохранить исходные данные в CSV.

2. Выполнение exploratory data analysis: расчёт базовых статистик, построение парных графиков распределения признаков; сохранить графики в PNG.

3. Реализация и обучение модели k-means; сохранить обученную модель (joblib), добавить метки кластеров к данным, экспортировать результат в CSV, построить график кластеров и сохранить.

4. Реализация и обучение иерархической (агломеративной) кластеризации; сохранить модель, экспортировать метки в CSV, построить дендрограмму и сохранить в PNG.

5. Создание набора unit-тестов (pytest) для функций загрузки данных, предобработки и обучения моделей; выполнить тесты, сохранить вывод в текстовый файл, оформить результаты тестирования.

Методические указания по ходу выполнения работы:

Для задания 1 установите необходимые библиотеки (pip install pandas scikit-learn matplotlib joblib pytest). Создайте файл load_data.py, в нём загрузите датасет Iris из scikit-learn, преобразуйте в DataFrame и сохраните как iris.csv в папку data.

Для задания 2 в отдельном скрипте eda.py загрузите iris.csv, вычислите средние, медианы и стандартные отклонения по каждому признаку, сохраните таблицу статистики в файл stats.csv. Постройте парные графики (scatter matrix) и сохраните каждый график как png в папку plots.

Для задания 3 создайте скрипт kmeans_clustering.py: загрузите данные, выполните стандартизацию признаков, обучите модель k-means с числом кластеров 3, сохраните модель в файл kmeans_model.joblib, добавьте полученные метки к исходному DataFrame и экспортируйте как iris_kmeans.csv. Постройте двумерный график (например, по первым двум признакам) с раскраской по кластерам и сохраните как kmeans_plot.png в папку plots.

Для задания 4 создайте скрипт agglomerative_clustering.py: загрузите данные, выполните стандартизацию, обучите AgglomerativeClustering с тем же числом кластеров, сохраните модель в agglomerative_model.joblib, экспортируйте данные с метками в iris_agg.csv. Постройте дендрограмму (используя scipy.linkage) и сохраните как dendrogram.png в папку plots.

Для задания 5 создайте директорию tests и файл test_clustering.py. Напишите тесты, проверяющие корректность загрузки CSV, размерность получаемых меток и тип возвращаемого объекта модели. Запустите pytest, перенаправьте вывод в файл test_results.txt и сохраните его в корень проекта.

Формат сдачи: создайте архив ZIP, содержащий папки data, plots, models, tests, все .py-файлы, CSV-файлы, сохранённые модели (.joblib), графики (.png), файл test_results.txt и отчёт.

Отчёт в формате ODT или PDF. Структура: титульный лист (название практики, ФИО, группа, дата); цель и задачи; описание выполненных шагов по каждому заданию; таблицы статистики и результаты кластеризации (CSV-файлы); скриншоты/вставки всех сохранённых графиков (PNG); фрагменты выводов тестов (из test_results.txt); выводы и выводы о качестве полученных кластеров.

Сдача: упаковать все материалы в один ZIP-архив и отправить по электронной почте преподавателю не позднее конца учебного дня.

Тема практической работы №7. Оценка качества модели с использованием ROC-кривой и F-меры, объем часов 10

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

Цель практической работы:

Научиться самостоятельно выполнять оценку качества классификационной модели с помощью ROC-кривой и F-меры.

Задание(я):

1. Установить необходимые библиотеки (pip install scikit-learn matplotlib pandas joblib).

2. Загрузить открытый бинарный датасет (например, Breast Cancer) из scikit-learn, выполнить базовую предобработку и разделить данные на тренировочную и тестовую выборки. Сохранить исходный набор в файл data.csv.

3. Обучить простой классификатор (логистическая регрессия) на тренировочных данных. Сохранить обученную модель в файл model.joblib.

4. На тестовой выборке рассчитать вероятности, построить ROC-кривую, вычислить AUC, построить график и сохранить его как roc_curve.png. Рассчитать F-меру при пороге 0.5 и при оптимальном пороге по ROC, результаты записать в файл metrics.csv.

5. Сохранить весь код в файл script.py, собрать все полученные файлы (data.csv, model.joblib, roc_curve.png, metrics.csv, script.py) в один ZIP-архив и подготовить отчёт.

Методические указания по ходу выполнения работы:

Для задания 1 используйте терминал/командную строку, выполните указанные pip-команды и проверьте успешность установки.

В задании 2 импортируйте нужный датасет из scikit-learn, выполните масштабирование признаков при необходимости, разделите данные функцией train_test_split и экспортируйте полную таблицу в CSV при помощи pandas.

В задании 3 создайте объект модели, обучите её методом fit, после обучения сохраните объект с помощью joblib.dump в файл model.joblib.

В задании 4 используйте метод predict_proba для получения вероятностей, постройте ROC-кривую и вычислите AUC функциями roc_curve и auc. Сохраните график командой plt.savefig('roc_curve.png'). Рассчитайте F-меру (precision, recall, f1_score) для двух порогов и запишите результаты в CSV (столбцы: threshold, precision, recall, f1).

В задании 5 соберите весь исходный код в один файл script.py, убедитесь, что в нём присутствуют комментарии с описанием шагов. Сформируйте ZIP-архив, включив в него все файлы из пунктов 2-4 и подготовьте отчёт в формате ODT или PDF.

Отчёт должен содержать: титульный лист, краткое описание каждой задачи, скриншоты кода (из script.py), изображение ROC-кривой, таблицу метрик из metrics.csv и выводы о качестве модели.

Отчёт в формате ODT или PDF: титульный лист, описание задач, скриншоты кода, график ROC-кривой (roc_curve.png), таблица метрик (metrics.csv) и раздел выводов.

Сдать ZIP-архив, содержащий script.py, data.csv, model.joblib, roc_curve.png, metrics.csv и отчёт, по электронной почте преподавателю до конца занятия.

Тема практической работы №8. Настройка гиперпараметров модели с использованием GridSearchCV, объем часов 10

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

Цель практической работы:

Научиться самостоятельно подбирать оптимальные гиперпараметры модели машинного обучения с помощью GridSearchCV и оценивать полученные результаты.

Задание(я):

1. Подготовка рабочего окружения, установка библиотек (scikit-learn, pandas, matplotlib, joblib) и загрузка датасета Iris. Результат: data.csv.

2. Предобработка данных: разделение на признаки и целевую переменную, кодирование, масштабирование, разбиение на обучающую и тестовую выборки. Результат: X_train.npy, X_test.npy, y_train.npy, y_test.npy.

3. Обучение базовой модели (например, LogisticRegression) без настройки гиперпараметров. Результат: baseline_model.joblib, метрики в baseline_metrics.csv.

4. Формирование сетки гиперпараметров (регуляризация C, тип penalty) и запуск GridSearchCV с кросс-валидацией (5-fold). Результаты: gridsearch_results.csv, лучшая модель best_model.joblib.

5. Анализ результатов GridSearch: построение графика зависимости качества (accuracy) от значений гиперпараметра C. Результат: hyperparameter_plot.png.

6. Оценка лучшей модели на отложенной тестовой выборке, расчёт точности, полноты, F-меры и построение ROC-кривой. Результаты: test_metrics.csv, roc_curve.png.

7. Подготовка отчёта: описание целей, шагов, полученных результатов, скриншотов кода и графиков. Результат: report.odt (или report.pdf).

Методические указания по ходу выполнения работы:

Создайте отдельную папку проекта и внутри неё подпапки data, models, plots, src.

Все библиотеки устанавливайте через pip install (например, pip install scikit-learn pandas matplotlib joblib).

Сохраняйте исходные CSV-файлы в папку data, а numpy-массивы – в src или data по вашему выбору.

Модели сохраняйте функцией joblib.dump в папку models с понятными именами (baseline_model.joblib, best_model.joblib).

Все графики сохраняйте через plt.savefig('имя.png') в папку plots.

Экспортируйте любые таблицы метрик в CSV через pandas.DataFrame.to_csv и сохраняйте в папку data.

Исходный код разбейте на отдельные .py файлы (например, data_preprocess.py, train_baseline.py, grid_search.py, evaluate.py) и сохраняйте в папку src.

Отчёт оформляйте в ODT или PDF: титульный лист, краткое описание задачи, пошаговый отчёт по каждому заданию, скриншоты кода, таблицы метрик, графики, выводы.

Все файлы (src, data, models, plots, report) упакуйте в один ZIP-архив для сдачи.

Отчёт должен содержать: титульный лист, цель практической работы, описание каждого задания, скриншоты кода, таблицы метрик (CSV-данные), сохранённые графики (PNG), выводы о выбранных гиперпараметрах. Формат – ODT или PDF.

Сдача: упаковать все папки (src, data, models, plots) и готовый отчёт в один ZIP-архив и отправить по email преподавателю до конца занятия.

Тема практической работы №9. Реализация многослойного перцептрона (MLP) для задачи классификации, объем часов 6

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя.

Цель практической работы:

Научиться самостоятельно разрабатывать, обучать и оценивать многослойный перцептрон (MLP) для задачи классификации, используя Python и библиотеку scikit-learn.

Задание(я):

1. Подготовить рабочее окружение и установить необходимые библиотеки (pip install scikit-learn matplotlib pandas joblib). Сохранить список пакетов в файл requirements.txt.

2. Скачать открытый датасет Iris, загрузить его в pandas, выполнить предварительный анализ (вывести первые строки, описательные статистики, построить парные графики). Сохранить исходный набор в файл iris.csv, а графики – в файлы pairplot.png и distribution.png.

3. Реализовать скрипт на Python, в котором создаётся MLP-модель, компилируется, обучается на тренировочной части датасета и сохраняется в файл model.pkl.

4. Оценить качество модели на тестовой части: построить матрицу ошибок, вычислить точность, полноту, F1-мера и сохранить результаты в файл metrics.txt; построить график зависимости функции потерь от эпох и сохранить как loss_curve.png.

5. Написать набор юнит-тестов (pytest) для проверки корректности функций предобработки, обучения и предсказания; выполнить тесты, сохранить их вывод в файл test_report.txt; собрать весь код, данные и отчёты в один архив.

Методические указания по ходу выполнения работы:

Для задания 1 создайте виртуальное окружение, активируйте его и установите перечисленные пакеты; после установки выполните команду pip freeze > requirements.txt.

В задании 2 загрузите датасет Iris через pandas.read_csv, разделите его на признаки и метку, выполните базовый EDA, сохраните полученный CSV и построенные графики с помощью matplotlib.pyplot.savefig.

В задании 3 напишите Python-скрипт, импортирующий необходимые модули, разбивающий данные на тренировочную и тестовую выборки (train_test_split), создающий объект MLPClassifier, обучающий его методом fit и сохраняющего модель функцией joblib.dump.

В задании 4 используйте предсказания модели на тестовом наборе, посчитайте метрики с помощью функций из sklearn.metrics, запишите их в текстовый файл; постройте график зависимости loss_curve_ от числа итераций и сохраните его.

В задании 5 создайте отдельный файл test_mlp.py, опишите тесты для функций предобработки, обучения и предсказания, запустите их командой pytest, перенаправьте вывод в test_report.txt; убедитесь, что все тесты проходят.

Подготовьте отчёт в формате ODT или PDF, включающий титульный лист, описание каждого задания, скриншоты результатов (графики, выводы), таблицу метрик и выводы по тестированию.

Соберите в один ZIP-архив следующие элементы: requirements.txt, iris.csv, pairplot.png, distribution.png, code.py (основной скрипт), model.pkl, metrics.txt, loss_curve.png, test_mlp.py, test_report.txt, отчёт.

Отчёт: ODT или PDF; титульный лист; раздел «Описание практики»; отдельный подраздел для каждого задания с текстовым описанием, скриншотами графиков и выводами; таблица метрик; раздел «Тестирование» с фрагментом вывода pytest; заключение.

Сдача: упаковать все указанные файлы в архив ZIP и отправить по email преподавателю до конца занятия.

Тема практической работы №10. Создание сверточной нейронной сети для распознавания изображений, объем часов 6

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя

Цель практической работы:

Научиться самостоятельно разрабатывать, обучать и оценивать сверточную нейронную сеть для распознавания изображений на базе открытого датасета.

Задание(я):

1. Подготовка окружения и загрузка датасета (MNIST). Результат: файл requirements.txt и скрипт preprocess.py.

2. Предобработка данных: нормализация и разделение на обучающую и тестовую выборки. Результат: файлы train.npy, test.npy, labels.npy.

3. Создание архитектуры CNN (ввод, несколько сверточных слоёв, слой подвыборки, полносвязный слой, выход). Результат: файл model.py с определением модели.

4. Компиляция и обучение модели. Сохранение обученной модели и графика обучения (точность и потеря). Результат: файл trained_model.joblib и графики accuracy.png, loss.png.

5. Оценка модели на тестовом наборе, построение матрицы ошибок и отчёта о метриках. Результат: файл metrics.csv и изображение confusion_matrix.png.

6. Подготовка отчёта и упаковка всех материалов в архив. Результат: архив project_cnn.zip, содержащий код, данные, графики и отчёт.

Методические указания по ходу выполнения работы:

Для всех заданий используйте Python 3.x, среды LibreOffice для отчёта и Git для контроля версии.

Установите необходимые библиотеки командой: pip install numpy pandas matplotlib scikit-learn tensorflow joblib.

В задании 1 создайте файл requirements.txt со списком установленных пакетов и скрипт preprocess.py, который загружает датасет MNIST из встроенных источников библиотеки tensorflow и сохраняет массивы в формате numpy.

В задании 2 в скрипте preprocess.py выполните нормализацию пикселей к диапазону [0,1] и разделите данные на обучающую (80%) и тестовую (20%) части, результаты сохраните в файлы train.npy, test.npy, labels.npy.

В задании 3 в файле model.py опишите модель CNN, используя слои Conv2D, MaxPooling2D, Flatten и Dense, задав количество фильтров, размер ядра и функции активации.

В задании 4 в отдельном скрипте train.py импортируйте модель, скомпилируйте её с оптимизатором Adam и функцией потерь categorical_crossentropy, запустите обучение на 5 эпох, сохраняйте модель функцией joblib.dump в файл trained_model.joblib. Постройте графики точности и функции потерь по эпохам, сохраните их через plt.savefig('accuracy.png') и plt.savefig('loss.png').

В задании 5 в скрипте evaluate.py загрузите обученную модель, выполните предсказание на тестовом наборе, вычислите метрики точности, полноты и F1, сформируйте таблицу метрик в CSV (metrics.csv) и визуализируйте матрицу ошибок, сохранив её в файл confusion_matrix.png.

В задании 6 подготовьте отчёт в формате ODT или PDF, включив титульный лист, описание каждой задачи, скриншоты кода (файлы .py), графики и таблицы метрик. Упакуйте весь проект (директорию с кодом, данными, графиками, отчётом) в архив ZIP.

Формат сдачи: архив ZIP, названный project_cnn.zip, должен содержать папки src (исходный код), data (файлы .npy и .csv), visuals (графики) и report (отчёт).

Отчёт ODT или PDF со следующей структурой: титульный лист (название работы, ФИО, группа, дата), цель работы, краткое описание каждого задания, скриншоты исходных файлов .py, сохранённые графики (accuracy.png, loss.png, confusion_matrix.png), таблица метрик (metrics.csv), выводы и возможные пути улучшения модели.

Сдача: архив ZIP (project_cnn.zip) отправить по электронной почте преподавателю не позднее конца учебного дня, указанного в расписании.

Тема практической работы №11. Реализация рекуррентной нейронной сети для анализа временных рядов, объем часов 6

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя

Цель практической работы:

Научиться самостоятельно реализовывать рекуррентную нейронную сеть для анализа временных рядов: подготовка данных, построение модели, обучение, оценка, сохранение артефактов и оформление отчёта.

Задание(я):

1. Подготовка рабочего окружения. Установить необходимые Python-библиотеки (numpy, pandas, matplotlib, scikit-learn, tensorflow). Создать проектную папку, в которой будут подпапки data, src, models, reports, plots.

2. Загрузка и предобработка датасета. С помощью pandas загрузить открытый датасет «AirPassengers» (или аналогичный временной ряд). Выполнить проверку пропусков, преобразовать даты в числовой формат, нормализовать значения, сформировать обучающие и тестовые последовательности фиксированной длины (window). Сохранить подготовленный набор в файл CSV в папку data.

3. Реализация модели RNN. В папке src создать файл rnn_model.py. Описать модель с использованием keras.layers.SimpleRNN (или LSTM) и Dense-слоя для предсказания следующего значения. Сохранить код в .py файл. Добавить функции для построения модели и её компиляции (optimizer='adam', loss='mse').

4. Обучение модели и визуализация процесса. Запустить обучение модели на подготовленных данных, указав количество эпох и размер батча. В процессе обучения сохранять историю (loss) и после завершения построить график изменения функции потерь. Сохранить график в формате PNG в папку plots. Сохранить обученную модель в файл (например, model.h5) в папку models.

5. Оценка модели, сохранение артефактов и подготовка отчёта. Выполнить предсказание на тестовом наборе, вычислить метрики (MAE, RMSE). Сохранить результаты предсказаний и метрик в CSV-файле в папку reports. Сформировать отчёт в ODT или PDF: титульный лист, описание каждого задания, скриншоты кода, графики, таблицы с метриками и выводы. Сохранить отчёт в папку reports.

Методические указания по ходу выполнения работы:

Для всех заданий используйте терминал/консоль: pip install numpy pandas matplotlib scikit-learn tensorflow.

Создайте структуру проекта заранее, чтобы каждый тип артефактов имел своё место.

При загрузке датасета проверьте наличие заголовков и корректность формата даты, при необходимости используйте pandas.to_datetime.

Для формирования окон (window) используйте скользящее окно фиксированной длины, каждое окно – вход, следующее значение – цель.

В файле rnn_model.py оформите функции: build_model(input_shape) → модель, train_model(model, X_train, y_train) → история обучения.

Сохраняйте историю обучения через pickle или напрямую из объекта History, а график функции потерь создавайте с помощью matplotlib и сохраняйте через plt.savefig('plots/loss.png').

Модель сохраняйте функцией model.save('models/rnn_model.h5').

Для оценки вычислите MAE и RMSE через scikit-learn.metrics, результаты запишите в CSV (reports/metrics.csv).

Отчёт оформляйте в LibreOffice Writer: титульный лист (название работы, ФИО, группа, дата), разделы «Цель», «Ход работы», «Результаты», «Выводы». Вставьте скриншоты кода (из src), графики (из plots) и таблицы метрик.

После завершения упакуйте весь проект (все папки) в один архив ZIP и назовите его согласно шаблону: PM01_RNN_Work.zip.

Формат отчёта: ODT или PDF. Структура: титульный лист; цель работы; описание каждого задания с указанием времени; скриншоты исходного кода; графики (PNG) с подписями; таблицы результатов (CSV) с метриками; выводы и обсуждение полученных результатов.

Сдача: архив ZIP с полной структурой проекта (data, src, models, plots, reports) отправить по электронной почте преподавателю не позднее конца занятия.

Тема практической работы №12. Проектирование архитектуры ИИ-системы с учетом модульности и масштабируемости, объем часов 6

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя

Цель практической работы:

Научиться самостоятельно проектировать модульную и масштабируемую архитектуру ИИ-системы, реализовывать её компоненты, писать тесты и упаковывать в Docker-контейнер.

Задание(я):

1. Сформулировать требования к простой ИИ-системе (например, классификатор на датасете Iris) и нарисовать схему модульной архитектуры. Результат: текстовый файл requirements.txt и схематический рисунок (png).

2. Реализовать два основных модуля системы: модуль предобработки данных и модуль модели (использовать scikit-learn). Результат: два Python-файла (preprocess.py, model.py) и сохранённый файл модели (model.joblib).

3. Написать набор юнит-тестов для функций предобработки и обучения модели с использованием pytest. Результат: файл test_suite.py и файл с результатами запуска тестов (test_report.txt).

4. Подготовить Docker-образ, включающий все зависимости и запуск модели через простейший Flask-API. Результат: Dockerfile, файл запуска (app.py) и скриншот вывода команды docker run (png).

5. Оформить итоговый отчёт, включив описание требований, схему архитектуры, фрагменты кода (скриншоты), результаты тестов и инструкцию по запуску Docker-контейнера. Результат: документ ODT или PDF.

Методические указания по ходу выполнения работы:

В задании 1 используйте любой графический редактор (LibreOffice Draw, draw.io) для создания схемы; сохраните её как PNG в папку docs.

В задании 2 создайте отдельную папку src; каждый модуль сохраняйте в отдельный .py файл; модель сохраняйте функцией joblib.dump в файл model.joblib в папку models.

В задании 3 разместите тесты в папку tests; выполните их командой pytest и перенаправьте вывод в файл test_report.txt внутри папки reports.

В задании 4 создайте Dockerfile в корне проекта; в нём укажите базовый образ python, установку зависимостей (pip install -r requirements.txt) и команду запуска Flask-приложения; соберите образ и запустите контейнер, сделайте скриншот вывода и сохраните его в папку docs.

В задании 5 оформите отчёт согласно шаблону: титульный лист, цель работы, описание каждого задания, схемы и скриншоты, выводы. Сохраните документ как report.odt (или report.pdf).

Формат сдачи: один ZIP-архив, содержащий папки src, tests, models, docs, reports, файл requirements.txt, Dockerfile, файл отчёта и любые дополнительные файлы.

Перед упаковкой убедитесь, что в архиве нет лишних файлов (например, .git, __pycache__).

Отчёт оформляется в ODT или PDF, содержит титульный лист, цель работы, описание каждого задания, схемы архитектуры (PNG), скриншоты кода и вывода тестов, таблицу с результатами тестов, инструкцию по запуску Docker-контейнера и выводы.

Сдача: упаковать всё содержимое в ZIP-архив и отправить по электронной почте преподавателю не позднее конца занятия.

Тема практической работы №13. Контейнеризация ИИ-модели с использованием Docker, объем часов 6

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя

Цель практической работы:

Освоить навыки самостоятельного контейнеризирования простой ИИ-модели с помощью Docker, включая подготовку окружения, написание Docker-файла, сборку образа, запуск контейнера и базовое тестирование.

Задание(я):

1. Обучить простую модель (например, классификатор на датасете Iris) и сохранить её в файл model.pkl.

2. Создать скрипт predict.py, который загружает модель и делает предсказания по входным данным из CSV.

3. Сформировать Docker-файл, описывающий образ со всеми зависимостями и точкой входа.

4. Собрать Docker-образ и запустить контейнер, проверив работу предсказания.

5. Добавить набор unit-тестов для функции предсказания, включить их в образ и выполнить внутри контейнера.

6. Подготовить отчёт, включив скриншоты, файлы проекта и результаты тестов.

Методические указания по ходу выполнения работы:

В задании 1 используйте библиотеку scikit-learn, обучите модель на датасете Iris и сохраните её с помощью joblib.dump('model.pkl').

В задании 2 создайте скрипт, который читает CSV-файл, загружает модель из model.pkl и выводит предсказания в консоль; сохраните файл как predict.py.

В задании 3 напишите Docker-файл, базирующийся на официальном образе python:3.10-slim, скопируйте файлы проекта, установите зависимости

(pip install scikit-learn pandas joblib), задайте точку входа на predict.py; сохраните как Dockerfile.

В задании 4 выполните сборку образа командой docker build -t iris-model . и запустите контейнер командой docker run --rm -v $(pwd)/data:/app/data iris-model python predict.py data/input.csv; сохраните вывод предсказаний в файл predictions.txt.

В задании 5 создайте файл test_predict.py с unit-тестами, используя pytest; добавьте запуск тестов в Docker-файл (RUN pytest); сохраните результаты тестов в файл test_report.txt.

В задании 6 подготовьте отчёт в формате ODT или PDF: титульный лист, цель, описание выполнения каждого задания, скриншоты терминала, содержимое ключевых файлов (Dockerfile, predict.py, test_report.txt), выводы. Упакуйте весь проект (src/, Dockerfile, model.pkl, data/, test_report.txt, отчёт) в архив ZIP.

Формат сдачи: архив ZIP, содержащий все исходные файлы, модель, Docker-файл, тесты и отчёт.

Отчёт ODT или PDF: титульный лист, цель работы, список заданий, подробное описание выполнения, скриншоты команд и выводов, содержимое Dockerfile, predict.py, test_report.txt, выводы о процессе контейнеризации.

Сдать архив ZIP с кодом, моделью, Docker-файлом, тестами и отчётом по электронной почте преподавателя до конца занятия.

Тема практической работы №14. Развертывание ИИ-системы в Kubernetes, объем часов 8

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и

подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя

Цель практической работы:

Научиться самостоятельно развертывать готовую ИИ-систему в Kubernetes, включая контейнеризацию, написание манифестов, деплой и проверку работоспособности.

Задание(я):

1. Подготовить Docker-образ с простым ИИ-моделью (например, скрипт, использующий scikit-learn и датасет Iris).

2. Опубликовать полученный образ в локальном реестре Docker (Docker Hub или собственный registry).

3. Сформировать манифесты Kubernetes (Deployment, Service, ConfigMap) для развертывания образа.

4. Установить и запустить локальный кластер Kubernetes (Minikube или Kind).

5. Применить манифесты, проверить доступность сервиса и корректность работы модели через HTTP-запрос.

6. Реализовать простейший набор тестов (pytest) для проверки эндпоинта и собрать логи/метрики. Сохранить результаты в CSV и скриншоты.

Методические указания по ходу выполнения работы:

В задании 1 создайте файл Dockerfile, в котором описаны все зависимости (python, pip, scikit-learn). Сохраните Dockerfile в папке "docker" и соберите образ, указав тег "ai-model:latest".

В задании 2 выполните вход в выбранный реестр, загрузите образ и зафиксируйте команду push в текстовом файле "push_log.txt". Сохраните его в корневой каталог проекта.

В задании 3 подготовьте YAML-файлы: deployment.yaml, service.yaml, configmap.yaml. Поместите их в папку "k8s". В каждом файле укажите имя образа, количество реплик и параметры порта. Сохраните манифесты без изменений.

В задании 4 установите Minikube (или Kind) согласно официальной инструкции, запустите кластер и проверьте его состояние командой "kubectl cluster-info". Сохраните вывод в файл "cluster_info.txt".

В задании 5 примените манифесты командой "kubectl apply -f k8s/". После деплоя выполните проверку доступности сервиса (curl или браузер) и зафиксируйте ответ в файл "service_response.txt". Сделайте скриншот окна терминала с успешным запросом и сохраните его как "response.png" в папке "report".

В задании 6 напишите набор unit-тестов для функции предсказания модели (используйте pytest). Запустите тесты, сохраните результат в файл "test_report.txt". Соберите логи контейнера (kubectl logs) и экспортируйте метрики (например, время отклика) в CSV-файл "metrics.csv". Сохраните все файлы в папке "tests".

Отчет в формате ODT или PDF. Структура отчета: титульный лист (название работы, ФИО, группа), цель практики, описание каждого задания с указанием времени, скриншоты (response.png, metrics.png), содержимое ключевых файлов (Dockerfile, YAML-манифесты, test_report.txt), выводы и оценка выполненных задач.

Сдача: упаковать все файлы проекта (папки docker, k8s, tests, report) и готовый отчет в один ZIP-архив. Отправить архив по электронной почте преподавателю до окончания занятия.

МДК 01.02. Разработка мобильных приложений с поддержкой искусственного интеллекта

Тема практической работы №1. Создание первого Android-приложения с базовыми интерфейсами, объем часов 16

У2. Реализовывать программные модули на основе требований технического задания.Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У3. Оформлять код в соответствии с принятыми стандартами и требованиями.Документировать разработанный программный код. Применять соглашения о наименованиях переменных, функций и классов (например, PEP8 для Python).

У4. Работать с системами контроля версий для управления проектами (Git, GitLab).Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

Цель практической работы:

Научиться самостоятельно создавать простое Android-приложение с базовым пользовательским интерфейсом, писать чистый поддерживаемый код и фиксировать проект в системе контроля версий.

Задание(я):

1. Установить Android Studio, создать новый проект с пустой Activity. Сохранить проект в папку Project1.

2. Создать XML-разметку на ConstraintLayout с поддержкой альбомной/портретной ориентации (layout-land), добавить стили в themes.xml, локализацию (английский/русский), оптимизировать изображения через векторные Drawable. Сохранить как activity_main.xml.

3. Вынести логику в MainViewModel с LiveData, добавить валидацию ввода, сохранение состояния при повороте экрана, анимацию изменения текста (ObjectAnimator), ripple-эффект для кнопки. Написать 2 unit-теста (JUnit/Espresso).

4. Настроить ktlint и detekt, добавить KDoc-комментарии, проанализировать память/CPU через Android Profiler, сохранить скриншоты в screenshots/, вывод — в profiler_results.txt. Настроить GitHub Actions для автоматических проверок.

5. Инициализировать репозиторий, создать ветки (feature/validation, feature/animations), разрешить конфликт слияния, выполнить rebase, добавить тег v1.0. Сохранить лог с графом: git log --graph --oneline --all > git_log.txt.

6. Настроить buildTypes (debug/release), включить R8-минификацию, сгенерировать подписанный Release APK. Проанализировать размер APK до/после оптимизации. Сохранить как app-release.apk.

7. Добавить в отчёт: скриншоты адаптивного UI, граф памяти из Profiler, примеры локализации, анализ конфликта в Git, сравнение размера APK до/после оптимизации. Сохранить как report.odt.

Методические указания по ходу выполнения работы:

В задании 1 скачайте и установите Android Studio, запустите мастер создания проекта, выберите шаблон Empty Activity, язык Kotlin, минимальную SDK 24 (Android 7.0), имя проекта FirstApp. В файле build.gradle через интерфейс Android Studio добавьте зависимости для ViewModel и Material Design. Настройте эмуляторы Pixel 5 (API 34), Tablet 10" (API 33) и Nexus S через Device Manager. Сгенерируйте файл .gitignore с шаблоном Android через сервис gitignore.io и сохраните в корень проекта.

В задании 2 откройте файл activity_main.xml в папке res/layout, замените LinearLayout на ConstraintLayout через контекстное меню Convert layout. Создайте папку layout-land в res, скопируйте в нее activity_main.xml и скорректируйте расположение элементов для альбомной ориентации. В themes.xml задайте цветовую схему Material Theme через атрибуты colorPrimary и colorSecondary. Для локализации создайте папки values-ru и values-en, добавьте в каждую файл strings.xml с переводами всех текстовых строк. Оптимизируйте иконки через меню New → Vector Asset, сохраните результат в res/drawable.

В задании 3 создайте класс MainViewModel.kt в папке java, наследуйте его от ViewModel, используйте LiveData для управления состоянием текста. В MainActivity.kt привяжите ViewModel через ViewModelProvider, реализуйте валидацию ввода (ограничение длины 50 символов, запрет пустых значений), сохранение состояния через метод onSaveInstanceState. Для анимации текста используйте ObjectAnimator через интерфейс Animation Editor, для кнопки задайте ripple-эффект через атрибут android:background с выбором системного стиля ?attr/selectableItemBackground. Создайте unit-тесты в папке test через меню New → Kotlin File, используя шаблоны JUnit4 и Espresso.

В задании 4 добавьте плагины ktlint и detekt в build.gradle через интерфейс Android Studio, запустите форматирование кода командой ./gradlew ktlintFormat в терминале. Добавьте KDoc-комментарии ко всем публичным методам через сочетание клавиш /** + Enter. Запустите Android Profiler через View → Tool Windows → Profiler, зафиксируйте нагрузку на CPU и память при работе с интерфейсом, сохраните скриншоты графиков в папку app/screenshots/. Для GitHub Actions создайте папку .github/workflows в корне проекта, добавьте файл ci.yml с шаблоном Android CI через интерфейс GitHub.

В задании 5 откройте терминал в корне проекта, выполните команды: git init, git add ., git commit -m "Initial commit". Создайте ветки: git branch feature/validation, git branch feature/animations, переключитесь на feature/validation через git checkout, внесите изменения в логику валидации и закоммитьте. Сымитируйте конфликт слияния: измените strings.xml в обеих ветках, разрешите конфликт через визуальный редактор Android Studio при слиянии в main. Выполните git rebase для ветки feature/animations перед слиянием, добавьте тег: git tag v1.0 -m "Release candidate". Сохраните детальный лог: git log --graph --oneline --all > git_log.txt.

В задании 6 в файле build.gradle включите опции minifyEnabled true и shrinkResources true для buildType release. Сгенерируйте подписанный APK через Build → Generate Signed Bundle/APK, создайте новое хранилище ключей с указанием пароля. После сборки проанализируйте размер APK через APK Analyzer (правый клик на файле → Analyze APK), сравните результаты до и после минификации. Сохраните финальный файл как app-release.apk в папку output.

В задании 7 создайте документ report.odt, добавьте титульный лист с названием работы, ФИО, группой и датой. В разделе «Ход выполнения» опишите этапы по каждому заданию с указанием времени. Вставьте скриншоты: адаптивного интерфейса в двух ориентациях, графиков памяти из Profiler, примеров локализации, фрагмента git log с графом ветвления. В выводах укажите, как MVVM упростил тестирование, на сколько процентов уменьшился размер APK после R8, и какие сложности возникли при разрешении Git-конфликта.

Все файлы проекта (папка Project1), APK-файл, скриншоты из папки screenshots/, отчёты (git_log.txt, profiler_results.txt) и документ report.odt упакуйте в один ZIP-архив.

Отчёт в формате ODT/PDF должен содержать: титульный лист, список выполненных заданий с описанием действий, скриншоты экранов с подписями, выводы о соблюдении чистого кода и работе с Git. Все разделы оформляйте заголовками уровня 1–2, изображения вставляйте непосредственно в текст документа.

Сдача: архив ZIP с каталогом проекта, APK-файлом, скриншотами и отчётом отправить по электронной почте преподавателю.

Тема практической работы №2. Разработка пользовательского интерфейса для мобильного приложения, объем часов 16

У2. Реализовывать программные модули на основе требований технического задания.Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У3. Оформлять код в соответствии с принятыми стандартами и требованиями.Документировать разработанный программный код. Применять соглашения о наименованиях переменных, функций и классов (например, PEP8 для Python).

У4. Работать с системами контроля версий для управления проектами (Git, GitLab).Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

Цель практической работы:

Научиться самостоятельно разрабатывать пользовательский интерфейс мобильного приложения на Android с использованием Kotlin, соблюдать принципы чистого кода и оформлять проект в системе контроля версий.

Задание(я):

1. Создать новый проект в Android Studio (шаблон Empty Activity). Настроить минимальную версию API, Gradle-зависимости, эмуляторы и систему контроля версий. Сохранить проект в папку «MobileUI_Project».

2. Спроектировать адаптивный главный экран: создать XML-разметку с поддержкой альбомной/портретной ориентации, стилизацию через themes.xml, локализацию на два языка, оптимизацию ресурсов. Сохранить файлы в res/layout, res/values, res/drawable.

3. Реализовать логику с архитектурой MVVM: вынести бизнес-логику в ViewModel, добавить валидацию ввода, сохранение состояния, анимации и unit-тесты. Обновить MainActivity.kt и связанные файлы.

4. Обеспечить качество кода: настроить ktlint и detekt, добавить документацию KDoc, проанализировать производительность через Android Profiler, настроить CI/CD через GitHub Actions. Ссохранить скриншоты в папку screenshots/ проекта, а краткий анализ добавить в profiler_results.txt и lint_report.txt .

5. Организовать работу с Git: создать ветки для разных фич, разрешить конфликты слияния, применить rebase, добавить теги версий. Сохранить детальный журнал коммитов с графом ветвления в git_log.txt.

Методические указания по ходу выполнения работы:

Для задания 1 откройте Android Studio, выберите New Project → Empty Activity, укажите имя проекта и расположение папки MobileUI_Project, выберите язык Kotlin, установите минимальную версию API 24 (Android 7.0). В файле build.gradle подключите зависимости: androidx.lifecycle:lifecycle-viewmodel-ktx, com.google.android.material:material, androidx.activity:activity-ktx. Настройте эмуляторы: Pixel 5 (API 33), Tablet 10" (API 30), Nexus S (малое разрешение). Сгенерируйте файл .gitignore с шаблоном для Android-проектов через gitignore.io.

Для задания 2 откройте файл разметки activity_main.xml, замените LinearLayout на ConstraintLayout, создайте альтернативную разметку для альбомной ориентации в папке layout-land. Добавьте стили в res/values/styles.xml и themes.xml с использованием Material Design. Реализуйте локализацию: создайте папки values-ru и values-en, добавьте переводы в strings.xml. Оптимизируйте изображения через Vector Asset Studio, сохраните их в res/drawable. Проверьте адаптивность интерфейса на эмуляторах с разным DPI (mdpi, xxhdpi).

Для задания 3 создайте класс MainViewModel.kt с использованием ViewModel и LiveData. В MainActivity.kt привяжите ViewModel, реализуйте валидацию ввода (проверка на пустое поле, ограничение длины текста), сохранение состояния через onSaveInstanceState. Добавьте анимацию появления текста с помощью ObjectAnimator и ripple-эффект для кнопки через атрибут android:background="?attr/selectableItemBackground". Напишите unit-тесты для ViewModel в папке test/java, используя JUnit и Espresso.

Для задания 4 установите ktlint через плагин Gradle, запустите форматирование кода командой ./gradlew ktlintFormat. Настройте detekt для анализа code smell, сохраните отчет в lint_report.txt. Добавьте KDoc-комментарии ко всем публичным методам. Запустите Android Profiler, проанализируйте потребление памяти и CPU при взаимодействии с интерфейсом, сохранить скриншоты в папку screenshots/ проекта, а краткий анализ добавить в profiler_results.txt. Создайте файл .github/workflows/ci.yml для автоматического запуска lint и тестов при пуше в репозиторий.

Для задания 5 в терминале Android Studio выполните: git init, git add ., git commit -m "Initial commit". Создайте ветки: git branch feature/validation, git branch feature/animations, переключитесь на них через git checkout, внесите изменения и закоммитьте. Сымитируйте конфликт слияния: измените один файл в обеих ветках, разрешите конфликт вручную. Слейте ветки в main с использованием git rebase. Присвойте тег версии: git tag v1.0 -m "Stable UI". Сохраните детальный лог: git log --graph --oneline --all > git_log.txt.

Все исходные файлы (activity_main.xml, MainActivity.kt, lint_report.txt, git_log.txt, .github/workflows/ci.yml и другие) и проект Android Studio должны быть упакованы в один ZIP-архив.

Формат сдачи: архив ZIP, содержащий папку MobileUI_Project с полным проектом и отдельные файлы отчётов, а также отчёт в формате ODT или PDF.

Отчёт ODT/PDF: титульный лист (название работы, ФИО, группа, дата), список задач с описанием выполненных действий, скриншоты главного экрана приложения и окна lint-отчёта, выводы о соблюдении чистого кода и работе с Git, приложение-приложение (приложенный ZIP-архив).

Сдача: подготовьте ZIP-архив с проектом и отчётом, отправьте его по электронной почте преподавателю.

Тема практической работы №3. Внедрение TensorFlow Lite модели в Android-приложение, объем часов 16

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

Цель практической работы:

Научиться самостоятельно внедрять TensorFlow Lite модель в Android-приложение, выполнять предобученную модель на устройстве и сохранять результаты работы.

Задание(я):

1. Установить Android Studio, создать новый проект с Empty Activity, настроить SDK. Сохранить скриншот настроек в файл setup.png.

2. Сконвертировать предобученную модель TensorFlow (например, MobileNetV2) в формат .tflite с помощью TensorFlow Lite Converter. Сохранить полученный файл model.tflite в папку app/src/main/assets.

3. Добавить зависимости TensorFlow Lite в файл build.gradle (implementation 'org.tensorflow:tflite:2.9.0'). Сохранить изменённый build.gradle как gradle_mod.txt.

4. Реализовать класс-обёртку для загрузки модели (Interpreter) и выполнения инференса над изображением, полученным из галереи. Сохранить исходный код в файл TFLiteHelper.py (или .java).

5. Создать UI: кнопка «Выбрать изображение», ImageView для показа выбранного фото и TextView для вывода результатов. Сохранить разметку layout.xml и скриншот экрана в ui.png.

6. Реализовать обработку результата: вывести топ-3 предсказания с вероятностями в TextView. Сохранить журнал логов в файл log.txt.

7. Провести тестирование на реальном устройстве или эмуляторе: проверить работу модели, измерить время инференса, зафиксировать показатели в таблице results.csv.

8. Оптимизировать модель (применить quantization) и сравнить время инференса с исходной моделью. Сохранить оптимизированный файл model_quant.tflite и обновлённую таблицу results.csv.

9. Подготовить отчёт: титульный лист, описание задач, скриншоты, таблицы, выводы. Сохранить как report.odt.

10. Упаковать всё в архив: исходный код (.java/.kt), файлы .tflite, .csv, .png, .txt, report.odt.

Методические указания по ходу выполнения работы:

В задании 1 откройте Android Studio, создайте новый проект и запишите параметры SDK; сделайте скриншот окна Settings и сохраните как setup.png.

В задании 2 используйте консольный инструмент TensorFlow Lite Converter для конвертации модели; поместите полученный model.tflite в папку assets проекта.

В задании 3 откройте файл build.gradle (модуль app) и добавьте строку зависимости; сохраните изменённый файл как gradle_mod.txt.

В задании 4 создайте отдельный класс, который будет инициализировать Interpreter, загружать модель из assets и принимать Bitmap для предсказания; сохраните файл с кодом.

В задании 5 откройте редактор разметки, добавьте необходимые элементы UI, сохраните layout.xml и сделайте скриншот работающего интерфейса (ui.png).

В задании 6 реализуйте вывод результатов в TextView, запишите лог работы в файл log.txt (используйте Logcat).

В задании 7 запустите приложение на устройстве, выполните несколько предсказаний, измерьте время выполнения (можно вывести в Logcat) и запишите данные в results.csv.

В задании 8 проведите повторную конвертацию модели с применением post-training quantization, замените модель в assets, повторите измерения и дополните results.csv.

В задании 9 подготовьте отчёт согласно шаблону: титульный лист, цель, список задач, скриншоты, таблицы результатов, выводы; сохраните как report.odt.

В задании 10 соберите все перечисленные файлы в один ZIP-архив и проверьте его целостность.

Отчёт должен включать титульный лист, цель работы, список выполненных заданий, скриншоты (setup.png, ui.png), таблицу results.csv, лог file log.txt, выводы о производительности модели. Формат – ODT или PDF.

Сдача: архив ZIP с кодом, моделями, данными, скриншотами и отчётом отправить по email преподавателю до конца занятия.

Тема практической работы №4. Оптимизация ИИ-модели для мобильного устройства, объем часов 18

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

Цель практической работы:

Освоить навыки самостоятельной оптимизации модели искусственного интеллекта для мобильных устройств с помощью TensorFlow Lite, включая измерение размеров, времени инференса и применение пост-тренировочного квантизации.

Задание(я):

1. Подготовка среды разработки (установка Python, pip, необходимые библиотеки). Ожидаемое время – 3 часа.

2. Загрузка предобученной модели (MobileNetV2) в формате TensorFlow, измерение её исходного размера и времени инференса на тестовом изображении.

3. Конвертация модели в формат TensorFlow Lite с базовыми настройками. Сохранить файл model.tflite.

4. Применение пост-тренировочного квантизации (float16 и int8) к полученной TFLite-модели, сохранение файлов model_float16.tflite и model_int8.tflite.

5. Оценка и сравнение размеров файлов и времени инференса для всех трёх вариантов (исходная, float16, int8). Экспортировать результаты в CSV (metrics.csv) и построить графики (size_plot.png, latency_plot.png).

6. Создание небольшого скрипта для запуска инференса TFLite-модели на том же тестовом изображении и сохранение вывода в файл output.txt.

7. Подготовка отчёта и упаковка всех материалов (исходный код .py, CSV, графики, модели, скрипт, текстовый вывод) в архив.

Методические указания по ходу выполнения работы:

В задании 1 установите Python 3.9+, выполните pip install tensorflow==2.12 numpy pandas matplotlib joblib. Сохраните список установленных пакетов в файл requirements.txt.

В задании 2 используйте API TensorFlow для загрузки модели MobileNetV2 без верхних слоёв. Запишите размер файла модели (в мегабайтах) и измерьте среднее время инференса на 10 запусков, используя модуль time. Сохраните результаты в переменные и экспортируйте их в CSV.

В задании 3 выполните конвертацию модели в TFLite с помощью TFLiteConverter. Сохраните полученный файл как model.tflite в папку results.

В задании 4 примените две стратегии квантизации: float16 и full integer (int8). Для каждой стратегии сохраните отдельный файл в папку results.

В задании 5 измерьте размер каждого .tflite файла и время инференса (10 запусков) аналогично заданию 2. Запишите все метрики в таблицу CSV (колонки: model_type, size_MB, latency_ms). Постройте два графика: размер модели vs тип и время инференса vs тип, сохраните их как PNG в папку results.

В задании 6 создайте скрипт, который загружает выбранную TFLite-модель, выполняет предсказание на том же тестовом изображении и сохраняет полученный вектор предсказаний в файл output.txt в папку results.

В задании 7 подготовьте отчёт в формате ODT или PDF. Структура отчёта: титульный лист, цель работы, описание каждого задания, скриншоты результатов (таблицы, графики, вывод скрипта), выводы о влиянии квантизации на размер и скорость. Вложите в архив ZIP все .py файлы, CSV, PNG, .tflite модели, requirements.txt и готовый отчёт.

Формат отчёта: ODT или PDF с титульным листом, разделом «Цель работы», описанием всех заданий, скриншотами таблиц и графиков, разделом «Выводы».

Сдача: упаковать все материалы в архив ZIP и отправить по email преподавателю до конца занятия.

Тема практической работы №5. Разработка мобильного приложения для распознавания изображений, объем часов 14

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ**.**

Цель практической работы:

Научиться самостоятельно разрабатывать простое Android-приложение, интегрировать в него модель машинного обучения для распознавания изображений, проводить отладку и логирование, а также оформлять результаты работы.

Задание(я):

1. Создать новый проект в Android Studio, выбрать пустую Activity и настроить базовые параметры проекта.

2. Спроектировать пользовательский интерфейс: добавить кнопку захвата изображения, элемент ImageView для показа снимка и TextView для вывода результата распознавания.

3. Подготовить предобученную модель TensorFlow Lite для распознавания цифр (датасет MNIST) и разместить её в каталоге assets проекта.

4. Добавить в проект зависимости TensorFlow Lite (через Gradle) и настроить проект для работы с библиотекой.

5. Реализовать функциональность захвата изображения с помощью камеры устройства и предобработку изображения (масштабирование, преобразование в оттенки серого).

6. Интегрировать модель TensorFlow Lite: загрузить модель, выполнить инференс на предобработанном изображении и отобразить полученную метку в TextView.

7. Встроить механизм логирования (Logcat) для записи ключевых этапов работы приложения (загрузка модели, захват изображения, результат инференса).

8. Провести отладку приложения: установить точки останова, пошагово выполнить код, исправить выявленные ошибки, проверить корректность вывода результата.

9. Сохранить артефакты работы: скриншоты интерфейса, файл модели .tflite, исходные файлы .kt/.java, журнал логов, сгенерировать APK.

Методические указания по ходу выполнения работы:

В задании 1 откройте Android Studio, выберите "New Project", задайте имя проекта, пакет и минимальную версию SDK, подтвердите создание.

В задании 2 откройте layout-файл, разместите элементы управления согласно макету, задайте идентификаторы, сохраните файл.

В задании 3 скачайте предобученную модель TensorFlow Lite для MNIST (файл .tflite) и поместите её в папку app/src/main/assets, убедитесь, что файл включён в сборку.

В задании 4 откройте файл build.gradle (модуль), добавьте зависимости implementation 'org.tensorflow:tensorflow-lite:2.x.x' и sync проект.

В задании 5 реализуйте запрос к камере через Intent, полученный bitmap преобразуйте к требуемому формату (28×28, серый), сохраните промежуточный файл в кэш.

В задании 6 загрузите модель из assets с помощью Interpreter, подготовьте входной тензор, выполните интерпретацию, получите массив вероятностей и определите класс с максимальной вероятностью, отобразите результат.

В задании 7 используйте класс Log для вывода сообщений о каждом важном этапе (например, Log.d("MyApp", "Model loaded")); проверьте вывод в Logcat.

В задании 8 откройте окно Debug, установите брейкпоинты в ключевых методах, выполните приложение в режиме отладки, проанализируйте стек вызовов и исправьте ошибки.

В задании 9 сделайте скриншоты экрана с отображением результата, сохраните их в формате PNG, экспортируйте журнал Logcat в файл .txt, упакуйте все исходные файлы, модель и APK в один ZIP-архив.

Формат сдачи: архив ZIP, содержащий каталог src с кодом, папку assets с моделью, папку res с ресурсами, файл отчёта, скриншоты и лог-файл.

Формат отчета: ODT или PDF. Структура отчёта – титульный лист (название практики, ФИО, группа, дата), цель работы, краткое описание выполненных заданий, пошаговое описание реализации без кода, скриншоты интерфейса (вставить PNG), выдержки из журнала Logcat, выводы о работе приложения и о процессе отладки.

Сдача: подготовьте ZIP-архив с кодом, моделью, скриншотами, журналом и отчётом, отправьте его по электронному адресу преподавателя не позднее последнего занятия.

Тема практической работы №6. Внедрение голосового помощника на основе ИИ в мобильное приложение, объем часов 14

У2. Реализовывать программные модули на основе требований технического задания. Писать чистый, понятный и поддерживаемый код. Использовать стандартные библиотеки и фреймворки для ускорения разработки.

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

Цель практической работы:

Научиться самостоятельно внедрять голосового помощника на основе ИИ в Android-приложение, используя модульный подход, отладку и логирование.

Задание(я):

1. Создать новый проект Android Studio (один экран).

2. Добавить в манифест разрешения на доступ к микрофону и интернет.

3. Реализовать модуль распознавания речи: создать класс-обработчик, настроить SpeechRecognizer, обеспечить передачу результата в главный поток.

4. Разработать простой модуль генерации ответа (правила-ответы) без внешних сервисов, оформить его как отдельный класс.

5. Интегрировать TextToSpeech: создать модуль озвучивания ответа, обеспечить корректное освобождение ресурсов.

6. Встроить логирование: добавить записи в Logcat на каждом этапе (получение аудио, распознавание, формирование ответа, озвучивание).

7. Провести отладку: установить точки останова, выполнить пошаговый запуск, зафиксировать найденные ошибки и их исправления.

8. Тестировать приложение на реальном устройстве или эмуляторе, собрать скриншоты работы (активный ввод, распознанный текст, ответ).

9. Подготовить отчёт: включить титульный лист, описание выполненных заданий, скриншоты, лог-выводы, выводы о проделанной работе.

Методические указания по ходу выполнения работы:

Для задания 1 создайте проект в Android Studio, выберите язык Kotlin и минимальный SDK 21.

В задании 2 откройте файл AndroidManifest.xml и внесите необходимые <uses-permission> элементы.

В задании 3 оформите класс SpeechModule, реализующий интерфейс RecognitionListener, и зарегистрируйте его в Activity.

В задании 4 создайте класс ResponseEngine с методом getResponse(String) возвращающим строку-ответ по простым правилам.

В задании 5 создайте класс TTSModule, инициализируйте TextToSpeech в onCreate, реализуйте метод speak(String).

В задании 6 используйте android.util.Log для записи сообщений с тегом "VoiceAssistant" в каждом ключевом методе.

В задании 7 откройте Debugger, установите breakpoints в методах SpeechModule, ResponseEngine и TTSModule, выполните пошаговое выполнение, исправьте обнаруженные проблемы.

В задании 8 запустите приложение, произнесите несколько фраз, сделайте скриншоты экрана и сохраните их в папку screenshots/.

В задании 9 оформите отчёт в LibreOffice Writer: титульный лист, список заданий, описание реализации, скриншоты, лог-выводы, выводы. Сохраните как PDF.

Все исходные файлы (.kt), файлы манифеста, скриншоты и лог-файлы должны быть упакованы в один ZIP-архив.

Формат сдачи: архив ZIP, содержащий папку src/ с кодом, папку resources/ с ресурсами, папку logs/ с логами, папку screenshots/ и файл отчёт.pdf.

Отчёт в формате ODT или PDF: титульный лист, цель и задачи, описание реализации модулей, скриншоты работы, фрагменты логов, выводы и список использованных команд (pip install, gradle).

Сдать архив ZIP с кодом, данными и отчётом через электронную почту преподавателя не позднее конца учебного дня, указанного в расписании.

Тема практической работы №7. Автоматизация тестирования мобильного ИИ-приложения, объем часов 12

У4. Работать с системами контроля версий для управления проектами (Git, GitLab). Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя.

Цель практической работы:

Научиться самостоятельно разрабатывать и выполнять автоматизированные тесты для Android-приложения с ИИ-моделью, используя Git для управления версиями и фиксировать результаты в отчёте.

Задание(я):

1. Создать локальный Git-репозиторий, выполнить начальное коммит-сообщение и настроить ветку develop. Результат: файл README.md и .gitignore, сохранённые в каталоге проекта.

2. Сгенерировать простое Android-приложение (один экран) с встроенной ИИ-моделью (например, классификатор изображений). Добавить исходный код в ветку feature/app. Результат: исходные файлы .java/.kt и layout-файлы, сохранённые в каталоге app/src.

3. Написать набор unit-тестов для бизнес-логики (класс-модель, функции предобработки). Поместить тесты в каталог test. Результат: файл с тестами и отчёт о выполнении (JUnit XML), сохранённые в каталоге test_results.

4. Настроить UI-тесты с использованием Espresso, охватить основные сценарии (запуск приложения, ввод данных, получение предсказания). Результат: файлы UI-тестов и скриншоты успешных запусков, сохранённые в каталоге ui_test_results.

5. Запустить все тесты (unit + UI), собрать метрики покрытия кода, построить график покрытия и сохранить его в файл coverage.png. Сохранить итоговый artefact-файл (coverage.png) и файл с результатами (results.csv). Результат: файлы coverage.png, results.csv, joblib-модель (если требуется), сохранённые в корневом каталоге проекта.

Методические указания по ходу выполнения работы:

Для задания 1 используйте команды git init, git add, git commit, git branch и git checkout. После создания ветки develop сделайте первый push в удалённый репозиторий (можно локальный bare-репозиторий).

Для задания 2 создайте новый проект в Android Studio, выберите шаблон "Empty Activity". Добавьте файл модели (например, .tflite) в папку assets и подключите его в коде приложения. Сохраните все изменения и сделайте commit в ветку feature/app.

Для задания 3 откройте каталог test, создайте класс тестов, используйте JUnit 5. Запустите тесты через Gradle и экспортируйте результаты в формате XML. Сохраните файл XML в папку test_results.

Для задания 4 в каталоге androidTest создайте тестовый класс, используйте Espresso для эмуляции ввода пользователем. При каждом прохождении теста делайте скриншот экрана и сохраняйте его в папку ui_test_results. После завершения тестов соберите их в один отчёт.

Для задания 5 запустите команду gradle test и gradle connectedAndroidTest, соберите покрытие кода с помощью JaCoCo, экспортируйте отчёт в формат CSV и постройте график покрытия (используйте matplotlib, сохраните график командой plt.savefig('coverage.png')). Сохраните все артефакты в корневом каталоге проекта.

Формат сдачи: подготовьте архив ZIP, включающий весь каталог проекта, файлы README.md, .gitignore, результаты тестов (XML, CSV), график coverage.png, а также отчёт в формате ODT или PDF.

Формат отчёта: ODT или PDF, включающий титульный лист (название работы, ФИО, группа, дата), описание каждого задания, скриншоты (интерфейс приложения, результаты тестов, график coverage.png), таблицу с результатами (results.csv) и выводы о покрытии и стабильности приложения.

Сдача: упаковать все файлы в архив ZIP и отправить по электронной почте преподавателю до конца занятия.

Тема практической работы №8. Развертывание мобильного приложения в Play Market, объем часов 10

У4. Работать с системами контроля версий для управления проектами (Git, GitLab). Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Выполнять настройки окружения и подготовку тестовых данных. Фиксировать результаты выполнения тестов и подготавливать отчеты о результатах тестов. Определять уровень критичности дефектов. Разрабатывать автоматизированные тесты для тестирования модулей и/или отдельных функций. Восстанавливать окружение и тесты после сбоя**.**

Цель практической работы:

Научиться самостоятельно развертывать Android-приложение в Google Play, используя систему контроля версий Git и выполнять базовое тестирование перед публикацией

Задание(я):

1. Создать локальный Git-репозиторий, выполнить первый commit и отправить проект в удалённый репозиторий (GitLab/GitHub). Сохранить снимок репозитория в файл repo_snapshot.zip.

2. Сконфигурировать ветки разработки: main и feature-release. Слить изменения и решить возможные конфликты. Сохранить лог слияний в файл merge_log.txt.

3. Сгенерировать keystore, подписать релизный артефакт (AAB) и собрать его через Android Studio. Сохранить файл app_release.aab и информацию о keystore в keystore_info.txt.

4. Подготовить листинг приложения в Google Play Console: название, описание, скриншоты, иконка. Сохранить скриншоты листинга в папку screenshots/.

5. Загрузить AAB в Play Console, заполнить метаданные и отправить приложение на проверку. Сохранить подтверждающий скриншот upload_confirmation.png.

6. Выполнить базовое юнит-тестирование модулей приложения (JUnit). Зафиксировать результаты в файле test_report.csv и сохранить скриншот вывода в test_output.png.

7. Сформировать итоговый отчёт, включающий описание выполненных шагов, скриншоты, логи и выводы. Сохранить отчёт в формате ODT (или PDF) как final_report.odt.

Методические указания по ходу выполнения работы:

Для задания 1 используйте командную строку Git: init, add, commit, remote add, push. После push выполните архивирование каталога проекта.

В задании 2 создайте ветку feature-release, выполните merge в main, при конфликте используйте инструменты Git для их разрешения. Сохраните вывод команды git log в merge_log.txt.

В задании 3 в Android Studio откройте меню Build → Generate Signed Bundle/APK, укажите созданный keystore и соберите AAB. Сохраните keystore-файл в безопасное место и запишите его параметры в keystore_info.txt.

В задании 4 в Google Play Console создайте новый листинг, загрузите необходимые изображения и заполните поля. Сохраните все изображения в папку screenshots/.

В задании 5 загрузите полученный AAB через раздел Release → Production, укажите релиз-ноты и отправьте на проверку. Сфотографируйте страницу подтверждения и сохраните как upload_confirmation.png.

В задании 6 откройте модуль тестов в Android Studio, запустите все JUnit-тесты, экспортируйте результаты в CSV (test_report.csv) и сделайте скриншот окна Run.

В задании 7 подготовьте отчёт согласно шаблону: титульный лист, цель, список задач, описание выполнения, скриншоты, логи, выводы. Сохраните как final_report.odt.

Все полученные файлы (repo_snapshot.zip, merge_log.txt, app_release.aab, keystore_info.txt, screenshots/, upload_confirmation.png, test_report.csv, test_output.png, final_report.odt) упакуйте в один архив ZIP для сдачи.

Отчёт оформляется в ODT (или PDF). Структура: титульный лист, цель практики, список задач, пошаговое описание выполнения, скриншоты (вставленные в текст), логи и таблицы результатов, выводы и рекомендации.

Сдача: упаковать все перечисленные файлы в архив ZIP и отправить по электронной почте преподавателю не позднее окончания 10-часового занятия.

МДК 01.03. Тестирование программных модулей

Тема практической работы №1. Определение целей тестирования для каждого уровня и вида тестирования, объем часов 20

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

У7. Определять критические сценарии работы системы, которые необходимо протестировать. Разрабатывать пошаговые тестовые сценарии на основе требований. Оценивать покрытие тестов и их соответствие техническому заданию.

Цель практической работы:

Научиться самостоятельно определять цели тестирования для различных уровней и видов тестирования, формировать соответствующие тестовые сценарии и оформлять отчёт о тестировании.

Задание(я):

1. Изучить теоретический материал о уровнях (модульное, интеграционное, системное, приемочное) и видах (функциональное, нефункциональное, производительность, регрессионное) тестирования. Составить таблицу-справочник и сохранить её в файл levels_and_types.csv.

2. Выбрать простой учебный проект (например, консольное приложение «Калькулятор»). На основании требований проекта определить цели тестирования для каждого уровня и вида. Оформить цели в документ goals.txt и сохранить.

3. Сформировать набор тест-кейсов для модульного уровня, исходя из целей, определённых в задании 2. Оформить тест-кейсы в таблицу и сохранить в файл unit_test_cases.csv.

4. Сформировать набор тест-кейсов для интеграционного и системного уровней, используя те же цели. Оформить в таблицы integration_test_cases.csv и system_test_cases.csv.

5. Оценить покрытие тестов: построить матрицу покрытий (уровень — вид тестирования — количество кейсов) и сохранить в файл

coverage_matrix.csv. По результатам построить столбчатый график покрытия и сохранить как coverage.png.

6. Подготовить итоговый отчёт: титульный лист, описание выполненных заданий, вложения (скриншоты таблиц и графика), выводы о соответствии целей тестирования требованиям проекта. Сохранить отчёт в формате ODT или PDF.

Методические указания по ходу выполнения работы:

Для всех таблиц используйте формат CSV, разделитель – запятая; первая строка – заголовки колонок.

График сохраняйте функцией plt.savefig('coverage.png') без вывода на экран.

Все исходные файлы (csv, txt, ODT/PDF) помещайте в одну папку проекта.

Код, использованный для построения графика, сохраните в файл plot_coverage.py.

При сдаче упакуйте папку проекта в архив ZIP: в архиве должны быть папка с кодом, все csv-файлы, файл plot_coverage.py, готовый отчёт ODT/PDF.

Не используйте внешние ссылки или репозитории; все необходимые инструменты – Python (pip install matplotlib, pandas) и LibreOffice.

Отчёт состоит из титульного листа, краткого введения, описания каждого задания (1-6) с указанием выполненных действий, скриншотов таблиц CSV и графика coverage.png, раздела «Выводы» с оценкой соответствия целей тестирования требованиям проекта. Формат – ODT или PDF.

Сдача: архив ZIP с кодом, данными и отчётом отправить по электронной почте преподавателю не позднее конца занятия.

Тема практической работы №2. Подготовка тестового пакета и задания на тестирование, объем часов 8

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

У7. Определять критические сценарии работы системы, которые необходимо протестировать. Разрабатывать пошаговые тестовые сценарии на основе требований. Оценивать покрытие тестов и их соответствие техническому заданию.

Цель практической работы:

Научиться самостоятельно разрабатывать тестовый пакет, создавать тестовые сценарии и автоматизировать их выполнение с помощью pytest, а также оформлять результаты тестирования в соответствии с требованиями отчётности.

Задание(я):

1. Подготовка тестового пакета: собрать исходный код проекта (один .py файл) и необходимые данные; оформить структуру каталогов (src, tests, data); сохранить в папку package.

2. Разработка тест-кейсов: составить перечень тестовых сценариев по принципам чёрного ящика (эквивалентное разбиение, граничные значения); оформить их в виде таблицы в файле test_cases.csv.

3. Реализация автоматических тестов: написать модуль unit_tests.py с использованием библиотеки pytest, реализовать тесты согласно разработанным кейсам.

4. Запуск тестов и сбор результатов: выполнить pytest, сохранить вывод в файл test_report.txt; построить график зависимости количества пройденных тестов от категории (сохранить как coverage.png).

5. Оформление отчёта: подготовить документ ODT/PDF с титульным листом, описанием выполненных заданий, скриншотами структуры каталога, содержимым test_cases.csv, фрагментами отчёта test_report.txt и графиком coverage.png; упаковать всё в ZIP-архив.

Методические указания по ходу выполнения работы:

Для задания 1 создайте корневой каталог с подпапками src, tests и data; поместите исходный скрипт в src, тестовые данные (если требуются) в data; сохраните структуру в файл package_structure.txt.

В задании 2 используйте табличный редактор LibreOffice Calc для оформления тест-кейсов; каждый кейс должен включать идентификатор, цель, входные данные, ожидаемый результат и тип теста; экспортируйте таблицу в CSV (test_cases.csv) и разместите её в папке tests.

В задании 3 разработайте тесты в файле unit_tests.py, расположив его в папке tests; используйте конструкции pytest (функции, assert); не забудьте импортировать проверяемый модуль из src.

В задании 4 запустите тесты командой pytest из корневой папки проекта, перенаправив вывод в файл test_report.txt; проанализируйте количество успешных и проваленных тестов, постройте столбчатый график с помощью matplotlib и сохраните его как coverage.png в папке reports.

В задании 5 подготовьте отчёт, включив в него: титульный лист, краткое описание цели и задач, описание структуры пакета, таблицу тест-кейсов, фрагменты отчёта о тестировании, скриншот структуры каталогов, график coverage.png; сохраните документ в формате ODT или PDF.

Для сдачи сформируйте ZIP-архив, содержащий все файлы проекта (src, tests, data, test_cases.csv, unit_tests.py, test_report.txt, coverage.png, отчёт ODT/PDF) и отправьте его преподавателю.

Отчёт оформляется в ODT или PDF, включает титульный лист (название практики, ФИО студента, группа, дата), разделы: цель работы, описание структуры пакета, таблица тест-кейсов (скриншот CSV), описание реализованных тестов, результаты выполнения (фрагмент test_report.txt), график coverage.png, выводы и выводы о покрытии тестов.

Сдача: подготовить ZIP-архив со всеми исходными файлами, данными, отчётом и графиками; отправить архив по электронной почте преподавателю не позднее конца занятия.

Тема практической работы №3. Подготовка тестового сценария, объем часов 6

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

У7. Определять критические сценарии работы системы, которые необходимо протестировать. Разрабатывать пошаговые тестовые сценарии на основе требований. Оценивать покрытие тестов и их соответствие техническому заданию.

Цель практической работы:

Освоить навыки самостоятельного разработки и документирования тестовых сценариев для проверки корректности программных модулей с использованием pytest.

Задание(я):

1. Сформировать тест-план на основе заданного простого Python-модуля (пример: функции сложения и деления). Сохранить план в файл test_plan.txt.

2. Составить набор тест-кейсов, применяя техники «чёрного ящика»: эквивалентное разбиение, граничные значения, отрицательные сценарии. Оформить их в таблице CSV (test_cases.csv).

3. Реализовать тесты в фреймворке pytest, используя подготовленные кейсы. Сохранить код в файле test_module.py.

4. Запустить тесты, собрать результаты, построить график количества пройденных/непройденных тестов и сохранить его как results.png.

5. Сохранить отчёт о тестировании в формате CSV (test_report.csv) и файл с покрытием (coverage.xml).

6. Оформить итоговый отчёт (ODT или PDF) с титульным листом, описанием выполненных заданий, скриншотами вывода pytest, графиком results.png и выводами.

Методические указания по ходу выполнения работы:

В задании 1 изучите требования к модулю, сформулируйте цели тестирования и опишите объём тестирования в документе test_plan.txt.

В задании 2 используйте техники эквивалентного разбиения и анализа граничных значений для определения входных наборов; оформите каждый тест-кейс в строке CSV-файла: ID, описание, входные данные, ожидаемый результат.

В задании 3 создайте директорию проекта, установите pytest (pip install pytest), создайте файл test_module.py и реализуйте функции-обертки, которые читают данные из test_cases.csv и вызывают проверяемые функции.

В задании 4 выполните команду pytest с параметрами для генерации отчёта и сохранения результатов; постройте столбчатый график (например, matplotlib) количества успешных и неуспешных тестов и сохраните его в results.png.

В задании 5 экспортируйте вывод pytest в CSV-файл test_report.csv (используйте опцию --junitxml и преобразуйте в CSV), сохраните файл покрытия coverage.xml (pytest-cov).

В задании 6 подготовьте отчёт: титульный лист (название работы, ФИО, группа), раздел «Описание задач», таблицу тест-кейсов, скриншоты вывода pytest, график results.png, выводы о покрытии и качестве тестов. Сохраните документ в формате ODT или PDF.

Формат сдачи: архив ZIP, содержащий папку проекта с файлами test_plan.txt, test_cases.csv, test_module.py, results.png, test_report.csv, coverage.xml и готовый отчёт (ODT/PDF).

Отчёт оформляется в ODT или PDF, включает титульный лист, описание выполненных заданий, таблицу тест-кейсов, скриншоты вывода pytest, график results.png, таблицу результатов (CSV) и раздел выводов.

Сдать архив ZIP, упакованный согласно указаниям, по электронной почте преподавателю не позже окончания занятия.

Тема практической работы №4. Обучение и прогноз модели логистической регрессии, объем часов 10

У4. Работать с системами контроля версий для управления проектами (Git, GitLab). Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Освоить самостоятельное обучение и оценку модели логистической регрессии, а также навыки управления проектом в Git и написания базовых тестов.

Задание(я):

1. Создать локальный репозиторий Git, выполнить первый commit, создать ветку feature/model и выполнить push.

2. Загрузить открытый датасет Iris, выполнить предобработку: выбрать два класса, преобразовать метки, разделить на обучающую и тестовую выборки.

3. Обучить модель логистической регрессии на обучающей части, сохранить обученную модель в файл model.joblib.

4. Оценить модель: построить матрицу ошибок, вычислить Accuracy, Precision, Recall, F1-score, построить ROC-кривую и сохранить графики в файлы (confusion.png, roc.png).

5. Написать набор unit-тестов для функций предобработки и предсказания модели, выполнить их с помощью pytest, оформить отчет и упаковать всё в архив.

Методические указания по ходу выполнения работы:

В задании 1 инициализируйте репозиторий командой git init, добавьте файлы, сделайте commit, создайте ветку, выполните push в удалённый репозиторий (можно локальный).

В задании 2 загрузите датасет Iris через библиотеку scikit-learn, отберите только два целевых класса, выполните кодирование меток, разделите данные функцией train_test_split, сохраните полученные наборы в CSV-файлы train.csv и test.csv.

В задании 3 обучите логистическую регрессию, используя scikit-learn, затем сохраните модель функцией joblib.dump в файл model.joblib в корне проекта.

В задании 4 расчитайте предсказания на тестовом наборе, сформируйте confusion matrix, вычислите метрики Accuracy, Precision, Recall, F1-score, постройте ROC-кривую, сохраните графики командой plt.savefig('confusion.png') и plt.savefig('roc.png'), экспортируйте таблицу метрик в CSV файл metrics.csv.

В задании 5 напишите тестовые функции в файле test_model.py, проверяющие корректность предобработки и предсказания (используйте pytest), запустите тесты, зафиксируйте результаты в файле test_report.txt. Оформите отчет в ODT или PDF: титульный лист, описание каждого задания, скриншоты кода, графиков и таблиц, выводы.

Для сдачи упакуйте в один ZIP-архив папку проекта, включающую весь исходный код (.py), файлы данных (.csv), модель (.joblib), графики (.png), тесты (.py), отчет (.odt/.pdf) и файл test_report.txt.

Отчет оформляется в ODT или PDF, содержит титульный лист, краткое описание целей, пошаговое описание выполненных заданий, скриншоты кода и результатов (графики, таблицы), таблицу метрик, выводы о качестве модели и о проделанной работе с Git.

Сдать архив ZIP с проектом и отчетом по электронной почте преподавателя не позднее конца занятия.

Тема практической работы №5. Построение и визуализация матрицы ошибок, объем часов 10

У4. Работать с системами контроля версий для управления проектами (Git, GitLab). Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Научиться самостоятельно выполнять построение, визуализацию и интерпретацию матрицы ошибок классификационной модели, а также оформлять результаты и управлять проектом с помощью Git.

Задание(я):

1. Создать локальный репозиторий Git, инициализировать проект, выполнить первый commit (результат: папка .git и файл README.md).

2. Загрузить открытый датасет Iris, сохранить в файл data.csv (результат: data.csv).

3. Реализовать скрипт preprocessing.py для предобработки данных (разделение на обучающую и тестовую выборки, сохранение массивов в файлы X_train.npy, X_test.npy, y_train.npy, y_test.npy).

4. Обучить простой классификатор (Logistic Regression) в файле train_model.py, сохранить обученную модель в model.joblib.

5. Вычислить матрицу ошибок, построить её тепловую карту, сохранить график в confusion_matrix.png.

6. Рассчитать метрики Accuracy, Precision, Recall, F1-score, сохранить таблицу в metrics.csv.

7. Написать набор unit-тестов (test_metrics.py) с использованием pytest, убедиться, что все тесты проходят.

8. Оформить отчёт (report.odt или report.pdf) с титульным листом, описанием каждого задания, скриншотами кода, графика и таблиц, выводами.

9. Скомплектовать весь проект в архив project.zip (код, данные, модели, графики, CSV, тесты, отчёт) и отправить преподавателю.

Методические указания по ходу выполнения работы:

Для задания 1 создайте новую папку проекта, выполните команду git init, добавьте README.md, выполните git add и git commit.

Для задания 2 скачайте датасет Iris из открытых источников, сохраните его в CSV, проверьте корректность формата.

Для задания 3 напишите скрипт, который загружает CSV, разделяет данные на признаки и метки, выполняет train_test_split, сохраняет массивы при помощи numpy.save.

Для задания 4 в отдельном скрипте импортируйте модель из sklearn, обучите её на сохранённых данных, сохраните модель через joblib.dump.

Для задания 5 используйте sklearn.metrics.confusion_matrix, постройте тепловую карту с помощью matplotlib, сохраните изображение командой plt.savefig('confusion_matrix.png').

Для задания 6 вычислите метрики через sklearn.metrics, сформируйте pandas.DataFrame и экспортируйте в CSV командой to_csv('metrics.csv').

Для задания 7 создайте тестовый файл, опишите функции проверки корректности расчёта метрик, запустите pytest и убедитесь, что все тесты проходят.

Для задания 8 подготовьте отчёт в LibreOffice: титульный лист, список заданий, скриншоты кода (делайте снимки экрана), вставьте график и таблицу, добавьте раздел «Выводы».

Для задания 9 упакуйте все файлы в ZIP-архив, проверьте наличие всех пунктов, отправьте архив по указанному email.

Отчёт в формате ODT или PDF. Структура: титульный лист; цель практической работы; последовательность выполненных заданий с номерами; скриншоты кода и терминала; изображение confusion_matrix.png; таблица metrics.csv; выводы и оценка результатов.

Сдача: архив ZIP (project.zip) отправить по email преподавателю до конца занятия.

Тема практической работы №6. Оценка качества нейронной сети с использованием ROC-кривой, объем часов 10

У4. Работать с системами контроля версий для управления проектами (Git, GitLab). Организовывать совместную работу над проектом через ветки разработки и слияние изменений. Разрешать конфликты при слиянии кода.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Научиться самостоятельно оценивать качество нейронной сети с помощью ROC-кривой и AUC, оформлять результаты в виде графиков и метрик, а также управлять проектом через Git.

Задание(я):

1. Подготовка проекта и данных.

2. Обучение простой нейронной сети на датасете Iris.

3. Оценка модели: построение ROC-кривой, вычисление AUC, сохранение графика.

4. Создание unit-тестов для функций предобработки и расчёта метрик.

5. Оформление репозитория: коммиты, ветка, merge, разрешение конфликтов; подготовка отчёта и архивирования.

Методические указания по ходу выполнения работы:

В задании 1 создайте папку проекта, инициализируйте Git-репозиторий, сделайте первый commit, загрузите данные Iris в CSV, сохраните скрипт загрузки как load_data.py.

В задании 2 реализуйте обучение сети в файле train_model.py, сохраняйте обученную модель в файл model.joblib, экспортируйте предсказания и истинные метки в CSV (predictions.csv).

В задании 3 напишите скрипт evaluate.py, постройте ROC-кривую, сохраните её как roc_curve.png, запишите значение AUC в файл metrics.txt.

В задании 4 создайте каталог tests, напишите тесты с использованием pytest в файлах test_preprocess.py и test_metrics.py, убедитесь, что все тесты проходят, зафиксируйте результаты в файле test_report.txt.

В задании 5 создайте отдельную ветку feature/report, добавьте в неё финальный отчёт (ODT или PDF), выполните merge в основную ветку, при конфликте используйте стандартные команды Git для их разрешения, сделайте финальный commit.

Формат сдачи: архив ZIP, содержащий весь каталог проекта, включая .py-файлы, CSV-данные, график roc_curve.png, файл metrics.txt, папку tests, файл test_report.txt и отчёт ODT/PDF.

Отчёт в ODT или PDF: титульный лист, цель работы, описание каждого задания, скриншоты кода (IDE), скриншот ROC-кривой, таблица метрик, выводы о качестве модели, список выполненных Git-коммитов с хешами.

Сдать архив ZIP с полным проектом по электронной почте преподавателю до конца занятия.

Тема практической работы №7. Разработка юнит-тестов для модели машинного обучения, объем часов 10

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У7. Определять критические сценарии работы системы, которые необходимо протестировать. Разрабатывать пошаговые тестовые сценарии на основе требований. Оценивать покрытие тестов и их соответствие техническому заданию.

Цель практической работы:

Освоить навыки самостоятельного написания и запуска юнит-тестов для модели машинного обучения, включая подготовку окружения, сохранение артефактов и анализ результатов.

Задание(я):

1. Подготовка рабочего окружения и установка необходимых библиотек (pip install scikit-learn pytest joblib matplotlib). Сохранить список установленных пакетов в файл requirements.txt.

2. Загрузка публичного датасета Iris, базовая предобработка и сохранение подготовленного набора в CSV (iris_prepared.csv).

3. Обучение простой модели (LogisticRegression) на подготовленных данных, сохранение обученной модели в файл model.joblib.

4. Разработка юнит-тестов для функций предобработки и метода predict модели, размещение тестов в файле test_model.py.

5. Запуск тестов с помощью pytest, сбор и сохранение отчёта о покрытии в файл coverage.txt, построение графика покрытия и сохранение его в coverage.png.

6. Оформление отчёта (ODT или PDF) с описанием выполненных шагов, скриншотами результатов тестов и графика, упаковка всех файлов (коды, данные, модели, отчёт) в ZIP-архив.

Методические указания по ходу выполнения работы:

В задании 1 создайте виртуальное окружение, активируйте его и выполните указанные pip install. Сохраните список пакетов командой pip freeze > requirements.txt.

В задании 2 загрузите датасет Iris из библиотеки scikit-learn, выполните стандартизацию признаков, разделите данные на обучающую и тестовую выборки, экспортируйте полученные наборы в CSV-файл iris_prepared.csv.

В задании 3 обучите модель LogisticRegression на подготовленных данных, проверьте её работоспособность, затем сохраните объект модели с помощью joblib.dump в файл model.joblib.

В задании 4 разработайте набор юнит-тестов в файле test_model.py: тесты должны проверять корректность предобработки (например, отсутствие NaN) и правильность работы метода predict (соответствие формата вывода, корректные типы).

В задании 5 запустите тесты командой pytest с опцией --cov, сохраните текстовый отчёт о покрытии в coverage.txt, постройте график покрытия (например, столбчатая диаграмма) и сохраните его как coverage.png.

В задании 6 подготовьте отчёт: титульный лист, цель работы, описания всех заданий, скриншоты вывода pytest, содержимое coverage.txt и изображение coverage.png. Сохраните отчёт в формате ODT или PDF. Сложите все файлы (коды, CSV, модель, requirements.txt, отчёт) в один ZIP-архив.

Отчёт — ODT или PDF. Структура: титульный лист; цель практической работы; последовательное описание выполненных заданий; скриншоты консольного вывода pytest; содержимое файла coverage.txt; график покрытия (coverage.png); выводы и комментарии студента.

Сдача: упаковать все материалы в ZIP-архив и отправить по электронной почте преподавателю до окончания занятия.

Тема практической работы №8. Интеграция модели ИИ в веб-приложение, объем часов 10

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У7. Определять критические сценарии работы системы, которые необходимо протестировать. Разрабатывать пошаговые тестовые сценарии на основе требований. Оценивать покрытие тестов и их соответствие техническому заданию.

Цель практической работы:

Научиться самостоятельно интегрировать обученную модель ИИ в веб-приложение, реализовать логирование, написать автоматические тесты и оформить результаты.

Задание(я):

1. Подготовка окружения и структуры проекта.

2. Обучение простой модели на датасете Iris, оценка качества и сохранение модели.

3. Реализация скрипта для загрузки модели и выполнения предсказания.

4. Создание Flask-приложения с эндпоинтом /predict, принимающим параметры и возвращающим результат.

5. Добавление логирования запросов и ошибок в Flask-приложение, сохранение логов в файл.

6. Написание автоматических тестов для функции предсказания с использованием pytest

7. Проверка работы веб-сервиса через curl/POSTMAN, сохранение скриншотов ответов.

8. Формирование архива с кодом, моделью, логами, тестами и отчётом.

Методические указания по ходу выполнения работы:

В задании 1 создайте папку проекта, установите необходимые библиотеки (flask, scikit-learn, pandas, joblib, pytest) командой pip install, сохраните список установленных пакетов в файл requirements.txt.

В задании 2 загрузите датасет Iris с помощью pandas, разделите его на обучающую и тестовую выборки, обучите классификатор (например, LogisticRegression), вычислите метрики точности и сохраните модель в файл model.joblib с помощью joblib.dump; метрики запишите в CSV файл metrics.csv.

В задании 3 создайте отдельный скрипт inference.py, в котором реализуется загрузка модели из model.joblib и функция predict, принимающая массив признаков и возвращающая предсказание; сохраните скрипт в корне проекта.

В задании 4 разработайте Flask-приложение app.py, реализующее маршрут /predict, который принимает JSON с признаками, вызывает функцию predict из inference.py и возвращает результат в формате JSON; сохраните приложение в корне проекта.

В задании 5 добавьте в app.py настройку модуля logging: вывод запросов и ошибок в файл app.log, уровень INFO; убедитесь, что каждый запрос записывается в лог.

В задании 6 создайте каталог tests и файл test_inference.py, в котором с помощью pytest напишите минимум три тест-кейса для функции predict (корректные данные, некорректные данные, граничные значения); результаты тестов сохраняйте в файл test_report.txt.

В задании 7 запустите Flask-приложение, выполните несколько запросов к /predict с помощью curl (или аналогичного инструмента), зафиксируйте ответы и сделайте скриншоты терминала/команд; сохраните скриншоты в папку screenshots.

В задании 8 упакуйте все файлы проекта (code/*.py, model.joblib, metrics.csv, app.log, requirements.txt, tests/, screenshots/, отчёт) в архив project.zip.

Отчёт в формате ODT или PDF. Структура: титульный лист, цель работы, список выполненных заданий с указанием времени, описание проделанных действий, ссылки на файлы (model.joblib, metrics.csv, app.log, test_report.txt), скриншоты запросов, выводы о работе модели и веб-сервиса.

Сдача: отправить архив ZIP (project.zip) по e-mail преподавателя не позднее конца 10-часового занятия.

Тема практической работы №9. Юзабилити-тестирование приложения после интеграции, объем часов 8

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Освоить навыки самостоятельного проведения юзабилити-тестирования интегрированного ИИ-приложения, включая планирование тестов, сбор и анализ пользовательских данных, логирование и оформление отчёта.

Задание(я):

1. Подготовить окружение: установить Python, pip, необходимые пакеты (flask, scikit-learn, pandas, matplotlib, pytest). Сохранить список установленных пакетов в файл requirements.txt.

2. Запустить готовый пример приложения (Flask-сервер с простым предиктивным модулем). Сохранить инструкцию запуска в файл run.txt.

3. Сформировать план юзабилити-тестирования: определить цели теста, сценарии взаимодействия пользователя, критерии оценки (время отклика, понятность интерфейса, количество ошибок ввода). Сохранить план в файл usability_plan.md.

4. Провести ручные тесты с 3-5 добровольцами, используя подготовленные сценарии. Зафиксировать замечания, время выполнения действий и комментарии в таблице CSV (usability_results.csv).

5. Добавить в приложение базовое логирование (уровень INFO) для записи действий пользователя и времени отклика. Сохранить лог-файл (app.log).

6. На основе собранных данных построить графики зависимости времени отклика от типа запроса и количество ошибок по сценариям. Сохранить графики как PNG-файлы (response_time.png, error_counts.png).

7. Проанализировать полученные результаты: выделить проблемные зоны, предложить улучшения интерфейса и логики обработки запросов. Оформить выводы в файл analysis.md.

8. Подготовить итоговый отчёт (ODT или PDF) со следующей структурой: титульный лист, цель работы, описание среды, план тестирования, результаты (таблицы, скриншоты, графики), анализ и выводы. Включить скриншоты интерфейса приложения и примеры логов.

9. Упаковать весь материал (исходный код .py, requirements.txt, run.txt, usability_plan.md, usability_results.csv, app.log, графики PNG, analysis.md, итоговый отчёт) в архив ZIP для сдачи.

Методические указания по ходу выполнения работы:

Для задания 1 используйте команду pip install -r requirements.txt; сохраните список пакетов в указанный файл.

В задании 2 опишите в run.txt способ запуска сервера (например, python app.py) и порт, на котором он работает.

В задании 3 создайте документ usability_plan.md, где перечислите цели, сценарии (например, ввод текста, получение предсказания), критерии и методику сбора данных.

В задании 4 проведите тесты, фиксируя данные в таблице CSV с колонками: пользователь, сценарий, время_начала, время_окончания, замечания, количество_ошибок.

В задании 5 настройте логирование в приложении так, чтобы в app.log записывались timestamp, действие пользователя и длительность обработки.

В задании 6 постройте графики с помощью matplotlib, сохраните их функцией plt.savefig('имя.png') в корневой папке проекта.

В задании 7 проанализируйте CSV-данные и логи, сформулируйте рекомендации и запишите их в analysis.md.

В задании 8 оформите отчёт согласно указанной структуре, включив скриншоты интерфейса (сделайте снимки экрана) и графики PNG.

В задании 9 убедитесь, что в архиве присутствуют все перечисленные файлы и подпапки; назовите архив согласно шаблону <фамилия_имя>_PR9.zip.

Отчёт в формате ODT или PDF. Структура: 1) Титульный лист (название работы, ФИО, группа, дата); 2) Цель и задачи; 3) Описание среды и способа запуска; 4) План юзабилити-тестирования; 5) Результаты (таблицы CSV, скриншоты интерфейса, графики PNG); 6) Анализ и выводы; 7) Приложения (лог-файл, исходный код). Все графики и скриншоты должны быть вставлены в отчёт и сопоставлены с описанием.

Сдача: упаковать все материалы в ZIP-архив и отправить по электронной почте преподавателю не позже конца 8-часового занятия.

Тема практической работы №10. Тестирование безопасности ИИ-приложений. Тестирование совместимости с браузерами, объем часов 8

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Научиться самостоятельно выполнять тестирование безопасности и совместимости с браузерами AI-веб-приложений, используя инструменты отладки, логирования и автоматизации тестов.

Задание(я):

1. Подготовка окружения. Установить Python, pip, PyCharm (или VS Code), Selenium, pytest, requests, и драйверы браузеров (Chrome, Firefox). Сохранить список установленных пакетов в файл requirements.txt.

2. Развёртывание простого AI-веб-приложения. Склонировать (или создать) минимальный Flask-сервис, который принимает текстовый запрос и возвращает предсказание (модель-заглушка). Сохранить исходный код в файле app.py и Docker-файл (опционально) в отдельной папке.

3. Написание и запуск юнит-тестов. Создать набор pytest-тестов, проверяющих корректность API-эндпоинта (правильный ответ, статус-код,

обработка пустого запроса). Сохранить тесты в файле test_api.py, результаты выполнения (stdout) экспортировать в файл test_report.txt.

4. Тестирование безопасности. С помощью Postman (или requests) выполнить проверки на уязвимости: ввод вредоносных строк (XSS, SQL-инъекция), проверка ограничения длины, проверка обработки неверных типов данных. Все запросы и ответы записать в CSV-файл security_log.csv, а обнаруженные уязвимости оформить в отдельный файл security_report.txt. При необходимости добавить скриншоты ответов (png).

5. Тестирование совместимости с браузерами. С помощью Selenium написать скрипт, открывающий веб-интерфейс приложения в Chrome и Firefox, проверяющий наличие основных элементов (поле ввода, кнопка отправки) и корректность отображения ответа. Сохранить скрипт в файле test_browser.py, результаты (успешные/неуспешные проверки) в файл browser_report.txt, а скриншоты страниц в папку screenshots/ (png).

6. Сбор логов и подготовка отчёта. Сконфигурировать логирование в приложении (logging-модуль) для записи всех запросов в файл app_log.csv. Сформировать итоговый отчёт, включающий цель работы, описание каждого задания, скриншоты, таблицы CSV, выводы о найденных уязвимостях и совместимости. Сохранить отчёт в формате ODT (или PDF) под именем report.odt.

Методические указания по ходу выполнения работы:

Для задания 1 используйте команду pip install -r requirements.txt после указания всех необходимых пакетов (selenium, pytest, requests, flask). Запишите список пакетов в файл requirements.txt.

Для задания 2 разместите все файлы приложения в отдельной папке app/. Запустите приложение командой python app.py и проверьте доступность по адресу http://127.0.0.1:5000/.

Для задания 3 создайте папку tests/ и поместите туда test_api.py. Запустите pytest и перенаправьте вывод в файл test_report.txt (pytest > test_report.txt).

Для задания 4 подготовьте набор «плохих» запросов в виде CSV-файла (payloads.csv). С помощью скрипта (requests) выполните запросы к API, сохраняйте ответы в security_log.csv, а результаты анализа в security_report.txt. При обнаружении уязвимости сделайте скриншот ответа и сохраните в папку screenshots/.

Для задания 5 установите драйверы ChromeDriver и GeckoDriver, укажите их пути в переменных окружения. В скрипте test_browser.py реализуйте последовательность открытий страниц, проверок элементов и сохранения скриншотов (screenshot.save('screenshots/chrome.png') и т.п.). Запишите результаты в browser_report.txt.

Для задания 6 настройте логирование в app.py: logging.basicConfig(filename='app_log.csv', level=logging.INFO, format='%(asctime)s,%(message)s'). После выполнения всех тестов соберите все файлы (requirements.txt, app/, tests/, security_*.csv, browser_report.txt, screenshots/, report.odt) в один архив.

Формат сдачи: архив ZIP, содержащий весь исходный код, файлы данных, скриншоты и итоговый отчёт.

Формат отчёта: ODT или PDF. Титульный лист (название работы, ФИО, группа, дата). Содержание. Описание целей и задач. По каждому заданию – краткое описание, использованные инструменты, результаты (таблицы CSV, скриншоты, выводы). Заключение с оценкой безопасности и совместимости. Приложения – полные логи и скрипты.

Сдача: архив ZIP отправить по email преподавателю до конца занятия (указать срок).

Тема практической работы №11. Тестирование API, объем часов 8

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Научиться самостоятельно выполнять тестирование API, включая написание юнит- и интеграционных тестов, автоматизацию их запуска и оформление отчётов.

Задание(я):

1. Подготовка рабочего окружения (установить Python, pip, создать виртуальное окружение, установить pytest, requests, Flask, Postman).

2. Реализация простого API на Flask с двумя эндпоинтами (GET /ping, POST /sum). Сохранить код в файл app.py.

3. Написание юнит-тестов для функций API с использованием pytest. Сохранить тесты в файл test_unit.py.

4. Написание интеграционных тестов, которые отправляют HTTP-запросы к запущенному серверу (использовать библиотеку requests). Сохранить в файл test_integration.py.

5. Автоматизация запуска всех тестов через один скрипт (run_tests.py) и сохранение результатов в файл test_report.txt.

6. Создание коллекции запросов в Postman, выполнение тестов, экспорт коллекции и результатов в файлы postman_collection.json и postman_report.json.

7. Сбор всех артефактов (исходный код, тесты, скрипты, отчёты, скриншоты выполнения) в отдельную папку.

8. Оформление отчёта о выполненной работе (структура, скриншоты, выводы) и упаковка в ZIP-архив.

Методические указания по ходу выполнения работы:

В задании 1 создайте виртуальное окружение (python -m venv venv), активируйте его и выполните pip install pytest requests Flask. Убедитесь, что все пакеты установлены, проверив их версии в консоли.

В задании 2 реализуйте API, разместив код в файле app.py. Запустите приложение (python app.py) и проверьте работу эндпоинтов через браузер или curl. Не забудьте добавить простое логирование запросов (print или logging).

В задании 3 напишите юнит-тесты, проверяющие функции, которые обслуживают эндпоинты (например, проверка возвращаемого значения функции-обработчика). Сохраните тесты в файле test_unit.py. Запустите их командой pytest test_unit.py и убедитесь, что все тесты проходят.

В задании 4 напишите интеграционные тесты, которые отправляют реальные HTTP-запросы к запущенному серверу. Тесты должны проверять

статус-код и тело ответа. Сохраните их в файле test_integration.py. Перед запуском убедитесь, что сервер работает в фоновом режиме.

В задании 5 создайте скрипт run_tests.py, который последовательно запускает pytest для юнит- и интеграционных тестов и записывает вывод в файл test_report.txt (используйте subprocess). Проверьте, что файл отчёта создан и содержит результаты всех тестов.

В задании 6 в Postman создайте коллекцию запросов к эндпоинтам /ping и /sum, добавьте проверки (tests) в каждый запрос. Выполните коллекцию, экспортируйте её в файл postman_collection.json и экспортируйте результаты выполнения в файл postman_report.json. Сохраните оба файла в рабочую папку.

В задании 7 соберите все файлы: app.py, test_unit.py, test_integration.py, run_tests.py, test_report.txt, postman_collection.json, postman_report.json, а также скриншоты работы сервера и результатов тестов. Поместите их в одну папку "project_artifacts".

В задании 8 подготовьте отчёт в формате ODT или PDF со следующей структурой: титульный лист, цель работы, перечень выполненных заданий, описание среды разработки, скриншоты выполнения (сервер, результаты pytest, результаты Postman), выводы. Сохраните отчёт как report.odt (или report.pdf). Упакуйте папку "project_artifacts" вместе с отчётом в ZIP-архив.

Отчёт должен быть оформлен в ODT или PDF, включать титульный лист, цель работы, пошаговое описание выполненных заданий, скриншоты (сервер запущен, вывод pytest, результаты Postman), таблицу с результатами тестов и раздел с выводами и рекомендациями.

Сдача: упаковать папку с исходным кодом, тестами, артефактами и отчётом в ZIP-архив и отправить по e-mail преподавателю до конца занятия.

Тема практической работы №12. Мониторинг производительности ИИ-модели с использованием систем мониторинга и оповещения и мониторинга и визуализации данных, объем часов 8

У5. Использовать инструменты для отладки программного кода. Идентифицировать и исправлять ошибки в программе. Применять методы логирования для анализа выполнения программ.

У6. Проводить различные виды тестирования (юнит-тестирование, интеграционное тестирование). Разрабатывать тестовые сценарии для проверки корректности работы программных модулей. Автоматизировать тестирование программного обеспечения.

Цель практической работы:

Научиться самостоятельно выполнять мониторинг и визуализацию производительности обученной ИИ-модели, используя средства логирования, профилирования и тестирования.

Задание(я):

1. Подготовка рабочего окружения и установка необходимых пакетов (pip install scikit-learn pandas matplotlib joblib psutil pytest).

2. Обучение простой модели (например, классификатор на датасете Iris) и сохранение её в файл (joblib.dump).

3. Добавление в код логирования и измерения ключевых метрик (время инференса, использование CPU и памяти) с сохранением результатов в CSV-файл.

4. Построение графиков зависимости времени инференса и использования ресурсов, сохранение графиков в PNG-файлы (plt.savefig).

5. Разработка наборов unit-тестов (pytest) для проверки корректности работы модели и корректности измерений, а также подготовка отчёта с описанием выполненных шагов, скриншотами и выводами.

Методические указания по ходу выполнения работы:

Для задания 1 создайте отдельную папку проекта, откройте терминал и выполните перечисленные команды установки. Проверьте, что все пакеты успешно установлены, запустив их импорт в интерактивной консоли.

В задании 2 напишите скрипт train_model.py, который загружает датасет Iris, делит его на обучающую и тестовую части, обучает выбранный классификатор и сохраняет обученную модель в файл model.pkl. Сохраните скрипт в корне проекта.

В задании 3 добавьте в скрипт inference.py импорт модуля logging, настройте запись логов в файл logs.csv и используйте модуль psutil для

измерения текущего потребления CPU и памяти перед и после вызова predict. Запишите измеренные значения в CSV-файл metrics.csv.

Для задания 4 создайте скрипт plot_metrics.py, который читает metrics.csv, строит графики зависимости времени предсказания и использования ресурсов от номера прогона, сохраняет графики как inference_time.png и resource_usage.png, а также вставляет их в отчёт.

В задании 5 разработайте набор тестов в файле test_model.py, проверяющие: а) корректность загрузки модели; б) соответствие формы входных данных; в) отсутствие исключений при измерении метрик. Запустите pytest и убедитесь, что все тесты проходят. Затем сформируйте отчёт, включив титульный лист, описание каждого задания, скриншоты терминала, содержимое logs.csv и screenshots графиков, а также выводы о производительности модели.

Все исходные файлы (.py), CSV-файлы, PNG-графики и отчёт упакуйте в один архив ZIP для сдачи.

Формат отчёта: ODT или PDF. Структура отчёта – титульный лист (название работы, ФИО, группа, дата), краткое описание целей и задач, пошаговое описание выполнения каждого задания, скриншоты терминала и IDE, содержимое logs.csv (в виде таблицы), графики (вставленные PNG-изображения) с подписью, раздел с выводами о производительности модели и перечень использованных инструментов.

Сдача: упаковать все файлы проекта и отчёт в архив ZIP, назвать архив согласно шаблону <ФИО>_P12.zip и отправить по email преподавателю до конца занятия.

II. Общие рекомендации

По всем вопросам, связанным с изучением дисциплины (включая самостоятельную работу), консультироваться с преподавателем.

III. Контроль и оценка результатов

Оценка за выполнение практической работы выставляется по пятибалльной системе и учитывается как показатель текущей успеваемости студента.

Качественная оценка индивидуальных образовательных достижений		**Критерии оценки результата**
балл (оценка)	**вербальный аналог**	
5	отлично	Представленная работа выполнена в полном объёме и высокого качества. Все задания практической работы выполнены без ошибок, соблюдены методические указания. Теоретическое содержание профессионального модуля освоено полностью, без пробелов. Студент демонстрирует уверенное владение инструментами разработки ИИ-решений, корректно использует библиотеки и алгоритмы, грамотно оформляет код и отчёт, способен аргументированно объяснить полученные результаты.
4	хорошо	Уровень выполнения работы в целом соответствует требованиям. Все задания выполнены, однако в работе могут присутствовать незначительные ошибки или неточности в коде, расчётах или оформлении отчёта. Теоретическое содержание модуля освоено полностью, но отдельные практические навыки сформированы недостаточно устойчиво.
3	удовлетворительно	Работа выполнена частично и соответствует большинству основных требований. Теоретическое содержание модуля освоено не полностью, имеются пробелы, не носящие критического характера. Большинство заданий выполнено, однако допущены ошибки в реализации алгоритмов, интерпретации результатов или оформлении отчёта. Практические навыки сформированы на базовом уровне.
2	не удовлетворительно	Теоретическое содержание модуля освоено фрагментарно. Необходимые практические навыки разработки и анализа ИИ-решений не сформированы. Большинство заданий не выполнено или выполнено с существенными ошибками, отсутствует корректный отчёт и обоснование полученных результатов.