Оглавление

[Тема 1: Введение 5](#_Toc186255726)

[Цель данного проекта 5](#_Toc186255727)

[Тема 2: Обоснование выбора темы проекта 7](#_Toc186255728)

[Актуальность темы 7](#_Toc186255729)

[Быстрый рост области машинного обучения 7](#_Toc186255730)

[Популярность используемых библиотек 7](#_Toc186255731)

[Соответствие современным тенденциям 8](#_Toc186255732)

[Переход от классических методов к глубокому обучению 8](#_Toc186255733)

[Вопросы производительности и масштабируемости 8](#_Toc186255734)

[Тема 3: Определение цели и задач 8](#_Toc186255735)

[Цель исследования 8](#_Toc186255736)

[Тема 4: Основные понятия и определения 10](#_Toc186255737)

[Машинное обучение 10](#_Toc186255738)

[Надзорное обучение (Supervised Learning) 10](#_Toc186255739)

[Регрессия 10](#_Toc186255740)

[Классификация 10](#_Toc186255741)

[Модель машинного обучения 10](#_Toc186255742)

[Библиотеки машинного обучения 11](#_Toc186255743)

[Датасеты (Наборы данных) 11](#_Toc186255744)

[Предобработка данных 11](#_Toc186255745)

[Разделение данных 12](#_Toc186255746)

[Гиперпараметры 12](#_Toc186255747)

[Функции активации 12](#_Toc186255748)

[Функции потерь (Loss Functions) 13](#_Toc186255749)

[Оптимизаторы 13](#_Toc186255750)

[Тема 5: Метрики оценки качества моделей 13](#_Toc186255751)

[Для регрессии 13](#_Toc186255752)

[Для классификации 14](#_Toc186255753)

[Переобучение и недообучение 15](#_Toc186255754)

[Регуляризация 15](#_Toc186255755)

[Разложение данных на батчи (Batching) 15](#_Toc186255756)

[Кросс-валидация 16](#_Toc186255757)

[Эпоха (Epoch) 16](#_Toc186255758)

[Ранняя остановка (Early Stopping) 16](#_Toc186255759)

[Архитектура нейронной сети 16](#_Toc186255760)

[Оценка модели (Model Evaluation) 16](#_Toc186255761)

[Обучение модели (Model Training) 17](#_Toc186255762)

[Дата-сайенс (Data Science) 17](#_Toc186255763)

[Экспериментальные протоколы 17](#_Toc186255764)

[Фреймворки глубокого обучения 17](#_Toc186255765)

[Аппаратное ускорение 17](#_Toc186255766)

[Предсказательная способность модели 17](#_Toc186255767)

[Случайность и воспроизводимость 18](#_Toc186255768)

[Тема 6: Архитектура и методология проекта 18](#_Toc186255769)

[Общие сведения 18](#_Toc186255770)

[scikit-learn 18](#_Toc186255771)

[Подход к реализации моделей 18](#_Toc186255772)

[Архитектура моделей 19](#_Toc186255773)

[Классификация 19](#_Toc186255774)

[Регрессия 19](#_Toc186255775)

[Методы обучения 19](#_Toc186255776)

[TensorFlow 20](#_Toc186255777)

[Подход к реализации моделей 20](#_Toc186255778)

[Архитектура моделей 20](#_Toc186255779)

[Классификация 20](#_Toc186255780)

[Регрессия 20](#_Toc186255781)

[Методы обучения 21](#_Toc186255782)

[PyTorch 21](#_Toc186255783)

[Подход к реализации моделей 21](#_Toc186255784)

[Архитектура моделей 22](#_Toc186255785)

[Классификация 22](#_Toc186255786)

[Регрессия 22](#_Toc186255787)

[Методы обучения 22](#_Toc186255788)

[Сравнение подходов 23](#_Toc186255789)

[Простота реализации 23](#_Toc186255790)

[Гибкость и контроль 24](#_Toc186255791)

[Обучение и производительность 24](#_Toc186255792)

[Тема 7: Обеспечение безопасности 24](#_Toc186255793)

[Опасности использования сомнительных библиотек 24](#_Toc186255794)

[Вредоносный код 24](#_Toc186255795)

[Уязвимости и эксплойты 25](#_Toc186255796)

[Несовместимость лицензий 25](#_Toc186255797)

[Рекомендации по обеспечению безопасности 25](#_Toc186255798)

[Использование надёжных источников 25](#_Toc186255799)

[Проверка библиотек на наличие уязвимостей 25](#_Toc186255800)

[Использование виртуальных окружений 25](#_Toc186255801)

[Проверка целостности пакетов 26](#_Toc186255802)

[Тема 8: Обзор популярных инструментов для разработки на Python 26](#_Toc186255803)

[Интегрированные среды разработки (IDE) и текстовые редакторы 26](#_Toc186255804)

[PyCharm 26](#_Toc186255805)

[Visual Studio Code (VS Code) 26](#_Toc186255806)

[Atom 27](#_Toc186255807)

[Jupyter Notebook 27](#_Toc186255808)

[Инструменты управления пакетами и виртуальными окружениями 27](#_Toc186255809)

[pip 27](#_Toc186255810)

[virtualenv 27](#_Toc186255811)

[venv (встроенный модуль) 28](#_Toc186255812)

[Инструменты для тестирования 28](#_Toc186255813)

[unittest 28](#_Toc186255814)

[pytest 28](#_Toc186255815)

[Инструменты для отладки 28](#_Toc186255816)

[pdb 28](#_Toc186255817)

[ipdb 28](#_Toc186255818)

[PyCharm Debugger 29](#_Toc186255819)

[Visual Studio Code Debugger 29](#_Toc186255820)

[Инструменты для статического анализа кода 29](#_Toc186255821)

[pylint 29](#_Toc186255822)

[black 29](#_Toc186255823)

[Средства управления версиями и совместной работы 30](#_Toc186255824)

[Git 30](#_Toc186255825)

[GitHub, GitLab, Bitbucket 30](#_Toc186255826)

[Инструменты автоматизации и CI/CD 30](#_Toc186255827)

[Jenkins 30](#_Toc186255828)

[Travis CI 30](#_Toc186255829)

[GitHub Actions 30](#_Toc186255830)

[Фреймворки для веб-разработки 31](#_Toc186255831)

[Django 31](#_Toc186255832)

[Flask 31](#_Toc186255833)

[Библиотеки для научных вычислений и машинного обучения 31](#_Toc186255834)

[NumPy 31](#_Toc186255835)

[Pandas 31](#_Toc186255836)

[Matplotlib и Seaborn 32](#_Toc186255837)

[scikit-learn 32](#_Toc186255838)

[TensorFlow и Keras 32](#_Toc186255839)

[PyTorch 32](#_Toc186255840)

[Другие полезные инструменты 33](#_Toc186255841)

[Cookiecutter 33](#_Toc186255842)

[Sphinx 33](#_Toc186255843)

[Ansible 33](#_Toc186255844)

[Тема 9: Проектирование приложения 34](#_Toc186255845)

[Планирование разработки в Google Colab 34](#_Toc186255846)

[Особенности использования Colab 34](#_Toc186255847)

[Тема 10: Основные требования для реализации проекта 34](#_Toc186255848)

[Технические требования 34](#_Toc186255849)

[Программные требования 35](#_Toc186255850)

[Установка и настройка 36](#_Toc186255851)

[Данные и наборы данных 36](#_Toc186255852)

[Обработка и подготовка данных 36](#_Toc186255853)

[Тема 11: Анализ результатов 39](#_Toc186255854)

[Регрессия: 39](#_Toc186255855)

[Классификация: 39](#_Toc186255856)

[Тема 12: Обзор выполненной работы 41](#_Toc186255857)

[Постановка целей и задач 41](#_Toc186255858)

[Подготовка данных 41](#_Toc186255859)

[Реализация моделей 42](#_Toc186255860)

[Обучение моделей 43](#_Toc186255861)

[Сбор и анализ результатов 43](#_Toc186255862)

[Итоги сравнения 44](#_Toc186255863)

[Регрессия: 44](#_Toc186255864)

[Классификация: 44](#_Toc186255865)

[Выводы и рекомендации 45](#_Toc186255866)

[Особенности библиотек: 45](#_Toc186255867)

[Рекомендации по выбору моделей: 46](#_Toc186255868)

[Тема 13: Заключение 47](#_Toc186255869)

[Выбор и подготовка данных 47](#_Toc186255870)

[Реализация моделей 48](#_Toc186255871)

[Обучение моделей 48](#_Toc186255872)

[Оценка моделей 48](#_Toc186255873)

[Результаты 48](#_Toc186255874)

[Результаты регрессии 48](#_Toc186255875)

[Результаты классификации 49](#_Toc186255876)

[Тема 14: Дальнейшие планы 49](#_Toc186255877)

[Расширение набора моделей и алгоритмов 49](#_Toc186255878)

[Использование более сложных архитектур 49](#_Toc186255879)

[Исследование ансамблевых методов 50](#_Toc186255880)

[Включение дополнительных алгоритмов машинного обучения 50](#_Toc186255881)

[Оптимизация моделей и гиперпараметров 51](#_Toc186255882)

[Автоматизированный подбор гиперпараметров 51](#_Toc186255883)

[Регуляризация и борьба с переобучением 51](#_Toc186255884)

[Добавление методов регуляризации: 51](#_Toc186255885)

[Улучшение предобработки и расширение данных 52](#_Toc186255886)

[Аугментация данных 52](#_Toc186255887)

[Обработка дисбаланса классов 52](#_Toc186255888)

[Расширение набора данных 52](#_Toc186255889)

[Использование новых датасетов 52](#_Toc186255890)

[Обработка больших данных 53](#_Toc186255891)

[Интерпретируемость и объяснимость моделей 53](#_Toc186255892)

[Применение XAI (Explainable AI) методов 53](#_Toc186255893)

[Визуализация внутренних представлений 53](#_Toc186255894)

[Интеграция и деплой моделей 53](#_Toc186255895)

[Создание приложений или сервисов 53](#_Toc186255896)

[Оптимизация и сжатие моделей 54](#_Toc186255897)

[Сравнение и оценка производительности 54](#_Toc186255898)

[Расширение метрик оценки 54](#_Toc186255899)

[Тестирование устойчивости моделей 54](#_Toc186255900)

[Тема 15: Потенциальные результаты и выгоды от развития проекта 55](#_Toc186255901)

[Углубление знаний и навыков: 55](#_Toc186255902)

[Улучшение практических результатов: 55](#_Toc186255903)

[Подготовка к промышленным проектам: 55](#_Toc186255904)

[Научный вклад и публикации: 55](#_Toc186255905)

[Карьерный рост: 55](#_Toc186255906)

# Тема 1: Введение

В современном мире объём данных растёт экспоненциально, и вместе с ним увеличивается потребность в эффективных методах их анализа и обработки. **Машинное обучение** и **глубокое обучение** стали ключевыми инструментами для извлечения знаний из больших массивов данных, находя применение в таких областях, как компьютерное зрение, обработка естественного языка, прогнозирование, медицина, финансы и многое другое.

Язык программирования **Python** завоевал широкую популярность среди специалистов по анализу данных и разработчиков благодаря своей простоте и богатой экосистеме библиотек. Среди множества инструментов для машинного обучения особое место занимают библиотеки **scikit-learn**, **TensorFlow** и **PyTorch**, которые предоставляют мощные средства для разработки и внедрения моделей.

**Scikit-learn** является одним из наиболее популярных инструментов для реализации классических алгоритмов машинного обучения. Он предлагает простой и единообразный интерфейс для широкого спектра методов регрессии, классификации и кластеризации.

**TensorFlow**, разработанный компанией Google, представляет собой открытый фреймворк для численных вычислений и глубинного обучения. Он ориентирован на создание и обучение сложных нейронных сетей и позволяет масштабировать модели от персонального компьютера до кластеров серверов.

**PyTorch**, созданный компанией Facebook, активно используется в исследовательской сфере благодаря динамическим вычислительным графам и гибкости при разработке моделей глубокого обучения.

Несмотря на распространённость и функциональность этих библиотек, разработчики и исследователи часто сталкиваются с вопросом выбора наиболее подходящего инструмента для решения конкретной задачи. Каждая библиотека имеет свои особенности, и понимание их преимуществ и ограничений становится критически важным для успешной реализации проектов.

Цель данного проекта — провести **сравнительный анализ** библиотек **scikit-learn**, **TensorFlow** и **PyTorch** при решении задач **регрессии** и **классификации**. Это позволит выявить особенности каждой библиотеки, оценить их производительность, удобство использования и применимость к различным сценариям.

**Актуальность исследования** обусловлена потребностью в практических рекомендациях по выбору инструментов машинного обучения. С ростом значимости данных и сложности моделей правильный выбор технологии может существенно повлиять на эффективность разработки и качество конечного продукта.

**В рамках проекта** будут выполнены следующие этапы:

**Анализ функциональных возможностей библиотек**:

Изучение архитектуры и основных компонентов scikit-learn, TensorFlow и PyTorch.

Обзор поддерживаемых алгоритмов регрессии и классификации.

**Выбор и подготовка наборов данных**:

Для задачи регрессии будет использован, например, **набор данных о ценах на жилье в Калифорнии**.

Для задачи классификации — **набор данных MNIST** с изображениями рукописных цифр.

Проведена предобработка данных: очистка, нормализация, разделение на обучающую и тестовую выборки.

**Реализация моделей**:

Построение моделей регрессии и классификации с использованием каждой библиотеки.

Обеспечение сопоставимости моделей по архитектуре и гиперпараметрам.

**Обучение и оптимизация моделей**:

Обучение моделей на соответствующих наборах данных.

Настройка гиперпараметров с использованием подходящих методов оптимизации.

**Оценка и сравнение результатов**:

Оценка производительности моделей по метрикам качества:

* + Для регрессии: **среднеквадратичная ошибка (MSE)**, **средняя абсолютная ошибка (MAE)**, **коэффициент детерминации (R²)**.
  + Для классификации: **точность (Accuracy)**, **полнота (Recall)**, **прецизионность (Precision)**, **F1-score**.

Анализ времени обучения и предсказания, потребления ресурсов.

**Анализ удобства использования и функциональности**:

Оценка сложности и удобства реализации моделей в каждой библиотеке.

Изучение доступности документации и поддержки сообществом.

**Выводы и рекомендации**:

Формулирование выводов о преимуществах и недостатках каждой библиотеки.

Предоставление рекомендаций по выбору инструмента в зависимости от специфики задачи.

**Ожидаемые результаты** проекта включают в себя подробный сравнительный анализ рассматриваемых библиотек, практические выводы о их применимости к конкретным задачам и разработку рекомендаций для разработчиков и исследователей в области машинного обучения.

**Предлагаемое исследование** внесёт вклад в понимание того, как выбор инструментов может влиять на процесс разработки и результаты в области машинного обучения. Это особенно важно в условиях быстро меняющегося технологического ландшафта и увеличения разнообразия доступных библиотек и фреймворков.

# Тема 2: Обоснование выбора темы проекта

## Актуальность темы

### Быстрый рост области машинного обучения

* **Машинное обучение и глубокое обучение** становятся неотъемлемой частью современных технологий, проникая во множество сфер — от медицины и финансов до транспорта и развлечений.
* С увеличением объёма данных и вычислительных возможностей растёт потребность в эффективных инструментах для их обработки и анализа.

### Популярность используемых библиотек

* **scikit-learn**, **TensorFlow** и **PyTorch** являются одними из наиболее широко используемых библиотек в сообществе разработчиков и исследователей.
* Они предлагают различные возможности и подходят для разных типов задач и уровней сложности, что делает их сравнение особенно актуальным.

## Соответствие современным тенденциям

### Переход от классических методов к глубокому обучению

* В последние годы наблюдается сдвиг от использования классических алгоритмов машинного обучения к методам глубокого обучения для решения широкого спектра задач.
* Сравнение **scikit-learn** (представляющего классические методы) с **TensorFlow** и **PyTorch** (фокусирующимися на глубоком обучении) позволит понять, когда и почему стоит предпочесть тот или иной подход.

### Вопросы производительности и масштабируемости

* С ростом объёмов данных производительность и возможность масштабирования становятся критически важными.
* Анализ того, как разные библиотеки справляются с этими вызовами, поможет в принятии решений при разработке больших и сложных систем.

# Тема 3: Определение цели и задач

## Цель исследования

**Цель исследования** заключается в проведении **сравнительного анализа** библиотек **scikit-learn**, **TensorFlow** и **PyTorch** при решении типовых задач **регрессии** и **классификации**, с целью выявления их особенностей, преимуществ и недостатков, а также предоставления рекомендаций по выбору наиболее подходящей библиотеки для конкретных случаев применения.

Для достижения поставленной цели необходимо решить следующие **задачи**:

**Анализ существующих библиотек и инструментов**:

* Изучить функциональные возможности библиотек scikit-learn, TensorFlow и PyTorch.
* Проанализировать архитектурные особенности и принцип работы каждой библиотеки.

**Выбор подходящих наборов данных**:

* Подобрать репрезентативные датасеты для задач регрессии и классификации (например, California Housing для регрессии и MNIST для классификации).
* Обеспечить корректную предобработку и разделение данных для честного сравнения моделей.

**Реализация моделей на каждой из библиотек**:

* Разработать модели на основе нейронных сетей с сопоставимой архитектурой для каждой библиотеки.
* Гарантировать, что настройки гиперпараметров и условия обучения максимально сходны между моделями.

**Обучение моделей и проведение экспериментов**:

* Обучить модели на соответствующих наборах данных.
* Замерить время обучения, использовать одинаковые критерии остановки и методы оптимизации.

**Сбор и анализ результатов**:

* Оценить производительность моделей по выбранным метрикам качества (MSE, MAE, Accuracy и т.д.).
* Сравнить результаты между библиотеками, проанализировать различия и причины их возникновения.

**Оценка удобства использования и функциональности библиотек**:

* Проанализировать сложность реализации моделей в каждой библиотеке.
* Оценить документацию, поддержку сообществом и доступность ресурсов для обучения.

**Выводы и рекомендации**:

* Сформулировать выводы о преимуществах и недостатках каждой библиотеки в контексте решаемых задач.
* Предоставить рекомендации по выбору библиотеки в зависимости от специфики задачи и требований к модели.

**Определение направлений для дальнейшего исследования**:

* Предложить возможные улучшения и расширения исследования.
* Обозначить перспективные направления использования библиотек в других задачах машинного обучения.

# Тема 4: Основные понятия и определения

## Машинное обучение

* **Определение**: Область искусственного интеллекта, которая фокусируется на разработке алгоритмов и моделей, позволяющих компьютерам обучаться на данных и делать предсказания или решения без явного программирования для каждой задачи.
* **Роль в проекте**: Основная область, в рамках которой проводится сравнение моделей и библиотек для регрессии и классификации.

## Надзорное обучение (Supervised Learning)

* **Определение**: Подмножество машинного обучения, где алгоритмы обучаются на размеченных данных, то есть каждый пример в обучающем наборе данных имеет соответствующий правильный выход.
* **Роль в проекте**: Задачи регрессии и классификации являются типичными примерами надзорного обучения.

## Регрессия

* **Определение**: Задача предсказания непрерывной числовой величины на основе входных данных.
* **Пример**: Предсказание цены дома на основе его характеристик.
* **Роль в проекте**: Одна из двух основных задач, для которой сравниваются различные библиотеки и модели.

## Классификация

* **Определение**: Задача присвоения входным данным одного из нескольких дискретных классов.
* **Пример**: Определение цифры на изображении рукописного ввода.
* **Роль в проекте**: Вторая основная задача, используемая для сравнения библиотек и моделей.

## Модель машинного обучения

* **Определение**: Алгоритм или математическая функция, которая обучается на данных и используется для предсказания или принятия решений.
* **Роль в проекте**: Конкретные реализации моделей (например, нейронные сети) сравниваются между разными библиотеками.

## Библиотеки машинного обучения

**scikit-learn**:

* **Описание**: Популярная библиотека Python для машинного обучения, предоставляющая простой и эффективный инструментарий для обработки данных.
* **Роль**: Используется для реализации моделей с упором на простоту и скорость разработки.

**TensorFlow**:

* **Описание**: Открытая библиотека для численных вычислений и машинного обучения, разработанная Google.
* **Роль**: Используется для построения и обучения нейронных сетей с возможностью гибкой настройки и оптимизации.

**PyTorch**:

* **Описание**: Открытая библиотека машинного обучения, разработанная Facebook, ориентированная на быстрое прототипирование и исследования.
* **Роль**: Предоставляет гибкость и контроль при создании и обучении моделей.

## Датасеты (Наборы данных)

**California Housing Dataset**:

* **Описание**: Набор данных, содержащий информацию о цене жилья в Калифорнии и различных характеристиках домов и районов.
* **Использование**: Применяется для задачи регрессии, где цель — предсказать цену дома.

**MNIST Dataset**:

* **Описание**: Набор данных, содержащий изображения рукописных цифр от 0 до 9.
* **Использование**: Применяется для задачи классификации, где цель — распознать цифру на изображении.

## Предобработка данных

**Определение**: Процесс подготовки данных к подаче в модель, включая очистку, преобразование и масштабирование.

**Методы**:

* + **Нормализация**: Приведение значений признаков к общему масштабу.
  + **Стандартизация**: Преобразование данных с использованием среднего и стандартного отклонения.
* **Роль в проекте**: Критически важна для корректного обучения моделей и справедливого сравнения.

## Разделение данных

* **Обучающая выборка (Training Set)**:

**Описание**: Часть данных, используемая для обучения модели.

* **Валидационная выборка (Validation Set)**:

**Описание**: Часть данных, используемая для настройки гиперпараметров и предотвращения переобучения.

* **Тестовая выборка (Test Set)**:

**Описание**: Независимый набор данных для окончательной оценки производительности модели.

**Роль в проекте**: Позволяет оценить обобщающую способность модели и сравнить результаты между моделями.

## Гиперпараметры

* **Определение**: Настраиваемые параметры модели, которые не обучаются автоматически, а устанавливаются перед началом обучения.

**Примеры**:

* **Скорость обучения (Learning Rate)**: Определяет, насколько сильно обновляются веса модели при каждой итерации.
* **Количество эпох (Epochs)**: Число полных проходов по обучающему набору данных.
* **Размер батча (Batch Size)**: Количество примеров, обрабатываемых перед обновлением параметров модели.

**Роль в проекте**: Настройка гиперпараметров влияет на производительность и сходимость моделей.

## Функции активации

* **Определение**: Нелинейные функции, применяемые к выходу нейронов, позволяющие моделировать сложные зависимости.
* **Роль в проекте**: Выбор функции активации влияет на обучение и производительность нейронной сети.

## Функции потерь (Loss Functions)

* **Определение**: Функции, которые измеряют разницу между предсказаниями модели и реальными значениями, и которые модель стремится минимизировать во время обучения.
* **Примеры**:
* **MSE (Mean Squared Error)**: Используется в регрессии, рассчитывает среднее квадратов ошибок.
* **Cross-Entropy Loss**: Используется в классификации, измеряет разницу между двумя вероятностными распределениями.
* **Роль в проекте**: Правильный выбор функции потерь зависит от задачи и влияет на успех обучения модели.

## Оптимизаторы

* **Определение**: Алгоритмы, которые используются для обновления параметров модели с целью минимизации функции потерь.
* **Роль в проекте**: Выбор оптимизатора влияет на скорость сходимости и качество обучения модели.

# Тема 5: Метрики оценки качества моделей

## Для регрессии

* **MSE (Mean Squared Error)**:
* **Определение**: Среднее значение квадратов разницы между предсказанными и реальными значениями.
* **Роль**: Показывает общую ошибку модели; чем меньше MSE, тем лучше модель.
* **MAE (Mean Absolute Error)**:
* **Определение**: Среднее абсолютных разниц между предсказанными и реальными значениями.
* **Роль**: Интерпретируется в тех же единицах, что и целевая переменная; менее чувствителен к выбросам, чем MSE.
* **( R²) (Коэффициент детерминации)**:
* **Определение**: Мера того, какая доля дисперсии зависимой переменной объясняется моделью.
* **Роль**: Значение от 0 до 1; чем ближе к 1, тем лучше модель объясняет вариативность данных.

## Для классификации

* **Accuracy (Точность)**:
* **Определение**: Доля правильно классифицированных примеров от общего числа примеров.
* **Роль**: Простая и интуитивная метрика; может быть недостаточной при дисбалансе классов.
* **Precision (Прецизионность)**:
* **Определение**: Доля истинно положительных предсказаний среди всех положительных предсказаний.
* **Роль**: Важно, когда критично уменьшить количество ложных срабатываний.
* **Recall (Полнота)**:
* **Определение**: Доля истинно положительных предсказаний среди всех реальных положительных случаев.
* **Роль**: Важно, когда критично обнаружить все положительные случаи.
* **F1-score**:
* **Определение**: Гармоническое среднее между Precision и Recall.
* **Роль**: Обеспечивает баланс между Precision и Recall.
* **ROC AUC (Area Under the ROC Curve)**:
* **Определение**: Площадь под кривой ошибок (ROC), графически отображающей соотношение между True Positive Rate и False Positive Rate.
* **Роль**: Мера способности модели различать классы; значение от 0.5 (случайное угадывание) до 1 (идеальная модель).
* **Log Loss**:
* **Определение**: Мера качества предсказанных вероятностей; штрафует сильнее за уверенные, но ошибочные предсказания.
* **Роль**: Используется для оценки моделей, предсказывающих вероятности классов.

## Переобучение и недообучение

* **Переобучение (Overfitting)**:
* **Определение**: Ситуация, когда модель слишком хорошо подстроилась под обучающие данные, включая шум и выбросы, и плохо обобщает на новые данные.
* **Роль**: Ведёт к ухудшению производительности на тестовых данных; важно обнаруживать и предотвращать.
* **Недообучение (Underfitting)**:
* **Определение**: Ситуация, когда модель не способна уловить основные зависимости в данных.
* **Роль**: Модель показывает плохие результаты как на обучающих, так и на тестовых данных.

## Регуляризация

* **Определение**: Методы, используемые для предотвращения переобучения путем добавления дополнительной информации или ограничений в модель.
* **Виды**:
* **L1-регуляризация (Lasso)**: Добавляет сумму абсолютных значений коэффициентов в функцию потерь.
* **L2-регуляризация (Ridge)**: Добавляет сумму квадратов коэффициентов в функцию потерь.
* **Dropout**: В нейронных сетях случайно "отключает" некоторые нейроны во время обучения.
* **Роль в проекте**: Помогает моделям обобщать данные и улучшает их производительность на новых данных.

## Разложение данных на батчи (Batching)

* **Определение**: Разбиение обучающего набора данных на небольшие подвыборки (батчи) для поочередной подачи в модель.
* **Роль**: Уменьшает потребление памяти и позволяет использовать статистические свойства для улучшения обучения.

## Кросс-валидация

* **Определение**: Метод оценки модели, при котором данные разбиваются на несколько частей (фолдов), и модель обучается и тестируется на разных комбинациях этих частей.
* **Роль**: Оценивает обобщающую способность модели и помогает в подборе гиперпараметров.

## Эпоха (Epoch)

* **Определение**: Полный проход по всему обучающему набору данных во время обучения модели.
* **Роль**: Число эпох определяет, сколько раз модель увидит каждый пример из обучающей выборки.

## Ранняя остановка (Early Stopping)

* **Определение**: Техника, при которой процесс обучения прекращается, если метрика на валидационном наборе перестаёт улучшаться.
* **Роль**: Предотвращает переобучение, позволяя модель остановиться на оптимальном моменте.

## Архитектура нейронной сети

* **Определение**: Структура нейронной сети, включая количество слоёв, количество нейронов в каждом слое и связи между ними.
* **Виды слоёв**:
* **Входной слой**: Принимает входные данные.
* **Скрытые слои**: Обрабатывают данные и извлекают признаки.
* **Выходной слой**: Генерирует предсказания модели.
* **Роль в проекте**: Выбор архитектуры влияет на способности модели учиться и обобщать.

## Оценка модели (Model Evaluation)

* **Определение**: Процесс использования метрик и методов для определения качества и производительности модели.
* **Роль в проекте**: Позволяет сравнить модели между собой и выбрать наилучшую.

## Обучение модели (Model Training)

* **Определение**: Процесс настройки параметров модели на основе обучающих данных для минимизации функции потерь.
* **Роль в проекте**: Ключевая стадия, на которой модели учатся делать предсказания.

## Дата-сайенс (Data Science)

* **Определение**: Междисциплинарная область, объединяющая статистику, анализ данных, информатику для извлечения знаний и инсайтов из данных.
* **Роль в проекте**: Общая область, в рамках которой проводится работа над проектом.

## Экспериментальные протоколы

* **Определение**: Структурированный план проведения экспериментов, включая условия, параметры и методы оценки.
* **Роль в проекте**: Обеспечивает справедливое и воспроизводимое сравнение моделей и библиотек.

## Фреймворки глубокого обучения

* **TensorFlow и Keras**:
* **Описание**: Фреймворки для построения и обучения глубоких нейронных сетей с поддержкой различных архитектур и методов обучения.
* **PyTorch**:
* **Описание**: Фреймворк, ориентированный на динамические вычислительные графы и гибкость в исследовательских проектах.
* **Роль в проекте**: Используются для реализации более сложных моделей и предоставляют возможности для глубокого обучения.

## Аппаратное ускорение

* **GPU (Graphics Processing Unit)**:
* **Определение**: Аппаратное обеспечение, способное выполнять параллельные вычисления, ускоряя обучение моделей.
* **Роль в проекте**: Использование GPU может значительно ускорить обучение, особенно для глубоких нейронных сетей.

## Предсказательная способность модели

* **Обобщающая способность**:
* **Определение**: Способность модели хорошо работать на невидимых данных.
* **Роль в проекте**: Ключевой критерий оценки модели; модели, которые обобщают, имеют высокую ценность в практических приложениях.

## Случайность и воспроизводимость

* **Фиксация случайных сидов**:
* **Определение**: Установка определённых значений генераторов случайных чисел для обеспечения воспроизводимости результатов.
* **Роль в проекте**: Позволяет повторять эксперименты и сравнивать результаты справедливо.

# Тема 6: Архитектура и методология проекта

## Общие сведения

Для проведения сравнения были выбраны две задачи:

**Классификация**: распознавание рукописных цифр на наборе данных **MNIST**.

**Регрессия**: прогнозирование стоимости жилья на наборе данных **California Housing Dataset**.

Цель состоит в том, чтобы реализовать модели для этих задач с использованием **scikit-learn**, **TensorFlow** и **PyTorch**, сравнить их производительность и удобство использования.

## scikit-learn

### Подход к реализации моделей

**scikit-learn** — это библиотека машинного обучения на языке Python, предоставляющая простые и эффективные инструменты для анализа данных и моделирования. Она широко используется для задач классификации, регрессии, кластеризации и многого другого.

Для задач классификации и регрессии мы использовали классические многослойные персептроны (MLP), реализованные в классах MLPClassifier и MLPRegressor соответственно.

### Архитектура моделей

### Классификация

* **Модель**: MLPClassifier
* **Архитектура**:
* **Входной слой**: 784 нейрона (каждое изображение MNIST размером 28x28 разворачивается в вектор длиной 784).
* **Два скрытых слоя**:
  + **Первый скрытый слой**: 64 нейрона, функция активации ReLU.
  + **Второй скрытый слой**: 64 нейрона, функция активации ReLU.
* **Выходной слой**: 10 нейронов (соответствует количеству классов от 0 до 9), функция активации softmax применяется автоматически внутри MLPClassifier.

### Регрессия

* **Модель**: MLPRegressor
* **Архитектура**:
* **Входной слой**: 8 нейронов (соответствует количеству признаков в датасете California Housing).
* **Два скрытых слоя**:
  + **Первый скрытый слой**: 64 нейрона, функция активации ReLU.
  + **Второй скрытый слой**: 64 нейрона, функция активации ReLU.
* **Выходной слой**: 1 нейрон (предсказание стоимости жилья).

### Методы обучения

* **Оптимизатор**: Параметр solver='adam' указывает на использование оптимизатора Adam.
* **Скорость обучения**: learning\_rate\_init=0.001.
* **Размер батча**: batch\_size=32.
* **Ранняя остановка**:
* Параметр early\_stopping=True активирует раннюю остановку обучения, если ошибка на валидационном наборе не улучшается.
* n\_iter\_no\_change=5 означает, что обучение прекратится, если в течение 5 эпох подряд не будет улучшения.
* **Валидация**: Доля данных для валидации во время обучения устанавливается через validation\_fraction=0.2.
* **Максимальное количество эпох**: max\_iter=100.
* **Инициализация весов**: random\_state=42 для воспроизводимости результатов.

## TensorFlow

### Подход к реализации моделей

**TensorFlow** — это платформа с открытым исходным кодом для машинного обучения, разработанная компанией Google. Использование високоуровневого API **Keras**, встроенного в TensorFlow, позволяет быстро и просто создавать и обучать нейронные сети.

### Архитектура моделей

### Классификация

* **Модель**: Sequential модель из Keras.
* **Архитектура**:
* **Входной слой**: keras.layers.Input(shape=(784,)) — вектор признаков изображений MNIST.
* **Два скрытых слоя**:
  + **Первый скрытый слой**: keras.layers.Dense(64, activation='relu').
  + **Второй скрытый слой**: keras.layers.Dense(64, activation='relu').
* **Выходной слой**: keras.layers.Dense(10, activation='softmax') — 10 нейронов для предсказания вероятностей классов.

### Регрессия

* **Модель**: Sequential модель из Keras.
* **Архитектура**:
* **Входной слой**: keras.layers.Input(shape=(8,)) — вектор признаков для регрессии.
* **Два скрытых слоя**:
  + **Первый скрытый слой**: keras.layers.Dense(64, activation='relu').
  + **Второй скрытый слой**: keras.layers.Dense(64, activation='relu').
* **Выходной слой**: keras.layers.Dense(1) — один нейрон для предсказания непрерывного значения.

### Методы обучения

**Компиляция модели**

* **Оптимизатор**: Adam с learning\_rate=0.001.
* **Функция потерь**:
* **Классификация**: sparse\_categorical\_crossentropy — используется для многоклассовой классификации с целочисленными метками классов.
* **Регрессия**: mean\_squared\_error — средняя квадратичная ошибка.
* **Метрики**:
* **Классификация**: accuracy — метрика точности.
* **Регрессия**: mean\_squared\_error.

**Обучение модели**

* **Размер батча**: batch\_size=32.
* **Количество эпох**: До 100, с ранней остановкой.
* **Ранняя остановка**:
* Использован колбэк EarlyStopping из Keras с параметрами:
  + patience=5: обучение прекращается, если в течение 5 эпох подряд нет улучшения на валидационном наборе.
  + restore\_best\_weights=True: после остановки восстанавливаются веса модели с наилучшим значением функции потерь на валидации.
* **Или**: реализован собственный колбэк CustomEarlyStopping для более гибкого контроля над процессом ранней остановки.
* **Валидация**: Передается в fit через параметр validation\_data=(X\_val, y\_val).

## PyTorch

### Подход к реализации моделей

**PyTorch** — это фреймворк для глубокого обучения с открытым исходным кодом, разработанный Facebook AI Research. Он предоставляет гибкий способ построения моделей нейронных сетей и контроля процесса обучения с использованием динамических вычислительных графов.

В PyTorch модели создаются путем определения классов, наследующих от nn.Module, в которых задается архитектура сети и метод forward для прямого распространения.

### Архитектура моделей

### Классификация

* **Модель**: Класс NeuralNetPyTorch, наследующий от nn.Module.
* **Архитектура**:
* **Входной слой**: nn.Linear(784, 64) — 784 входных признака (пикселей) преобразуются в 64 нейрона.
* **Первый скрытый слой**: nn.Linear(64, 64) с функцией активации ReLU.
* **Второй скрытый слой**: nn.Linear(64, 64) с функцией активации ReLU.
* **Выходной слой**: nn.Linear(64, 10) — 10 нейронов для классификации на 10 классов.
* **Прямое распространение**:
* Метод forward(self, x) определяет, как данные проходят через слои модели.

### Регрессия

* **Модель**: Класс NeuralNetPyTorch, наследующий от nn.Module.
* **Архитектура**:
* **Входной слой**: nn.Linear(8, 64) — 8 входных признаков.
* **Первый скрытый слой**: nn.Linear(64, 64) с функцией активации ReLU.
* **Второй скрытый слой**: nn.Linear(64, 64) с функцией активации ReLU.
* **Выходной слой**: nn.Linear(64, 1) — один нейрон для предсказания непрерывного значения.

### Методы обучения

**Функция потерь**

* **Классификация**: nn.CrossEntropyLoss() — функция потерь, сочетающая в себе LogSoftmax и NLLLoss, подходит для многоклассовой классификации.
* **Регрессия**: nn.MSELoss() — среднеквадратичная ошибка.

**Оптимизатор**

* **Оптимизатор**: torch.optim.Adam с коэффициентом обучения lr=0.001.

**Цикл обучения**

Обучение в PyTorch требует явной реализации цикла обучения:

**Эпохи**: Задается количество эпох обучения, например, num\_epochs=100.

**Обучение**:

* Модель переводится в режим обучения: model.train().
* Проходим по загрузчику данных (DataLoader), получая батчи данных.
* Для каждого батча:
  + **Обнуление градиентов**: optimizer.zero\_grad().
  + **Прямой проход**: outputs = model(X\_batch).
  + **Вычисление функции потерь**: loss = criterion(outputs, y\_batch).
  + **Обратный проход**: loss.backward().
  + **Обновление параметров модели**: optimizer.step().

**Валидация**:

* Модель переводится в режим оценки: model.eval().
* Вычисление функции потерь и метрик на валидационном наборе без вычисления градиентов (with torch.no\_grad()).

**Ранняя остановка**:

* Реализуется с помощью отслеживания функции потерь на валидационном наборе.
* Если функция потерь не улучшается в течение patience эпох, обучение прекращается.
* Можно сохранять наилучшие веса модели с помощью torch.save() и загружать их при остановке обучения.

**Отслеживание метрик**:

* В каждой эпохе сохраняются значения функции потерь и метрик (точности) для последующего анализа и построения кривых обучения.

## Сравнение подходов

### Простота реализации

* **scikit-learn**: Предоставляет высокоуровневые классы MLPClassifier и MLPRegressor, где большинство конфигураций скрыто от пользователя. Требует минимального количества кода для реализации и обучения моделей.
* **TensorFlow**: Используя Keras, можно быстро построить модели, определив их архитектуру в последовательном порядке. Предоставляет баланс между простотой и гибкостью.
* **PyTorch**: Требует более детальной реализации. Необходимо самостоятельно определить класс модели, явно прописать цикл обучения, обработки данных и управление обучением.

### Гибкость и контроль

* **scikit-learn**: Ограничен возможностями встроенных классов. Меньше контроля над процессом обучения и архитектурой моделей.
* **TensorFlow**: Предоставляет высокоуровневый API с Keras, но также позволяет использовать низкоуровневые операции для более тонкой настройки.
* **PyTorch**: Максимальная гибкость и контроль над всеми аспектами модели и процесса обучения.

### Обучение и производительность

* **scikit-learn**: Быстрое обучение для простых моделей, но может быть медленнее на больших данных или сложных архитектурах.
* **TensorFlow**: Может использовать аппаратное ускорение (GPU), что ускоряет обучение моделей.
* **PyTorch**: Также поддерживает обучение на GPU, позволяет оптимизировать обучение для более сложных моделей.

# Тема 7: Обеспечение безопасности

## Опасности использования сомнительных библиотек

### Вредоносный код

* **Загрузки из ненадёжных источников**: Установка библиотек из непроверенных репозиториев или сайтов может привести к тому, что вместе с библиотекой возможна загрузка вредоносного кода.
* **Троянские программы и бэкдоры**: Злоумышленники могут внедрять скрытые функции в библиотеку, которые выполняют нежелательные действия, такие как кража данных или получение несанкционированного доступа к системе.

### Уязвимости и эксплойты

* **Известные уязвимости**: Использование библиотек с известными уязвимостями без обновления до исправленных версий.
* **Отсутствие поддержки и обновлений**: Сомнительные или устаревшие библиотеки могут не получать обновления безопасности, что делает их уязвимыми для атак.

### Несовместимость лицензий

* **Нарушение лицензий**: Использование библиотек без учёта их лицензионных ограничений может привести к юридическим проблемам.
* **Скрытые условия использования**: Сомнительные библиотеки могут содержать неблагоприятные условия, которые влияют на способность использовать или распространять программное обеспечение.

## Рекомендации по обеспечению безопасности

### Использование надёжных источников

* **Официальные репозитории**: Устанавливать необходимо библиотеки из официальных и проверенных репозиториев, таких как PyPI для Python.
* **Верификация источников**: Если необходимо установить библиотеку вручную, убедитесь в том, что скачиваете её с официального сайта или репозитория разработчика.

### Проверка библиотек на наличие уязвимостей

* **Инструменты проверки безопасности**:
* **safety**: Проверяет установленные зависимости на известные уязвимости. bash pip install safety safety check
* **Bandit**: Анализирует код Python на предмет уязвимостей безопасности. bash pip install bandit bandit -r your\_project/
* **Обновление зависимостей**: Регулярно обновляйте библиотеки до последних стабильных версий, где исправлены известные уязвимости.

### Использование виртуальных окружений

* **Изолирование окружений**: Необходимо создавать виртуальные окружения для проектов (с помощью venv, virtualenv, conda), чтобы изолировать зависимости и уменьшить риск конфликтов.
* **Управление зависимостями**: Фиксируйте версии используемых библиотек в файле requirements.txt или Pipfile.lock.

### Проверка целостности пакетов

* **Анализ исходного кода**: Если возможно, необходимо просмотривать исходный код библиотеки на предмет подозрительных участков.
* **Проверка цифровых подписей и хешей**: Некоторые библиотеки предоставляют контрольные суммы или цифровые подписи для проверки целостности пакета.

# Тема 8: Обзор популярных инструментов для разработки на Python

## Интегрированные среды разработки (IDE) и текстовые редакторы

### PyCharm

* **Описание**: PyCharm — мощная интегрированная среда разработки (IDE) для Python от компании JetBrains.
* **Особенности**:
* Умное автодополнение кода
* Мощные инструменты отладки
* Встроенная поддержка Django, Flask и других фреймворков
* Интеграция с системами контроля версий (Git, SVN)
* Поддержка разработки на научных платформах (Scientific Mode)

### Visual Studio Code (VS Code)

* **Описание**: Лёгкий и настраиваемый редактор кода от Microsoft с широкими возможностями расширения.
* **Особенности**:
* Расширения для Python, предоставляющие автодополнение, отладку, проверку синтаксиса
* Интеграция с терминалом
* Поддержка Docker и Kubernetes
* Встроенная поддержка Git
* Расширения для разных фреймворков и языков

### Atom

* **Описание**: Редактор кода, разработанный GitHub, настраивается с помощью пакетов и тем.
* **Особенности**:
* Менеджер пакетов для установки новых пакетов или создания своих
* Интеграция с Git и GitHub
* Автодополнение кода
* Возможность совместного редактирования (Teletype)

### Jupyter Notebook

* **Описание**: Интерактивная среда для выполнения блоков кода, обычно используемая в научных исследованиях и анализе данных.
* **Особенности**:
* Выполнение кода по ячейкам
* Визуализация данных прямо в ноутбуке
* Поддержка Markdown для оформления
* Простая интеграция с библиотеками для машинного обучения и анализа данных (NumPy, Pandas, Matplotlib)

## Инструменты управления пакетами и виртуальными окружениями

### pip

* **Описание**: Стандартный менеджер пакетов для Python.
* **Особенности**:
* Установка пакетов из PyPI
* Управление зависимостями проекта
* Прост в использовании

### virtualenv

* **Описание**: Инструмент для создания изолированных виртуальных окружений Python.
* **Особенности**:
* Позволяет создавать отдельные окружения с собственными пакетами и зависимостями
* Избегает конфликтов версий пакетов между проектами

### venv (встроенный модуль)

* **Описание**: Модуль для создания виртуальных окружений, встроенный в стандартную библиотеку Python начиная с версии 3.3.
* **Особенности**:
* Аналогичен virtualenv
* Не требует дополнительной установки

## Инструменты для тестирования

### unittest

* **Описание**: Стандартный модуль Python для написания юнит-тестов.
* **Особенности**:
* Входит в стандартную библиотеку
* Обеспечивает основные возможности для создания и выполнения тестов

### pytest

* **Описание**: Мощный фреймворк для тестирования, который позволяет писать простые и масштабируемые тесты.
* **Особенности**:
* Прост в использовании и легко расширяем
* Поддержка фикстур для настройки тестовой среды
* Богатая экосистема плагинов

## Инструменты для отладки

### pdb

* **Описание**: Стандартный отладчик Python.
* **Особенности**:
* Позволяет пошагово выполнять код
* Устанавливать точки останова
* Интерактивно исследовать состояние программы

### ipdb

* **Описание**: Расширение pdb с интеграцией IPython.
* **Особенности**:
* Предоставляет продвинутый интерфейс с возможностями IPython
* Поддержка автодополнения и синтаксической подсветки

### PyCharm Debugger

* **Описание**: Отладчик, встроенный в PyCharm.
* **Особенности**:
* Графический интерфейс для отладки
* Просмотр значений переменных
* Поддержка отладки многопоточных и асинхронных приложений

### Visual Studio Code Debugger

* **Описание**: Интегрированный отладчик в VS Code.
* **Особенности**:
* Поддержка точек останова, пошагового выполнения
* Интеграция с терминалом
* Просмотр текущего состояния программы

## Инструменты для статического анализа кода

### pylint

* **Описание**: Инструмент для проверки качества кода Python.
* **Особенности**:
* Проверка стиля кода по стандартам PEP 8
* Выявление ошибок и потенциальных проблем
* Расчёт оценки кода

### black

* **Описание**: Форматер кода, который автоматически форматирует код по стандартам.
* **Особенности**:
* Обеспечивает единообразный стиль кода
* Не требует настройки и имеет минимальные опции

## Средства управления версиями и совместной работы

### Git

* **Описание**: Распределённая система контроля версий.
* **Особенности**:
* Отслеживание изменений в коде
* Ветвление и слияние
* Поддержка работы в команде

### GitHub, GitLab, Bitbucket

* **Описание**: Веб-сервисы для хостинга репозиториев Git.
* **Особенности**:
* Совместная работа над кодом
* Системы управления задачами и bug tracking
* Интеграция с CI/CD инструментами

## Инструменты автоматизации и CI/CD

### Jenkins

* **Описание**: Сервер автоматизации с открытым исходным кодом.
* **Особенности**:
* Настройка процессов сборки, тестирования и деплоя
* Поддержка множества плагинов
* Расширяемость и интеграция с различными инструментами

### Travis CI

* **Описание**: Облачная платформа для непрерывной интеграции.
* **Особенности**:
* Автоматизация сборки и тестирования
* Интеграция с GitHub
* Поддержка множества языков и платформ

## GitHub Actions

* **Описание**: Инструмент для автоматизации рабочего процесса прямо в GitHub.
* **Особенности**:
* Настройка автоматических действий на основе событий
* Поддержка CI/CD
* Большое количество готовых действий в Marketplace

## Фреймворки для веб-разработки

### Django

* **Описание**: Высокоуровневый веб-фреймворк, поощряющий быструю разработку и чистый, прагматичный дизайн.
* **Особенности**:
* Полный набор инструментов для создания веб-приложений
* ORM для работы с базами данных
* Административный интерфейс «из коробки»

### Flask

* **Описание**: Микрофреймворк для веб-разработки, минималистичный и расширяемый.
* **Особенности**:
* Простота и гибкость
* Возможность добавления необходимых расширений
* Подходит для создания API и небольших приложений

# Библиотеки для научных вычислений и машинного обучения

### NumPy

* **Описание**: Библиотека для работы с многомерными массивами и матрицами.
* **Особенности**:
* Высокопроизводительные операции над массивами
* Большое количество математических функций

### Pandas

* **Описание**: Библиотека для анализа и манипулирования данными.
* **Особенности**:
* Структуры данных Series и DataFrame
* Простые методы чтения из различных источников данных
* Мощные инструменты для обработки и анализа данных

### Matplotlib и Seaborn

* **Описание**: Библиотеки для визуализации данных.
* **Особенности**:
* Построение графиков и диаграмм
* Настройка внешнего вида графиков
* Seaborn предоставляет более привлекательные стили и упрощённый синтаксис

### scikit-learn

* **Описание**: Библиотека для машинного обучения.
* **Особенности**:
* Большое количество алгоритмов для классификации, регрессии, кластеризации
* Инструменты для предобработки данных
* Простота использования и хорошая документация

### TensorFlow и Keras

* **Описание**: Платформа для численных вычислений и машинного обучения, Keras — высокоуровневый API для TensorFlow.
* **Особенности**:
* Разработка и обучение нейронных сетей
* Поддержка распределённого обучения
* Большое сообщество и множество примеров

### PyTorch

* **Описание**: Фреймворк для машинного обучения, ориентированный на разработку и исследование.
* **Особенности**:
* Динамические вычислительные графы
* Интуитивный и питоничный синтаксис
* Активное использование в научных исследованиях

## Другие полезные инструменты

### Cookiecutter

* **Описание**: Инструмент для создания шаблонов проектов.
* **Особенности**:
* Генерация структуры проекта по заданному шаблону
* Поддержка различных типов проектов (пакеты, веб-приложения и др.)

### Sphinx

* **Описание**: Инструмент для генерации документации.
* **Особенности**:
* Поддержка reStructuredText и Markdown
* Автоматическая генерация документации из docstrings
* Настраиваемые темы и расширения

### Ansible

* **Описание**: Инструмент для автоматизации конфигурации и управления серверами.
* **Особенности**:
* Простые описания задач на YAML
* Безагентная архитектура
* Использование модулей для выполнения команд на удалённых машинах

Python имеет богатый набор инструментов, которые покрывают практически все аспекты разработки: от написания кода и отладки до тестирования, деплоя и автоматизации задач. Выбор конкретных инструментов зависит от потребностей, специфики проекта и личных предпочтений.

# Тема 9: Проектирование приложения

## Планирование разработки в Google Colab

### Особенности использования Colab

**Google Colab** — это облачная платформа, предоставляющая среду выполнения Jupyter Notebook с поддержкой Python и аппаратного ускорения (GPU и TPU). Использование Colab влияет на планирование разработки следующим образом:

* **Доступность**: Работа в Colab возможна из любого места с доступом к интернету, что облегчает командную работу и доступ к проекту.
* **Аппаратное ускорение**: Возможность бесплатно использовать GPU и TPU для ускорения обучения моделей.
* **Предустановленные библиотеки**: Большинство необходимых библиотек уже установлено, что ускоряет настройку окружения.
* **Облачное хранение**: Данные и модели могут быть сохранены на Google Диске или других облачных сервисах.

# Тема 10: Основные требования для реализации проекта

## Технические требования

**Оборудование**

* **Компьютер с достаточной производительностью**:
* **Процессор**: Многоядерный CPU с высокой тактовой частотой поможет ускорить обработку данных и обучение моделей на CPU.
* **Оперативная память (RAM)**: Рекомендуется иметь не менее **8 ГБ** памяти, лучше **16 ГБ** или больше, чтобы работать с моделями и наборами данных без проблем.
* **Графический процессор (GPU)**:
  + Для ускорения обучения моделей в **TensorFlow** и **PyTorch** желательно наличие дискретного GPU от NVIDIA с поддержкой **CUDA**.
  + **CUDA Compute Capability**: Нужно проверить, чтобы GPU поддерживал необходимую версию CUDA для используемых версий TensorFlow и PyTorch.
* **Хранилище данных**:
* Достаточное свободное пространство на диске для хранения наборов данных, результатов обучения и визуализаций.

**Интернет-соединение**

* Необходимо для загрузки наборов данных, установки библиотек и, при необходимости, обращения к онлайн-ресурсам и документации.

## Программные требования

**Операционная система**

* **Windows**, **macOS** или **Linux**: Все три библиотеки поддерживаются на основных операционных системах.

**Язык программирования**

* **Python**: Версия **3.6** или выше.

**Необходимые библиотеки и инструменты**

* **scikit-learn**: Рекомендуется версия, совместимая с версией Python (обычно последняя стабильная версия).
* **TensorFlow**: Версия **2.x**, так как она включает в себя Keras API для удобного построения моделей.
* Если планируется использование GPU, желательна установка версии TensorFlow с поддержкой GPU и убеждаемся в наличии соответствующей версии **CUDA** и **cuDNN**.
* **PyTorch**: Последняя стабильная версия, совместимая с версией Python и CUDA (для GPU).
* **Дополнительные библиотеки**:
* **NumPy**: Для работы с массивами и векторами.
* **pandas**: Для загрузки и предварительной обработки данных.
* **matplotlib** и **seaborn**: Для построения графиков и визуализаций.
* **scikit-learn**: Помимо моделей, пригодится для предобработки данных и вычисления метрик.
* **SciPy**: Для научных вычислений.
* **torchvision** (для PyTorch): Если планируется работа с изображениями и использование встроенных наборов данных.
* **OpenCV** или **Pillow**: Для работы с изображениями, если потребуется.
* **Jupyter Notebook** или **JupyterLab**: Для удобной работы в интерактивных блокнотах.
* **Среда разработки**:
* Рекомендуется использовать IDE или текстовый редактор с поддержкой Python. Примеры: **PyCharm**, **Visual Studio Code**, **Atom**.

## Установка и настройка

* Устанавливаем все библиотеки и убеждаемся, что они корректно работают. Проверяем версии и совместимость.
* **Виртуальное окружение**:
* Рекомендуется создать виртуальное окружение (например, с помощью **venv** или **conda**), чтобы изолировать зависимости и избежать конфликтов между библиотеками.

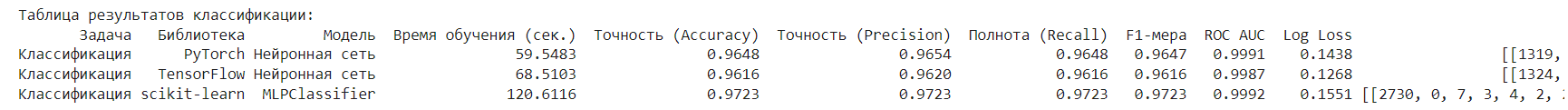
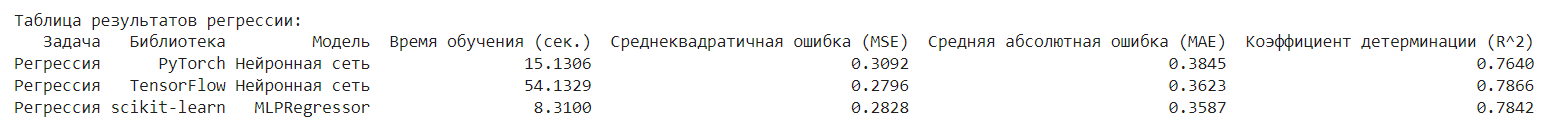
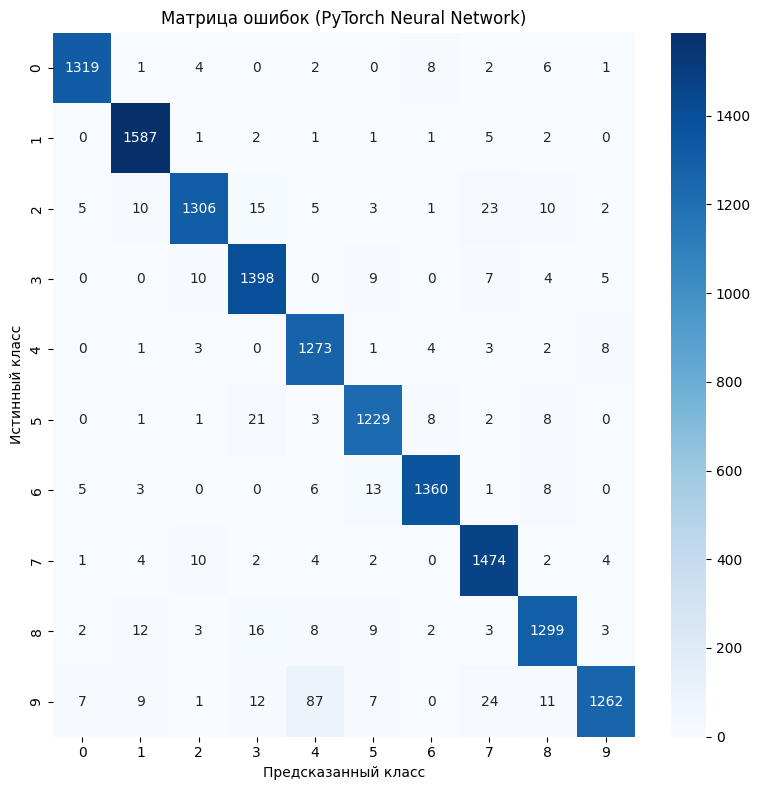
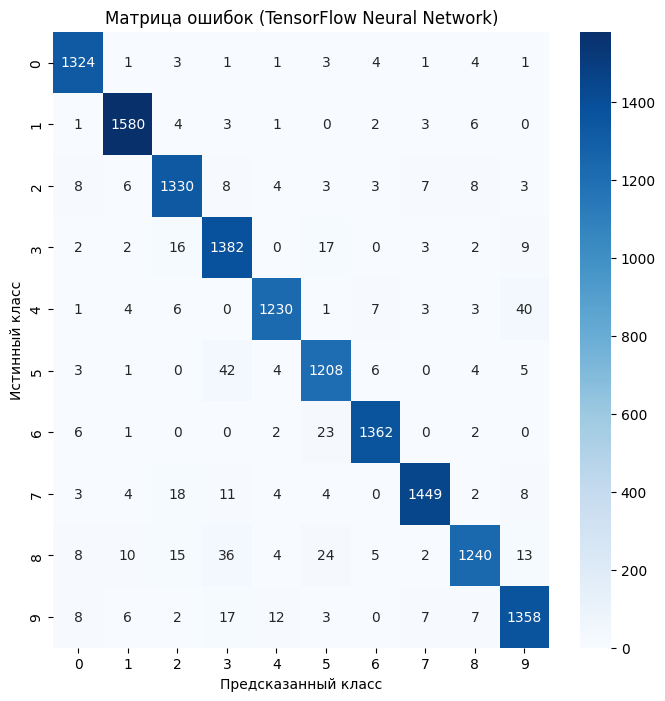
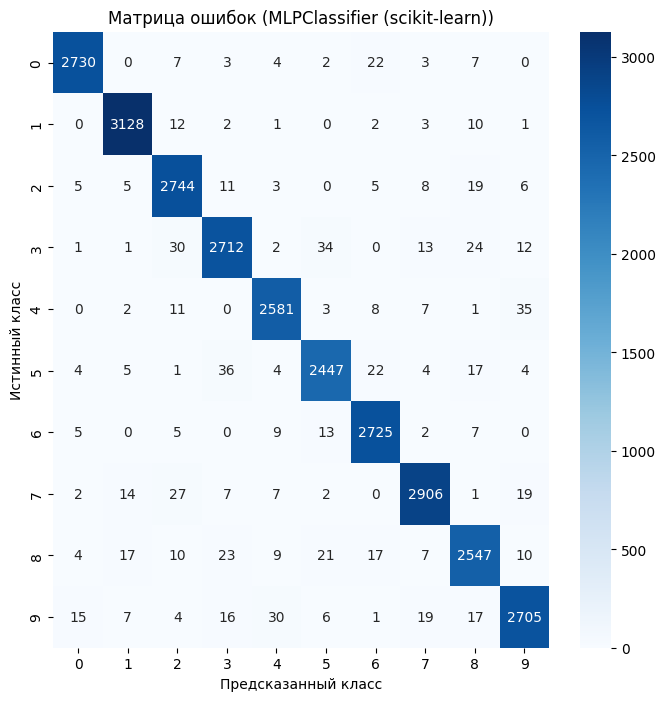
## Данные и наборы данных

**Выбор и загрузка наборов данных**

* **Задача классификации**:
* **MNIST**: Набор данных с изображениями рукописных цифр.
* **Задача регрессии**:
* **California Housing Dataset**: Содержит информацию о стоимости жилья в Калифорнии.
* **Способы загрузки**:
* В scikit-learn есть встроенные функции для загрузки этих наборов данных. Об этом, раскрывается в теме разработке.
* Можно также загрузить данные из открытых источников или воспользоваться **TensorFlow Datasets** или **torchvision.datasets**.

## Обработка и подготовка данных

* **Предварительная обработка**:
* Масштабирование и нормализация признаков.
* Разделение данных на обучающую, валидационную и тестовую выборки.
* Преобразование данных в формат, необходимый для каждой библиотеки (тензоры для PyTorch и TensorFlow).
* **Аугментация данных** (опционально для улучшения моделей):
* Применение методов аугментации изображений для расширения обучающего набора данных.



\*мартица ошибок слишком большая и малоинформативная в виде таблицы. Данные взяты из файла Google Colab, там можно ознакомиться детально с таблицей и дополнительным анализом.

# Тема 11: Анализ результатов

## Регрессия:

**Среднеквадратичная ошибка (MSE):** MSE измеряет средний квадрат разницы между предсказанными и фактическими значениями. Чем меньше MSE, тем лучше. В данном случае TensorFlow модель имеет наименьшую MSE (0.2796), что говорит о лучшей точности предсказаний по сравнению с другими моделями.

**Средняя абсолютная ошибка (MAE):** MAE измеряет среднее абсолютное значение разницы между предсказанными и фактическими значениями. Как и MSE, чем меньше MAE, тем лучше. TensorFlow модель снова лидирует с MAE 0.3623.

**Коэффициент детерминации (R²):** R² показывает, насколько хорошо модель объясняет дисперсию зависимой переменной. R² принимает значения от 0 до 1. Чем ближе R² к 1, тем лучше модель соответствует данным. TensorFlow модель имеет наивысший R² (0.7866), что указывает на лучшую общую производительность.

**Время обучения:** scikit-learn модель обучается значительно быстрее (8.31 секунды) по сравнению с PyTorch (15.13 секунды) и TensorFlow (54.13 секунды).

Хотя TensorFlow модель демонстрирует лучшую производительность с точки зрения MSE, MAE и R², она значительно дольше обучается. Scikit-learn модель, напротив, обучается намного быстрее, но ее метрики точности немного хуже. PyTorch занимает промежуточное положение.

Выбор лучшей модели зависит от конкретной задачи и приоритетов. Если точность важнее скорости обучения, TensorFlow — лучший выбор. Если скорость обучения критична, scikit-learn может быть предпочтительнее, несмотря на небольшое снижение точности. PyTorch может быть хорошим компромиссом.

## Классификация:

**Точность (Accuracy):** Показывает долю правильно классифицированных объектов. Все модели имеют точность выше 96%, что указывает на их хорошую общую производительность. scikit-learn модель имеет самую высокую точность (97.23%).

**Точность (Precision):** Измеряет долю правильно классифицированных положительных объектов среди всех объектов, которые модель отнесла к положительным. Значения precision также высоки для всех моделей, близки к значениям accuracy, что говорит о небольшом количестве ложноположительных результатов. В идеале, модель должна минимизировать как ложноположительные, так и ложноотрицательные результаты. Однако, часто приходится искать баланс между ними, в зависимости от задачи.

**Полнота (Recall):** Измеряет долю правильно классифицированных положительных объектов среди всех фактически положительных объектов. Аналогично precision, значения recall высоки и близки к accuracy, что свидетельствует о малом количестве ложноотрицательных результатов.

**F1-мера:** Гармоническое среднее между precision и recall. Высокие значения F1-меры (также близкие к accuracy) подтверждают баланс между precision и recall.

**ROC AUC:** Площадь под ROC-кривой. Значения близкие к 1 указывают на отличную способность моделей различать классы.

**Log Loss:** Измеряет уверенность модели в своих предсказаниях. Чем ниже значение Log Loss, тем лучше. `TensorFlow` модель имеет наименьший Log Loss (0.1268).

**Матрица ошибок:** Показывает количество правильно и неправильно классифицированных объектов для каждого класса. По главной диагонали расположены *истинно положительные* значения (correctly classified), остальные ячейки представляют собой ошибки.

Матрицы ошибок для моделей **scikit-learn**, **PyTorch** и **TensorFlow** имеют размер 10x10, что означает 10 классов. Большая часть значений сосредоточена по главной диагонали, что говорит о хорошей классификации. Можно проанализировать недиагональные элементы, чтобы понять, какие классы чаще всего путаются между собой. Например, в матрице PyTorch видно, что класс 4 иногда путается с классом 9 (87 объектов).

**Время обучения:** scikit-learn модель обучалась значительно дольше (120 секунд), чем модели на PyTorch (59 секунд) и TensorFlow (68 секунд).

Все три модели показывают отличные результаты. Модель scikit-learn имеет немного лучшую точность, но обучается значительно дольше. Модель TensorFlow имеет наименьший Log Loss. Выбор лучшей модели зависит от приоритетов: если важна скорость обучения, то PyTorch или TensorFlow предпочтительнее, если важна максимальная точность - scikit-learn. Анализ матрицы ошибок позволяет выявить проблемные классы и, возможно, улучшить модель, например, подобрав гиперпараметры или увеличив объем данных для обучения.

# Тема 12: Обзор выполненной работы

## Постановка целей и задач

* **Главная цель**: Провести сравнительный анализ библиотек scikit-learn, TensorFlow и PyTorch в задачах классификации и регрессии.
* **Основные задачи проекта**:
* Выбрать подходящие наборы данных для каждой задачи.
* Реализовать модели на каждой из библиотек с использованием сопоставимых архитектур.
* Обучить модели и собрать метрики производительности.
* Проанализировать результаты и сделать выводы о производительности и удобстве использования библиотек.
* Предоставить рекомендации по выбору библиотек и моделей для подобных задач.

## Подготовка данных

* **Задача классификации**:
* **Набор данных**: **MNIST** — содержит изображения рукописных цифр от 0 до 9.
* **Предобработка**:
  + Масштабирование пиксельных значений изображений в диапазон [0, 1].
  + Распределение данных на обучающую, валидационную и тестовую выборки.
* **Задача регрессии**:
* **Набор данных**: **California Housing** — содержит информацию о ценах на жилье в Калифорнии и различных характеристиках регионов.
* **Предобработка**:
  + Нормализация признаков (стандартизация с нулевым средним и единичным стандартным отклонением).
  + Разделение данных на обучающую, валидационную и тестовую выборки.

## Реализация моделей

* **Архитектура моделей**:
* Использована **единая архитектура** для каждой модели на всех платформах:
  + **Два скрытых слоя**, каждый из которых содержит **64 нейрона**.
  + **Функция активации**: **ReLU** для скрытых слоёв.
  + **Выходной слой**:
  + Для классификации: соответствует количеству классов с использованием функции активации Softmax (TensorFlow и PyTorch) или логистической функции (scikit-learn).
  + Для регрессии: один нейрон с линейной активацией.
* **Реализация в библиотеках**:
* **scikit-learn**:
  + **MLPClassifier** и **MLPRegressor** с соответствующими параметрами.
  + Простая настройка модели с минимальным количеством кода.
* **TensorFlow**:
  + Использован **Keras Sequential API** для построения моделей.
  + Возможность тонкой настройки и добавления callback-функций для управления обучением.
* **PyTorch**:
  + Созданы классы моделей, наследующие от **nn.Module**.
  + Требует более явного определения архитектуры и процесса обучения, но предоставляет большую гибкость.

## Обучение моделей

* **Общие параметры обучения**:
* **Оптимизатор**: **Adam** с параметрами по умолчанию.
* **Функция потерь**:
  + Для регрессии: **Среднеквадратичная ошибка** (MSE).
  + Для классификации: **Кросс-энтропия** (Cross-Entropy Loss).
* **Контроль обучения**:
  + Использование **ранней остановки** (Early Stopping) для предотвращения переобучения.
  + Мониторинг метрик на валидационной выборке.
* **Особенности обучения**:
* Обеспечение сопоставимости условий обучения между моделями.
* Фиксация генераторов случайных чисел для воспроизводимости результатов.

## Сбор и анализ результатов

* **Метрики оценки**:
* **Для регрессии**:
  + **Среднеквадратичная ошибка (MSE)**.
  + **Средняя абсолютная ошибка (MAE)**.
  + **Коэффициент детерминации (R²)**.
* **Для классификации**:
  + **Доля правильных ответов (Accuracy)**.
  + **Точность (Precision)**, **Полнота (Recall)**, **F1-мера**.
  + **ROC AUC** и **Log Loss**.
* **Время обучения**:
* Замер времени обучения каждой модели для оценки эффективности.
* **Анализ результатов**:
* Сравнение метрик между моделями и библиотеками.
* Построение графиков кривых обучения, матриц ошибок.
* Интерпретация причин различий в результатах.

## Итоги сравнения

### Регрессия:

* **TensorFlow** показал наименьшие значения MSE и MAE, но обучался дольше.

**Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание**

* **scikit-learn** быстро обучился и показал результаты, близкие к TensorFlow.

Изображение выглядит как текст, Шрифт, белый, снимок экрана

Автоматически созданное описание

* **PyTorch** обучился быстрее TensorFlow, но метрики качества чуть ниже.

Изображение выглядит как текст, Шрифт, белый, алгебра

Автоматически созданное описание

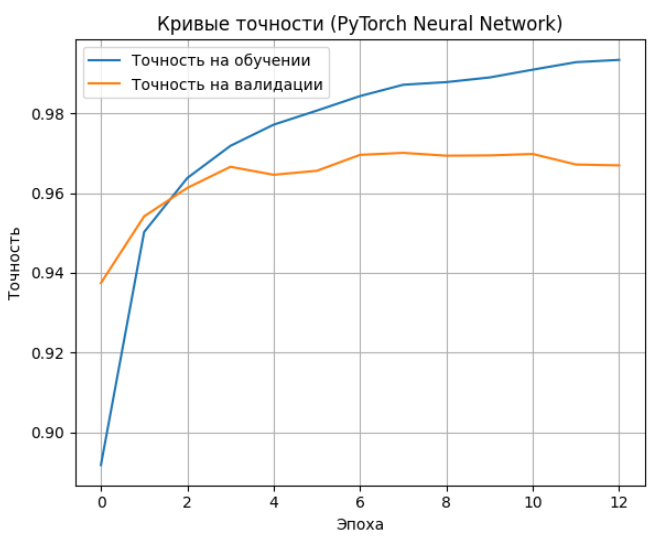
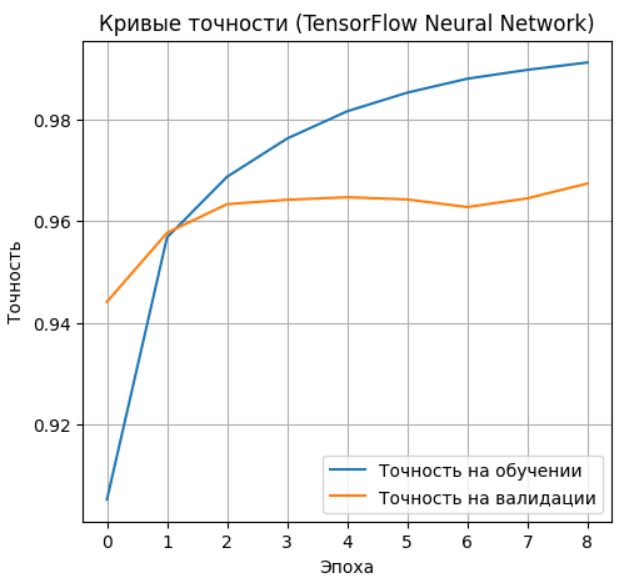
### Классификация:

* **scikit-learn** продемонстрировал наивысшую точность, но потребовал больше времени на обучение.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

* **PyTorch** и **TensorFlow** обучились быстрее, с незначительно меньшими показателями точности.



## Выводы и рекомендации

### Особенности библиотек:

* **scikit-learn**:
  + Идеален для быстрых прототипов и задач, не требующих сложных нейронных сетей.
  + Прост в использовании, с минимальным объемом кода для запуска моделей.
* **TensorFlow**:
  + Обеспечивает баланс между простотой и возможностями глубокой настройки моделей.
  + Подходит для сложных архитектур и имеет развитый экосистему (Keras, TensorBoard).
* **PyTorch**:
  + Предоставляет максимальную гибкость и контроль над процессом обучения.
  + Широко используется в научных исследованиях и при разработке новых подходов.

### Рекомендации по выбору моделей:

* **Для задач классификации на изображениях** (как MNIST) рекомендуется использовать **свёрточные нейронные сети** (CNN) для лучшего распознавания паттернов.
* **Для задач регрессии** следует рассмотреть **ансамблевые методы** (Random Forest, Gradient Boosting) для возможности моделирования нелинейных зависимостей.
* **Подбор гиперпараметров** и **тщательная настройка** могут существенно улучшить результаты моделей на всех библиотеках.

Проект предоставил **глубокое понимание** сильных и слабых сторон библиотек **scikit-learn**, **TensorFlow** и **PyTorch** в контексте задач регрессии и классификации. Был осуществлён целостный цикл работы: от подготовки данных и реализации моделей до анализа результатов и вынесения рекомендаций.

Полученные результаты подтверждают, что **нет универсального решения**, и выбор инструмента зависит от конкретных требований задачи, доступных ресурсов и опыта разработчика. **scikit-learn** подходит для простых и средних по сложности задач с ограниченным временем разработки. **TensorFlow** и **PyTorch** предоставляют расширенные возможности для глубокого обучения, с различиями в удобстве и гибкости.

Выполнение данного проекта позволило:

* Приобрести практический опыт работы с тремя основными библиотеками машинного обучения и глубокого обучения.
* Понять важность тщательной настройки моделей и предобработки данных.
* Осознать необходимость балансирования между качеством моделей и требуемыми ресурсами (время обучения, вычислительные мощности).
* Научиться проводить систематический анализ и сравнение моделей.

Это знание будет полезно при решении будущих задач машинного обучения и выборе наиболее подходящих инструментов и подходов для эффективного достижения поставленных целей.

# Тема 13: Заключение

## Выбор и подготовка данных

* **Задача регрессии**: Использован набор данных **California Housing**, который содержит информацию о ценах на жилье в Калифорнии и различных характеристиках районов.
* **Задача классификации**: Использован набор данных **MNIST**, содержащий изображения рукописных цифр от 0 до 9.
* **Предобработка данных**:
* **Масштабирование признаков**: Нормализация или стандартизация данных для ускорения обучения и улучшения сходимости моделей.
* **Разделение данных**: Данные разделены на обучающую, валидационную и тестовую выборки для оценки обобщающей способности моделей.

## Реализация моделей

* **scikit-learn**:
* **Регрессия**: MLPRegressor с двумя скрытыми слоями по 64 нейрона.
* **Классификация**: MLPClassifier с аналогичной архитектурой.
* **TensorFlow**:
* Использован **Keras Sequential API** для создания нейронных сетей с двумя скрытыми слоями по 64 нейрона, функции активации **ReLU**, выходной слой с соответствующей функцией активации для задачи.
* **PyTorch**:
* Реализованы классы моделей, наследующие от nn.Module, с аналогичной архитектурой.
* Написаны циклы обучения и валидации вручную, что предоставило больше контроля над процессом обучения.

## Обучение моделей

* **Общие параметры обучения**:
* **Оптимизатор**: **Adam** с параметрами по умолчанию.
* **Функции потерь**: MSE для регрессии, Cross-Entropy для классификации.
* **Эпохи обучения**: Использована ранняя остановка для предотвращения переобучения.

## Оценка моделей

* Модели оценивались по следующим метрикам:
* **Регрессия**: Среднеквадратичная ошибка (MSE), Средняя абсолютная ошибка (MAE), Коэффициент детерминации (R²).
* **Классификация**: Точность (Accuracy), Точность (Precision), Полнота (Recall), F1-мера, ROC AUC, Log Loss.

## Результаты

### Результаты регрессии

* **TensorFlow** показал наилучшие метрики качества (наименьшие MSE и MAE, наибольшее R²), хотя время обучения было самым высоким.
* **scikit-learn** достиг близких к TensorFlow результатов при наименьшем времени обучения.
* **PyTorch** обучился быстрее TensorFlow, но качество модели было несколько ниже.

### Результаты классификации

* **scikit-learn** достиг наилучших результатов по основным метрикам качества, но время обучения было наибольшим.
* **PyTorch** и **TensorFlow** обучились быстрее, показывая немного меньшую точность.
* **TensorFlow** имел наименьший Log Loss, указывая на более уверенные предсказания вероятностей.

# Тема 14: Дальнейшие планы

## Расширение набора моделей и алгоритмов

### Использование более сложных архитектур

* **Свёрточные нейронные сети (Convolutional Neural Networks, CNN)**:
* **Что делать**: Реализовать CNN для задачи классификации изображений (например, MNIST).
* **Как развивать**:
  + Построить модели с несколькими свёрточными и пуллинговыми слоями.
  + Использовать Batch Normalization и Dropout.
* **Ожидаемые результаты**:
  + Улучшение точности классификации.
  + Понимание работы CNN и их преимуществ для обработки изображений.
* **Рекуррентные нейронные сети (Recurrent Neural Networks, RNN) и LSTM**:
* **Что делать**: Применить RNN или LSTM для задач, связанных с последовательностями (если есть подходящие данные).
* **Ожидаемые результаты**:
  + Изучение возможностей обработки последовательных данных.
  + Расширение навыков работы с временными рядами или текстом.

### Исследование ансамблевых методов

* **Random Forest, Gradient Boosting (XGBoost, LightGBM, CatBoost)**:
* **Что делать**: Реализовать и сравнить ансамблевые методы для задач регрессии и классификации.
* **Как развивать**:
  + Настроить гиперпараметры моделей.
  + Понять, как эти модели обрабатывают нелинейности и взаимодействия признаков.
* **Ожидаемые результаты**:
  + Потенциальное улучшение метрик качества.
  + Понимание работы ансамблевых методов и их применения.

### Включение дополнительных алгоритмов машинного обучения

* **Методы снижения размерности**:
* **PCA (Principal Component Analysis)**, **t-SNE**, **UMAP**.
* **Что делать**: Применить эти методы для визуализации данных и выявления скрытых структур.
* **Ожидаемые результаты**:
  + Улучшение понимания данных.
  + Возможность снизить размерность данных для ускорения обучения моделей.
* **Кластеризация**:
* **Алгоритмы**: K-Means, DBSCAN, Hierarchical Clustering.
* **Что делать**: Исследовать возможность кластеризации данных и её влияние на модели.
* **Ожидаемые результаты**:
  + Выявление скрытых групп в данных.
  + Возможность улучшить предобработку данных.

## Оптимизация моделей и гиперпараметров

### Автоматизированный подбор гиперпараметров

* **Grid Search и Random Search**:
* **Что делать**: Организовать систематический поиск оптимальных значений гиперпараметров.
* **Как развивать**:
  + Использовать GridSearchCV в scikit-learn или аналогичные инструменты в TensorFlow и PyTorch.
* **Ожидаемые результаты**:
  + Улучшение качества моделей за счёт оптимальных настроек.
* **Байесовская оптимизация (Bayesian Optimization)**:
* **Инструменты**: Optuna, Hyperopt, Scikit-Optimize.
* **Что делать**: Применить более продвинутые методы оптимизации гиперпараметров.
* **Ожидаемые результаты**:
  + Более эффективный поиск оптимальных параметров.
  + Изучение передовых методов оптимизации.

## Регуляризация и борьба с переобучением

### Добавление методов регуляризации:

* **L1/L2 регуляризация**, **Dropout**, **Early Stopping**.
* **Что делать**: Исследовать влияние различных методов регуляризации на модели.
* **Ожидаемые результаты**:
  + Улучшение обобщающей способности моделей.
  + Понимание, как предотвращать переобучение.

## Улучшение предобработки и расширение данных

### Аугментация данных

* **Что делать**: Применить методы аугментации для увеличения объёма данных и повышения их разнообразия.
* **Как развивать**:
* В классификации изображений: повороты, сдвиги, масштабирование, отражения.
* В других типах данных: генерация синтетических примеров.
* **Ожидаемые результаты**:
* Повышение устойчивости моделей к вариациям данных.
* Улучшение метрик качества на тестовых данных.

### Обработка дисбаланса классов

* **Что делать**: Если в данных наблюдается несбалансированность классов, применить методы её устранения.
* **Методы**:
* **Oversampling** (SMOTE), **Undersampling**, **Class Weights**.
* **Ожидаемые результаты**:
* Улучшение точности для миноритарных классов.
* Более справедливые и сбалансированные модели.

## Расширение набора данных

### Использование новых датасетов

* **Что делать**: Включить дополнительные наборы данных с разными характеристиками.
* **Примеры**:
* Для классификации: CIFAR-10, Fashion-MNIST.
* Для регрессии: Boston Housing, Ames Housing.
* **Ожидаемые результаты**:
* Проверка моделей на новых данных.
* Оценка обобщающей способности моделей.
* Расширение опыта работы с разными типами данных.

### Обработка больших данных

* **Что делать**: Попробовать работать с более крупными наборами данных или высокоразмерными данными.
* **Как развивать**:
* Использовать распределённые вычисления.
* Изучить оптимизацию потребления памяти и времени.
* **Ожидаемые результаты**:
* Понимание масштабируемости моделей и библиотек.
* Навыки работы с большими объёмами данных.

## Интерпретируемость и объяснимость моделей

### Применение XAI (Explainable AI) методов

* **Инструменты**: SHAP, LIME, ELI5.
* **Что делать**: Анализировать вклад признаков в предсказания моделей.
* **Ожидаемые результаты**:
* Глубокое понимание того, как модели принимают решения.
* Возможность объяснить результаты модели пользователям или стейкхолдерам.

### Визуализация внутренних представлений

* **Что делать**: Визуализировать активации слоёв нейронных сетей, фильтры свёртки.
* **Ожидаемые результаты**:
* Инсайты о том, какие особенности данных важны для модели.
* Улучшение архитектуры моделей на основе полученной информации.

## Интеграция и деплой моделей

### Создание приложений или сервисов

* **Что делать**: Развернуть модели в виде веб-сервисов или мобильных приложений.
* **Инструменты**:
* **Flask**, **Django** для веб-приложений.
* **TensorFlow Lite**, **PyTorch Mobile** для мобильных приложений.
* **Ожидаемые результаты**:
* Практический опыт в развертывании моделей.
* Возможность предоставления моделей для реальных пользователей.

### Оптимизация и сжатие моделей

* **Методы**: Квантизация, прунинг, сжатие весов.
* **Что делать**: Уменьшить размер моделей и ускорить их работу для применения на устройствах с ограниченными ресурсами.
* **Ожидаемые результаты**:
* Повышение эффективности моделей.
* Возможность внедрения моделей в edge-устройства или IoT.

## Сравнение и оценка производительности

### Расширение метрик оценки

* **Что делать**: Включить дополнительные метрики, такие как время предсказания, потребление памяти.
* **Ожидаемые результаты**:
* Более полное понимание эффективности моделей.
* Возможность выбора моделей с учетом ограничений системы.

### Тестирование устойчивости моделей

* **Что делать**: Оценить, как модели справляются с данными, отличающимися от обучающих (adversarial examples, noisy data).
* **Ожидаемые результаты**:
* Понимание ограничений моделей.

# Тема 15: Потенциальные результаты и выгоды от развития проекта

## Углубление знаний и навыков:

Работа по указанным направлениям позволит значительно расширить свои компетенции в области машинного обучения и глубокого обучения.

## Улучшение практических результатов:

Применение более продвинутых моделей и методов приведет к повышению точности и эффективности моделей, делая их более применимыми в реальных задачах.

## Подготовка к промышленным проектам:

Опыт работы с развёртыванием моделей, оптимизацией и масштабированием подготовит к решению задач в индустрии.

## Научный вклад и публикации:

Глубокое исследование моделей и методов может привести к новым научным открытиям и публикациям.

## Карьерный рост:

Навыки и знания, приобретенные в процессе развития проекта, позволят стать более конкурентоспособным специалистом на рынке труда.

Развитие проекта может пойти по множеству направлений, от углубления технических навыков и исследований новых алгоритмов до практического применения и развертывания моделей в реальных системах. Важно выбирать те направления, которые наиболее соответствуют интересам, целям и карьерным планам.