САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт копьютерных наук и кибербезопасности

УТВЕРЖДАЮ
Доцент ВШ ПИ
_____ А. В. Щукин
«_____» _____ 2025г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Ильчуку Александру Евгеньевичу гр. 5130903/10301

1. Тема работы: Разработка средств автоматического извлечения корректирующих и предупредительных действий из репозиториев кода с помощью инструментов анализа.
2. Срок сдачи студентом законченной работы: 17.05.2025.
3. Исходные данные по работе:
   3.1. An experience in automatically extracting CAPAs from code repositories: [Электронный ресурс]. URL: https://arxiv.org/pdf/2212.09910 (дата обращения: 12.12.2024).
   3.2. A meta-analytical comparison of Naive Bayes and Random Forest for software defect prediction: [Электронный ресурс]. URL: https://www.researchgate.net/publication/350459831_A_metaanalytical_comparison_of_Naive_Bayes_and_Random_Forest_for_software_defect_prediction (дата обращения: 12.12.2024).
   3.3. Examining the Success of an Open Source Software Project Analysing Its Repository: [Электронный ресурс]. URL: https://doi.org/10.5281/zenodo.10046579 (дата обращения: 12.12.2024).
   3.4. Github API documentation: [Электронный ресурс]. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28 (дата обращения: 12.12.2024).
   3.5. PyGithub documentation: [Электронный ресурс]. URL: https://pygithub.readthedocs.io/en/stable/ (дата обращения: 12.12.2024).

3.6. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation: [Электронный ресурс]. URL: https://www.mdpi.com/2079-9292/9/8/1295 (дата обращения: 12.12.2024).

4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Обзор литературы по теме ВКР.
   4.2. Исследование программных продуктов в сфере анализа качества репозиториев кода.
   4.3. Изучение и сравнительный анализ средств извлечения предупреждающих действий.
   4.4. Разработка способа корректного извлечения метрик из репозиториев кода.
   4.5. Разработка метода обработки и анализа данных на основе изученных средств.
   4.6. Создание решения автоматического выделения предупреждающих действий на основе обработанных данных.
   4.7. Добавление веб инетрфейса с визуализацией выявленных аномалий.
   4.8. Тестирование разработанного программного решения.
5. Перечень графического материала (с указанием обязательных чертежей):
   5.1. Диаграмма вариантов использования.
   5.2. Архитектура разработанной программы.
6. Консультанты по работе:
   6.1. Ст. преподаватель ВШ ПИ, Е. Е. Андрианова (нормоконтроль).
7. Дата выдачи задания: 12.12.2024.

Руководитель ВКР В. А. Пархоменко

Консультант Е. Е. Андрианова

Задание принял к исполнению 12.12.2024

Студент А. Е. Ильчук

РЕФЕРАТ

На 28 с., 0 рисунков, 6 таблиц, 0 приложений
КЛЮЧЕВЫЕ СЛОВА: РЕПОЗИТОРИЙ, АНАЛИЗ ДАННЫХ, МЕТРИКИ, КАЧЕСТВО КОДА.[1]
Тема выпускной квалификационной работы: «Разработка средств автоматического извлечения корректирующих и предупредительных действий из репозиториев кода с помощью инструментов анализа»[2].
[3]

ABSTRACT

28 pages, 0 figures, 6 tables, 0 appendices
KEYWORDS: REPOSITORY, DATA ANALYSIS, METRICS, CODE QUALITY.
The subject of the graduate qualification work is «Developing tools for artificially extracting corrective preventive actions from a code repository using analysis tools».

---

[1]Всего **слов**: от 3 до 15. Всего **слов и словосочетаний**: от 3 до 5. Оформляются в именительном падеже множественного числа (или в единственном числе, если нет другой формы), оформленных по правилам русского языка. *Внимание! Размещение сноски после точки является примером как запрещено оформлять сноски.*

[2]Реферат **должен содержать**: предмет, тему, цель ВКР; метод или методологию проведения ВКР; результаты ВКР; область применения результатов ВКР; выводы.

[3]ОТ 1000 ДО 1500 печатных знаков (ГОСТ Р 7.0.99-2018 СИБИД) на русский или английский текст. Текст реферата повторён дважды на русском и английском языке для демонстрации подхода к нумерации страниц.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность исследования. Современные разработки программного обеспечения (ПО) становятся всё более сложными, требуя высоких стандартов качества и точного контроля за процессом разработки. В условиях глобализации и быстрого развития технологий компании стремятся не только создавать качественные продукты, но и эффективно управлять процессом их создания. Однако существующие подходы к анализу и управлению качеством ПО требуют значительных временных и человеческих ресурсов. Методы анализа данных и машинного обучения, которые уже зарекомендовали себя в смежных областях, могут быть использованы для автоматизации контроля качества и выявления проблем на ранних этапах. Одной из ключевых задач в этой области является анализ данных из репозиториев исходного кода, таких как GitHub.

Коммиты в репозиториях содержат важную информацию о внесённых изменениях: количество добавленных и удалённых строк кода, изменённые файлы, временные интервалы между изменениями. Анализ этих данных позволяет выявить потенциальные отклонения от нормального процесса разработки и предложить корректирующие и предупреждающие действия (CAPA). Несмотря на широкий спектр существующих инструментов для анализа данных из репозиториев, большинство из них либо недостаточно автоматизированы, либо не позволяют выявлять комплексные закономерности в данных.

Цель исследования: разработка системы, которая позволит автоматизировать процесс анализа коммитов и извлечения CAPA на основе методов машинного обучения и кластеризации.

Задачи исследования:

- Провести обзор существующих методов анализа данных из репозиториев кода.
- Изучить применимость методов кластеризации и алгоритмов машинного обучения для анализа коммитов.
- Разработать систему для автоматического извлечения данных о коммитах из нескольких репозиториев GitHub.
- Реализовать механизм выявления аномалий и классификации коммитов на основе предложенных методов.
- Создать интерактивный дашборд для визуализации результатов анализа.
- Оценить эффективность предложенного подхода на реальных данных.

Подробнее актуальность исследования и обзор методов рассмотрены в разделе 1.1.

Таким образом, исследование направлено на решение задачи повышения эффективности управления качеством программного обеспечения за счёт использования современных технологий анализа данных. Предложенная система должна не только автоматизировать процесс анализа данных, но и предоставлять разработчикам полезные рекомендации для улучшения качества кода и предотвращения потенциальных проблем.

ГЛАВА 1. ОБЗОР ИСТОЧНИКОВ ПО ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Современные исследования в области анализа данных из репозиториев кода

Современные исследования в области анализа данных из репозиториев кода сосредоточены на автоматизации процессов контроля качества и выявления отклонений на основе методов машинного обучения. Эти подходы позволяют не только выявлять проблемы, но и предотвращать их за счёт своевременных корректирующих действий.

Одной из ключевых задач является анализ коммитов, который рассматривается как источник ценной информации о процессе разработки. Коммиты содержат метрики, такие как количество добавленных и удалённых строк кода, изменённые файлы и временные интервалы между изменениями. Эти данные используются для анализа отклонений от нормального процесса разработки.

Использование метрик позволяет объективно оценить состояние проекта и выявить аномалии в процессе разработки. Наиболее часто используемые метрики включают:

- **Количество добавленных строк кода**: может указывать на крупные изменения или добавление новой функциональности.
- **Количество удалённых строк кода**: часто связано с рефакторингом или устранением избыточного кода.
- **Общее количество изменений**: помогает определить интенсивность работы над проектом.
- **Временные интервалы между коммитами**: являются индикатором ритма работы команды.

Эти метрики формируют основу для дальнейшего анализа и автоматической классификации действий разработчиков.

1.2. Выявление аномалий и использование паттернов

Исследования показывают, что анализ временных рядов метрик позволяет выявлять повторяющиеся паттерны, связанные с аномалиями. Z-нормализованное евклидово расстояние применяется для измерения схожести между временными рядами, где каждый временной ряд предварительно нормализуется путём вычитания среднего значения и деления на стандартное отклонение. Это делает ряды сопоставимыми

независимо от их масштаба. Этот метод позволяет выявить закономерности в изменениях метрик и сопоставить их с характерными действиями разработчиков. Анализ делится на три основных этапа:

A. Построение временных рядов на основе ключевых метрик.
B. Поиск паттернов с использованием алгоритмов кластеризации.
C. Связь обнаруженных паттернов с корректирующими и предупреждающими действиями (CAPA).

Таблица 1.1

Примеры паттернов аномалий и их интерпретация

Тип паттерна	**Возможные причины**	**Рекомендуемые действия**
Резкий рост добавления строк	Добавление новой функциональности	Проведение ревизии кода
Частые изменения в одном файле	Проблемы с архитектурой	Разделение задачи на более мелкие части
Длительные задержки между коммитами	Потеря фокуса команды	Уточнение приоритетов и планов

1.3. Разновидности методов машинного обучения

Современные алгоритмы машинного обучения используются для автоматизации анализа данных, так как они позволяют справляться с большими объёмами информации, высокой вариативностью и сложными взаимосвязями между метриками. В контексте анализа данных из репозиториев кода, выбор алгоритмов обусловлен их способностью выявлять закономерности в многомерных временных рядах, обеспечивать интерпретируемость данных, учитывать особенности категориальных и числовых данных. К часто используемым моделям относится:

– **Random Forest** – обеспечивает высокую точность (83%) и интерпретируемость.
– **XGBoost** – оптимизирован для больших данных, скорость обучения выше (82%).
– **CatBoost** – стабильно работает с категориальными данными (81%).
– **Deep Forest** – каскадная структура деревьев решений, эффективна для небольших проектов.

1.4. Подбор инструментальных средств

Разработка аналитической системы требует использования широкого спектра технических и инструментальных средств, которые обеспечивают поддержку на всех этапах разработки: от обработки данных до визуализации результатов. В данной главе представлен детальный анализ инструментальных средств, критерии

их выбора и обоснование принятия решений. Такой подход позволяет выбрать оптимальные технологии, соответствующие целям проекта и обеспечивающие его успешную реализацию.

1.4.1. Основной язык программирования

Для выполнения задач проекта был выбран Python. Выбор языка обоснован анализом следующих критериев:

- **Поддержка библиотек и инструментов**: Наличие развитой экосистемы для анализа данных, машинного обучения и визуализации.
- **Простота и скорость разработки**: Интуитивно понятный синтаксис, упрощающий реализацию сложных алгоритмов.
- **Производительность**: Достаточна для выполнения задач проекта при использовании оптимизированных библиотек.
- **Совместимость**: Возможность интеграции с различными инструментами и системами.

Таблица 1.2

Сравнение языков программирования

Критерий	**Python**	**R**	**C++**
Поддержка библиотек	Отличная	Хорошая	Отличная
Простота разработки	Высокая	Средняя	Низкая
Производительность	Средняя	Низкая	Высокая
Совместимость	Отличная	Средняя	Хорошая

Вывод: Python оказался оптимальным выбором благодаря универсальности, простоте и обширной поддержке библиотек. В отличие от R, Python лучше подходит для машинного обучения, а по сравнению с C++ обеспечивает более быстрый цикл разработки.

1.4.2. Библиотеки для анализа и машинного обучения

Для обработки данных и построения моделей машинного обучения в проекте используются инструменты, которые обеспечивают эффективность, простоту использования и соответствие требованиям задач. Основные выбранные библиотеки и модели:

– **Pandas**: Инструмент для работы с табличными данными, включающий фильтрацию, трансформацию, агрегацию и анализ.
– **Scikit-learn**: Универсальная библиотека для задач классического машинного обучения, включая классификацию и кластеризацию.
– **Deep Forest**: Каскадная структура деревьев решений, подходящая для небольших объёмов данных с высокой интерпретируемостью.

Таблица 1.3

Сравнение моделей машинного обучения

Критерий	**Random Forest**	**XGBoost**	**CatBoost**	**Deep Forest**
Область применения	Универсальный	Большие данные	Категориальные данные	Малые и средние данные
Производительность	Высокая	Высокая (GPU)	Высокая	Умеренная
Интерпретируемость	Высокая	Средняя	Средняя	Высокая
Работа с зависимостями	Простые связи	Сложные связи	Ограниченные связи	Глубокие взаимосвязи
Простота настройки	Простая	Сложная	Сложная	Простая

Выбор в пользу Deep Forest обусловлен его высокой интерпретируемостью и способностью выявлять сложные зависимости в данных.

1.4.3. Средства визуализации

Для представления данных и взаимодействия с пользователями были рассмотрены несколько инструментов визуализации.

Таблица 1.4

Сравнение инструментов визуализации

Критерий	**Plotly**	**Matplotlib**	**Seaborn**
Интерактивность	Высокая	Низкая	Низкая
Простота интеграции	Отличная	Средняя	Средняя
Поддержка сложных графиков	Отличная	Хорошая	Средняя

Выбран Plotly и Dash благодаря их интерактивности и возможности создания дашбордов.

1.4.4. Среда разработки

Для удобства работы над проектом была выбрана среда PyCharm.

1.4.5. Методы кластеризации и анализа данных

Для анализа данных выбраны методы кластеризации, такие как KMeans.

Таблица 1.5

Сравнение сред разработки

Критерий	**PyCharm**	**VS Code**	**Jupyter Notebook**
Отладка	Отличная	Хорошая	Ограниченная
Управление зависимостями	Простое	Требует расширений	Не поддерживается

Таблица 1.6

Сравнение алгоритмов кластеризации

Критерий	**KMeans**	**DBSCAN**	**Иерархическая кластеризация**
Производительность	Высокая	Средняя	Низкая
Простота реализации	Простая	Средняя	Сложная
Число кластеров	Задаётся заранее	Определяется динамически	Определяется визуально

1.5. Формулирование цели, задач и гипотез

Исходя из обзора предметной области и выбранных инструментальных средств, сформулируем цель, задачи и гипотезы для разработки системы автоматического анализа данных из репозиториев GitHub.

Цель: Создание системы автоматического извлечения и анализа данных коммитов из репозиториев GitHub с использованием методов кластеризации и машинного обучения для выявления корректирующих и предупреждающих действий (CAPA) и визуализации результатов анализа через интерактивный дашборд.

Задачи:

- Изучить потребности пользователей в автоматическом анализе данных репозиториев кода.
- Рассмотреть существующие методы и подходы для анализа коммитов, включая методы кластеризации и машинного обучения.
- Сформулировать ограничения и требования к системе для повышения её функциональности и точности.
- Разработать архитектуру системы, включающую автоматическое извлечение данных, их анализ и визуализацию.
- Внедрить алгоритмы кластеризации (например, k-средние) для группировки данных коммитов.
- Применить алгоритмы машинного обучения (случайные леса, глубокие леса) для классификации коммитов и выявления CAPA.
- Разработать и интегрировать визуализацию результатов анализа с использованием Plotly и Dash.

– Автоматизировать создание отчетов и pull request'ов с рекомендациями для GitHub.
– Протестировать систему, оценить её эффективность и точность работы.

1.6. Выводы

Обзор методов и инструментов анализа показал, что современные подходы эффективно справляются с задачами выявления проблем и улучшения процессов разработки. Анализ временных рядов является более эффективным методом по сравнению с традиционными подходами, так как он учитывает динамику изменений и позволяет выявлять закономерности, недоступные при статическом анализе. Существующие инструменты, такие как SonarQube, CodeClimate и GitPrime, предоставляют ценные возможности для анализа качества кода и производительности команды, но ограничены в своих функциях по сравнению с TOM. TOM выделяется благодаря интеграции анализа временных рядов и машинного обучения для предоставления корректирующих действий. Однако для повышения своей полезности TOM может быть дополнен следующими функциями: Визуализация процессов: Отображение динамики изменений для наглядного анализа. Расширенная аналитика: Добавление долгосрочных трендов и отчётов для стратегического управления проектами. Таким образом, TOM представляет собой перспективный инструмент, который с учётом доработок сможет удовлетворить потребности как разработчиков, так и менеджеров проектов, предоставляя комплексный подход к анализу и улучшению процессов разработки.

ГЛАВА 2. РАЗРАБОТКА МЕТОДА, АЛГОРИТМА, МОДЕЛИ ИССЛЕДОВАНИЯ

2.1. Обоснование выбора метода

В данной главе рассматриваются основные методы и алгоритмы, используемые для анализа данных из репозиториев программного кода. Выбор методов обоснован на основе современных исследований и их практической применимости в задачах анализа коммитов.

2.2. Формализация задачи

Основная цель исследования заключается в автоматическом анализе истории коммитов в репозиториях GitHub и выявлении потенциальных аномалий с последующей генерацией корректирующих действий (CAPA).
Формально, каждый коммит c_i представляется вектором признаков:

$$c_i = \{a_i, d_i, t_i, f_i, \tau_i\}, \tag{2.1}$$

где:
– a_i — количество добавленных строк кода,
– d_i — количество удаленных строк кода,
– t_i — общее число изменений (сумма добавленных и удаленных строк),
– f_i — количество измененных файлов,
– τ_i — время с момента последнего коммита.

2.3. Выбор алгоритмов анализа

Для обработки данных применяются следующие алгоритмы:

2.3.1. Метод кластеризации KMeans

Метод KMeans используется для установления пороговых значений на основе характеристик коммитов. Кластеризация выполняется по признакам a, d, t, f, τ, что позволяет разделить коммиты на нормальные и потенциально аномальные.

Data: Набор данных коммитов C
Result: Разделение на два кластера: нормальные и аномальные коммиты
1.1. Инициализировать KMeans с параметром $k = 2$
2.2. Применить алгоритм к данным C
3.3. Определить центры кластеров μ_1, μ_2
4.4. Рассчитать пороговые значения как среднее между центрами
5.5. Отнести каждый коммит c_i к соответствующему кластеру

2.3.2. Обучение моделей машинного обучения

Для классификации коммитов в дальнейшем используются два алгоритма:

– **Случайный лес (Random Forest)** — ансамблевый метод на основе деревьев решений, обеспечивающий высокую точность и устойчивость к шуму.
– **Наивный байесовский классификатор (Naive Bayes)** — простая вероятностная модель, используемая для базовой оценки данных.

Эксперименты показали, что случайный лес демонстрирует более высокую точность (1.0), в то время как наивный Байес достигает точности 0.8.

2.4. Глубокий лес (Deep Forest) для предсказания CAPA

Для повышения точности классификации была внедрена модель глубокого леса (Cascade Forest). Из-за несовместимости библиотеки deep-forest с Python 3.12 была создана виртуальная среда на Python 3.9.

Data: Набор данных коммитов C
Result: Предсказанные корректирующие действия CAPA
1.1. Разделить данные на тренировочный и тестовый наборы.
2.2. Обучить модель глубокого леса на тренировочных данных.
3.3. Оценить точность модели на тестовом наборе.
4.4. Применить модель к новым данным для предсказания корректирующих действий.

2.5. Заключение

В данной главе был рассмотрен выбор методов и алгоритмов для анализа коммитов в репозиториях GitHub. Использование метода кластеризации KMeans позволило определить пороговые значения, а применение случайного леса и глубокого леса обеспечило высокую точность предсказаний. Внедрение модели глубокого леса позволило более точно выявлять сложные зависимости в данных и автоматически формировать корректирующие рекомендации.

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Анализ данных из репозитория

Для анализа данных использовался репозиторий на GitHub. Данные включали историю коммитов, информацию об авторах, количество добавленных и удалённых строк кода, изменения в файлах и временные интервалы между коммитами. На основе этих данных разработана система автоматического извлечения информации, её анализа и формирования рекомендаций по корректирующим и предупреждающим действиям (CAPA).

```
# Чтение конфигурации из файла
config = {}
with open('config.txt', 'r') as file:
for line in file:
name, value = line.strip().split('=')
config[name] = value

access_token = config['access_token']
repos = config['repos'].split(',')

# Аутентификация с использованием токена доступа
g = Github(access_token)

# Создание файла для записи данных
with open('repository_data.csv', mode='w', newline='',
   encoding='utf-8') as file:
writer = csv.writer(file)
writer.writerow(['Repo', 'Commit SHA', 'Author', 'Date', '
   Message', 'Additions', 'Deletions', 'Total Changes', 'File
   Changes', 'Time Since Last Commit', 'Has CAPA'])

for repo_name in repos:
repo = g.get_repo(repo_name)
commits = repo.get_commits()

previous_commit_date = None
for commit in commits:
sha = commit.sha
author = commit.commit.author.name
date = commit.commit.author.date
message = commit.commit.message
stats = commit.stats
```

```
    additions = stats.additions
    deletions = stats.deletions
    total_changes = stats.total
    files_changed = commit.files

    if previous_commit_date:
    time_since_last_commit = abs((date - previous_commit_date).
        days)
    else:
    time_since_last_commit = 0

    previous_commit_date = date
    has_capa = 0  # Изначально все коммиты не CAPA
    writer.writerow([repo_name, sha, author, date, message,
        additions, deletions, total_changes, len(files_changed),
        time_since_last_commit, has_capa])

print("Данные успешно сохранены в repository_data.csv")
```

3.2. Методы обработки данных и обучение моделей

Система реализована на языке Python и включает следующие этапы:

- Извлечение данных о коммитах из репозитория GitHub.
- Запись данных в CSV-файл для последующего анализа.
- Обработка данных и определение пороговых значений с использованием алгоритма кластеризации KMeans.
- Обучение модели для определения CAPA с использованием алгоритмов машинного обучения (случайный лес и наивный байесовский классификатор).
- Генерация рекомендаций CAPA на основе анализа данных.
- Запись рекомендаций в файл и автоматическое создание pull request на GitHub.
- Визуализация данных с использованием интерактивного дашборда на основе библиотеки Dash.

Для выявления аномалий использовался метод кластеризации KMeans, который позволил установить пороговые значения на основе характеристик коммитов (количество добавленных и удалённых строк, изменения в файлах, временные интервалы между коммитами и другие параметры).

Далее проводилось обучение моделей машинного обучения, включая случайный лес и наивный байесовский классификатор. Экспериментальные результаты по-

казали, что случайный лес обеспечил более высокую точность классификации (1.0), по сравнению с наивным байесовским классификатором (0.8), что связано с возможными сложными зависимостями между признаками и наличием выбросов в данных.

```
# Загрузка данных из файла
data = pd.read_csv('repository_data.csv', encoding='utf-8')

# Использование KMeans для определения пороговых значений
features = ['Additions', 'Deletions', 'Total Changes', 'File
    Changes', 'Time Since Last Commit']
kmeans = KMeans(n_clusters=2, random_state=42)
data['Cluster'] = kmeans.fit_predict(data[features])

# Вычисление пороговых значений на основе кластеров
cluster_centers = kmeans.cluster_centers_
ADDITION_THRESHOLD = cluster_centers[:, 0].mean()
DELETION_THRESHOLD = cluster_centers[:, 1].mean()
TOTAL_CHANGE_THRESHOLD = cluster_centers[:, 2].mean()
FILE_CHANGE_THRESHOLD = cluster_centers[:, 3].mean()
TIME_SINCE_LAST_COMMIT_THRESHOLD = cluster_centers[:, 4].mean
    ()
```

3.3. Интеграция модели глубокого леса

Для дальнейшего улучшения точности была внедрена модель глубокого леса (Deep Forest). Так как библиотека deep-forest несовместима с Python 3.12, была создана виртуальная среда на Python 3.9 для выполнения модели. Основная программа вызывала исполняемый файл в этой среде, передавала данные из репозитория и получала обработанные результаты.

Глубокий лес был обучен на тех же данных, что и предыдущие модели, но позволил более точно выявлять сложные зависимости между параметрами коммитов. Итоговые предсказания загружались обратно в основной процесс для последующего анализа.

```
# Сохранение важностей признаков в CSV
importance_df = pd.DataFrame({'Feature': features, 'Importance
    ': rf_model.feature_importances_})
importance_df.to_csv('feature_importances.csv', index=False)

input_csv = 'repository_data.csv'
output_csv = 'repository_data_with_predictions.csv'
```

```
python_venv = os.path.join(os.getcwd(), 'myenv', 'Scripts', '
    python.exe')
if not os.path.exists(python_venv):
print(f"Virtual environment not found at {python_venv}")
sys.exit(1)

result = subprocess.run([python_venv, 'deep_forest_task.py',
    input_csv, output_csv], capture_output=True, text=True)
print(result.stdout)
print(result.stderr, file=sys.stderr)

if result.returncode == 0:
data_with_predictions = pd.read_csv(output_csv)
print("Результаты предсказаний загружены успешно.")
else:
print("Ошибка при выполнении подзадачи.")
```

3.4. Генерация рекомендаций CAPA

На основе адаптивных пороговых значений для каждого коммита формировались рекомендации CAPA. Если одна из метрик превышала вычисленные пороговые значения, создавалась рекомендация, которая добавлялась в итоговый список. Пример рекомендаций:

- Если количество добавленных строк превышает порог, рекомендуется провести ревизию кода.
- Если количество удалённых строк значительно увеличилось, это может указывать на устранение проблем или рефакторинг кода.
- Если временной интервал между коммитами велик, это может свидетельствовать о задержке в разработке, требующей пересмотра приоритетов.

```
# Функция генерации рекомендаций CAPA
def generate_capa_recommendations(data, ADDITION_THRESHOLD,
    DELETION_THRESHOLD, TOTAL_CHANGE_THRESHOLD,
    FILE_CHANGE_THRESHOLD, TIME_SINCE_LAST_COMMIT_THRESHOLD):
recommendations = []
for index, commit_data in data.iterrows():
suggestion = []
if commit_data['Additions'] > ADDITION_THRESHOLD:
suggestion.append(' | Review large additions |')
if commit_data['Deletions'] > DELETION_THRESHOLD:
```

```
suggestion.append(' | Review large deletions (Many deletions may indicate a fix) |')
if commit_data['Total Changes'] > TOTAL_CHANGE_THRESHOLD:
suggestion.append(' | Review total changes for potential issues (Too many changes since last commit) |')
if commit_data['File Changes'] > FILE_CHANGE_THRESHOLD:
suggestion.append(' | Review changes in multiple files (Many files updated, expected compatibility issues) |')
if abs(commit_data['Time Since Last Commit']) > abs(TIME_SINCE_LAST_COMMIT_THRESHOLD):
suggestion.append('| Review large time gap between commits (Large time gap between commits may indicate a slow development process) |')
if len(commit_data['Message']) < 15 and commit_data['Total Changes'] > TOTAL_CHANGE_THRESHOLD:
suggestion.append('| Commit message is too short, consider providing more details |')
if not suggestion:
suggestion.append('No specific recommendations')

recommendations.append({
  'Repo': commit_data['Repo'],
  'Commit SHA': commit_data['Commit SHA'],
  'Message': commit_data['Message'],
  'Suggestion': '; '.join(suggestion)
})

return recommendations

# Выявление аномалий и генерация рекомендаций
recommendations = generate_capa_recommendations(data_with_predictions, ADDITION_THRESHOLD, DELETION_THRESHOLD, TOTAL_CHANGE_THRESHOLD, FILE_CHANGE_THRESHOLD, TIME_SINCE_LAST_COMMIT_THRESHOLD)
recommendations_df = pd.DataFrame(recommendations)
```

3.5. Разработка интерактивного дашборда

Для визуализации результатов анализа был разработан дашборд на основе библиотеки Dash. Дашборд содержит:

– Графики активности разработчиков.

– Динамику изменений кода.

– Выявленные аномалии и рекомендации CAPA.
– Историю коммитов с классификацией по типам изменений.

Это позволило наглядно представлять анализируемые данные и упрощать процесс принятия решений по улучшению качества кода.

3.6. Выводы

В данной главе рассмотрена реализация системы автоматического анализа коммитов. Использование методов машинного обучения и кластеризации позволило выявлять аномалии в процессе разработки и формировать рекомендации по корректирующим действиям.

Интеграция модели глубокого леса позволила повысить точность классификации, а интерактивный дашборд упростил процесс мониторинга состояния репозитория. Таким образом, предложенная система способствует повышению качества кода и автоматизации процесса контроля за разработкой программного обеспечения.

ГЛАВА 4. АПРОБАЦИЯ РЕЗУЛЬТАТОВ ИССЛЕДОВАНИЯ

4.1. Цель апробации

Апробация разработанного метода включает тестирование модели на реальных данных, оценку её эффективности и сравнение с альтернативными подходами. Основной целью является определение точности, устойчивости и применимости модели в различных сценариях.

4.2. Методика тестирования

Для проверки работоспособности модели были проведены следующие этапы тестирования:

– Оценка качества классификации коммитов с использованием метрик (точность);
– Анализ временной эффективности работы алгоритма на выборке разного объема;
– Сравнение результатов работы модели с альтернативными методами (например, случайный лес);

4.3. Практическое применение и апробация

Результаты апробации были проверены на нескольких репозиториях с различными паттернами разработки. Для этого были выбраны проекты с:

– Высокой частотой коммитов (активно разрабатываемые проекты);
– Длинными промежутками между коммитами (поддерживаемые проекты);
– Большим количеством изменений в кодовой базе.

Анализ показал, что метод корректно адаптируется к различным сценариям, предоставляя полезные рекомендации по улучшению процессов разработки.

4.4. Выводы

Результаты апробации показали, что предложенный метод анализа коммитов позволяет эффективно выявлять потенциальные проблемы в процессе разработки, формируя полезные рекомендации CAPA. Сравнительный анализ подтвердил конкурентоспособность модели относительно существующих методов, а тестирование на реальных данных продемонстрировало её практическую применимость.

ЗАКЛЮЧЕНИЕ

В ходе данной работы была разработана система автоматического анализа коммитов, направленная на выявление аномалий и формирование корректирующих и предупреждающих действий (CAPA). Использование методов машинного обучения и кластеризации позволило создать инструмент, способный анализировать историю изменений в коде и предлагать рекомендации для повышения качества программного обеспечения.

Основные результаты работы можно сформулировать следующим образом:

- Проведен обзор существующих методов анализа данных из репозиториев исходного кода и выявлены их ограничения.
- Разработан алгоритм автоматического извлечения данных о коммитах с последующей их обработкой и анализом.
- Предложен метод кластеризации коммитов с использованием алгоритма KMeans для определения пороговых значений изменений в коде.
- Обучены и протестированы модели машинного обучения (случайный лес, наивный байесовский классификатор и глубокий лес), показавшие высокую точность в задаче предсказания аномалий.
- Разработан механизм автоматического создания pull request с рекомендациями CAPA, который интегрируется в процесс разработки.
- Создан интерактивный дашборд для визуализации результатов анализа, что позволяет разработчикам легко отслеживать состояние репозитория и принимать решения на основе данных.

Практическая значимость предложенной системы заключается в том, что она позволяет автоматизировать контроль за качеством кода, минимизировать ошибки, возникающие в процессе разработки, и повысить прозрачность изменений в репозитории. Используемый подход может быть адаптирован для различных проектов и масштабируем для работы с крупными кодовыми базами.

В дальнейшем возможны следующие направления развития системы:

- Доработка алгоритмов выявления аномалий с учетом более сложных паттернов изменений в коде.
- Расширение набора метрик для анализа коммитов.
- Интеграция с другими инструментами контроля качества кода и CI/CD системами.

– Применение нейросетевых моделей для улучшения предсказательной способности системы.

Таким образом, проведенное исследование подтвердило эффективность предложенного подхода к анализу коммитов. Разработанная система способствует улучшению управления процессом разработки программного обеспечения, сокращает время на выявление потенциальных проблем и повышает качество выпускаемого кода.

СЛОВАРЬ ТЕРМИНОВ

CAPA (Corrective and Preventive Actions) — корректирующие и предупреждающие действия, направленные на устранение и предотвращение дефектов в процессе разработки программного обеспечения.

GitHub — веб-сервис для хостинга IT-проектов и их совместной разработки на базе системы управления версиями Git.

Коммит (commit) — фиксация изменений в репозитории Git, включающая информацию о внесённых правках, авторе и времени изменения.

KMeans — метод кластеризации данных, основанный на разбиении множества на k групп по схожести признаков.

Случайный лес (Random Forest) — ансамблевый метод машинного обучения, использующий множество деревьев решений для повышения точности прогнозов.

Наивный Байесовский классификатор — алгоритм машинного обучения, основанный на теореме Байеса и предположении независимости признаков.

Глубокий лес (Deep Forest) — метод машинного обучения, использующий каскадную структуру случайных лесов для улучшения классификации.

API (Application Programming Interface) — интерфейс программирования приложений, позволяющий взаимодействовать с внешними сервисами и библиотеками.

Pull Request (PR) — запрос на внесение изменений в репозиторий GitHub, который проходит процесс ревью перед слиянием в основную ветку.

Dash — фреймворк на Python для создания интерактивных дашбордов и веб-приложений для визуализации данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[1] Аваис М., Гу В., Дламини Г., Холматова З., Суччи Дж. An experience in automatically extracting CAPAs from code repositories // arXiv.org. 2022. URL: https://arxiv.org/pdf/2212.09910

[2] Bugayenko Y., Daniakin K., Farina M., Jolha F., и др. Extracting corrective actions from code repositories // В сб.: Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022). ACM, 2022. DOI: https://dl.acm.org/doi/abs/10.1145/3524842.3528517

[3] Холматова З., Корбашов В., Педрич В., Суччи Д. A meta-analytical comparison of Naive Bayes and Random Forest for software defect prediction // ResearchGate. 2021. URL: https://www.researchgate.net/publication/350459831_A_meta-analytical_comparison_of_Naive_Bayes_and_Random_Forest_for_software_defect_prediction

[4] Examining the Success of an Open Source Software Project Analysing Its Repository // Zenodo. 2025. DOI: https://doi.org/10.5281/zenodo.10046579

[5] Di Bella E., Tamburri D.A., Serebrenik A., Storey M.-A., Melegati J., Ferreira M. GitHub Projects: Quality Analysis of Open-Source Software // В сб.: Proceedings of the 10th International Conference on Open Source Systems. Cham: Springer, 2014. С. 159–169. URL: https://link.springer.com/chapter/10.1007/978-3-319-13734-6_6

[6] Utkin L. V. An imprecise deep forest for classification //Expert Systems with Applications. – 2020. – Т. 141. – С. 112978. URL: https://www.sciencedirect.com/science/article/pii/S0957417419306967

[7] Jain A. K., Murty M. N., Flynn P. J. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation // Electronics. 2020. Т. 9, № 8. DOI: https://www.mdpi.com/2079-9292/9/8/1295

[8] Pícha P. Detecting software development process patterns in project data // В кн.: Proceedings of the 23rd International Conference on Soft Computing MENDEL 2019. Brno: Springer, 2019. URL: https://otik.uk.zcu.cz/handle/11025/37196

[9] Github API documentation. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28

[10] PyGithub documentation. URL: https://pygithub.readthedocs.io/en/stable/