ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ
INFORMATION SYSTEM AND TECHNOLOGIES

УДК 004.05 **DOI: 10.18413/2518-1092-2023-8-3-0-1**

Кузьминых Е.С.
Ильина С.П.
Маслова М.А.

АНАЛИЗ СХОДСТВА КОДА И ПОИСКА ЕГО ЗАИМСТВОВАНИЙ

Севастопольский государственный университет, ул. Университетская, д. 33, г. Севастополь, 299053, Россия

e-mail: egor2014ru@mail.ru, sofi.ilina@mail.ru, mashechka-81@mail.ru

Аннотация
В настоящей статье рассматривается актуальная проблема в сфере программирования – заимствование кода и возможности его анализа. С увеличением числа программистов и обилием программных решений, появляется необходимость определения зависимостей и уникальности используемого кода. Вместе с возрастающим распространением кода через библиотеки и открытые источники, становится актуальным выявление случаев плагиата или незаконного использования. Данная статья исследует эффективные методы и инструменты для поиска заимствованного кода. Через представление существующих алгоритмов и подходов, авторы статьи предлагают оценку эффективности существующих решений и представляют примеры алгоритмов, способствующих анализу кода на схожесть.
Ключевые слова: информационная безопасность; ИБ; безопасность; кибербезопасность; код; программирование; анализ кода; поиск заимствований; заимствование кода; оценка сходства кода; дубликаты кода; машинное обучение; сортировка
Для цитирования: Кузьминых Е.С., Ильина С.П., Маслова М.А. Анализ сходства кода и поиска его заимствований // Научный результат. Информационные технологии. – Т.8, №3, 2023. – С. 3-10. DOI: 10.18413/2518-1092-2023-8-3-0-1

Kuzminykh E.S.
Ilina S.P.
Maslova M.A.

ANALYSIS CODE SIMILARITY AND SEARCH FOR ITS BORROWINGS

Sevastopol state University, 33 Universitetskaya St., Sevastopol, 299053, Russia

e-mail: egor2014ru@mail.ru, sofi.ilina@mail.ru, mashechka-81@mail.ru

Abstract
This article discusses an actual problem in the field of programming — the borrowing of code and the possibility of its analysis. With the increase in the number of programmers and the abundance of software solutions, there is a need to determine the dependencies and uniqueness of the code used. Along with the increasing distribution of code through libraries and open sources, it becomes relevant to identify cases of plagiarism or illegal use. This article explores effective methods and tools for finding borrowed code. Through the presentation of existing algorithms and approaches, the authors of the article offer an assessment of the effectiveness of existing solutions and provide examples of algorithms that contribute to the analysis of the code for similarity.
Keywords: information security; information security; security; cybersecurity; code; programming; code analysis; search for borrowings; code borrowing; code similarity assessment; code duplicates; machine learning; sorting

For citation: Kuzminykh E.S., Ilina S.P., Maslova M.A. Analysis code similarity and search for its borrowings // Research result. Information technologies. – T.8, №3, 2023. – P. 3-10. DOI: 10.18413/2518-1092-2023-8-3-0-1

ВВЕДЕНИЕ

В век развития информационных технологий появляется всё больше программистов, которые пишут свои уникальные программы или улучшают уже созданные, что приводит к огромному количеству созданных программ. Существует единая библиотека для любителей поделиться своими работами с другими. Такое распространение кода приводит к его воровству с небольшой доработкой или без. Известно, что существенная часть кодовой базы продуктов является наследованной от различных библиотек, фреймворков и пакетов. При анализе кода хочется уделять внимание уникальному коду и определять источники зависимостей. Для этого требуется реализовать эффективный алгоритм поиска заимствованного кода.

ОСНОВНАЯ ЧАСТЬ

В основном рассматриваются 4 типа клонов кода:

Полные дубликаты — это копии исходного кода, которые полностью повторяют друг друга. Такой тип клонирования возникает при копировании и вставке кода без каких-либо изменений.

Функциональные дубликаты — это копии исходного кода, которые выполняют ту же функцию, что и оригинал, но с некоторыми изменениями в структуре, например, с использованием других имен переменных или других алгоритмов решения задачи.

Ключевые фрагменты — это небольшие участки кода, которые используются в разных местах программы. Это может быть как полная строка кода, так и отдельный оператор или функция.

Структурные дубликаты — это копии исходного кода, которые имеют похожую структуру, но различаются по содержанию. Например, две функции могут иметь одинаковую структуру, но различаться в том, какие данные они обрабатывают и какие значения возвращают.

Существует несколько подходов к оценке сходства участков кода:

Метрический подход: используются метрики для измерения сходства, такие как расстояние Левенштейна и коэффициент Жаккара.

Для поиска необходимо воспользоваться следующими пунктами:

1) Дизассемблирование исполняемого файла с использованием IDA Pro;
2) Деление кода ассемблера на перекрывающиеся куски кода, состоящие из дальнейших действий функции;
3) Упорядочение ряда команд при помощи абстракции информации о ячейках памяти и регистрах;
4) Подсчет метрик для нормализованных последовательностей;
5) Дальнейшее сравнение метрик, с результатом – получением групп клонов.

Синтаксический подход: основан на анализе структуры и синтаксисе кода. В этом подходе сначала происходит разбор кода для получения его абстрактного синтаксического дерева (AST), а затем происходит сравнение AST-деревьев.

Семантический подход: основан на анализе смысла кода. В этом подходе используется анализ контекста и вычисления зависимостей между переменными и функциями в коде.

Гибридный подход: комбинирует два или более подхода для достижения лучших результатов [2, 3].

Каждый из этих подходов имеет свои преимущества и недостатки, и выбор определенного подхода зависит от конкретной задачи.

Рассматриваемые алгоритмы могут использовать различные свойства и признаки для решения задач.

Например, в алгоритмах машинного обучения для классификации объектов часто используются следующие свойства и признаки:

– числовые признаки (например, размер, вес, цена);
– категориальные признаки (например, цвет, форма, материал);
– текстовые данные (например, название товара, описание продукта);
– изображения и видео (например, фотографии или видеозаписи объектов).

В зависимости от задачи и типа данных, которые необходимо обработать, могут использоваться и другие свойства, и признаки. Например, для анализа временных рядов будут использоваться временные признаки, а для голосового управления — звуковые характеристики голоса пользователя.

Существует несколько методов преобразования категориальных признаков:

– кодирование меток — способ замены категориальных признаков на числовые значения. Любое свойство категориального признака заменяется на подходящее ему число. Данный способ подходит для признаков с несколькими значениями, например, уровень образования;

– однократное кодирование — способ замены категориальных признаков на бинарные значения. Каждая переменная категориального признака преобразуется в столбец бинарных значений, где 1 указывает на присутствие значения признака, а 0 – на его отсутствие. Данный способ подходит для признаков без порядка значений, таких как цвет либо тип объекта;

– двоичное кодирование — способ замены категориальных признаков в бинарные значения методом кодирования Грея. В данном методе каждое значение категориального признака заменяется на бинарное значение. Далее каждый последующий столбец имеет величину, отличную на одно значение от предшествующей. Данный метод можно использовать и для признаков без порядка значений.

Числовыми признаками считаются: масштабирование, нормализация, дискретизация, стандартизация [4].

Проанализированные подходы могут столкнуться со следующими проблемами:

– недостаточное количество данных — это может привести к низкой точности моделей и переобучению. Решение: сбор большего количества данных или использование техник аугментации данных;

– несбалансированные классы — это может привести к низкой точности моделей для меньшего класса объектов. Решение: использование взвешивания классов, увеличение обучающих данных для меньшего класса;

– переобучение — это происходит, когда модель слишком сложная для задачи или когда она обучается на выбросах. Решение: использование регуляризации, выбор упрощенной модели, использование случайных выборок (dropout) [5].

– недообучение — это происходит, когда модель слишком простая для задачи или когда она не имеет достаточно данных для обучения. Решение: использование более сложной модели, использование большего количества данных;

– скрытые переменные — это может быть вызвано отсутствием информации о состоянии системы. Решение: использование вероятностных моделей или методов инверсии;

– необходимость интерпретации результатов — это может быть вызвано необходимостью объяснения принятого решения или причины ошибки. Решение: использование методов интерпретации, таких как анализ важности функций, сводные таблицы и графические представления.

Решение каждой из этих проблем зависит от конкретной задачи и используемых данных. Важно подобрать наиболее подходящие методы для решения каждой из проблем.

Одним из главных преимуществ методов, использующих машинное обучение (ML), является автоматическое извлечение закономерностей из больших объемов данных. Это позволяет решать задачи, которые трудно или невозможно решить с помощью традиционных методов.

Например, ML-алгоритмы могут использоваться для построения моделей, которые предсказывают будущие события на основе прошлых данных. Это может применяться в различных отраслях, таких как финансы, медицина, производство и т.д.

Еще одним преимуществом методов, использующих машинное обучение, является возможность создания универсальных моделей, которые могут быть применены к различным задачам. Например, нейронные сети могут быть обучены распознавать образы, а также делать предсказания на основе временных рядов.

Кроме того, ML-алгоритмы могут автоматически адаптироваться к изменениям в данных, что делает их более гибкими и универсальными. Они также могут работать с данными, которые трудно интерпретировать или имеют сложную структуру.

Наконец, использование ML-методов может помочь снизить затраты на ручную обработку и анализ данных, что может быть особенно полезно в случае больших объемов информации.

Для проверки работы исследуемых алгоритмов могут использоваться различные тестовые данные в зависимости от конкретной задачи. Например, для задач классификации можно использовать наборы данных, содержащие изображения или тексты с определенными метками классов. Для задач регрессии — наборы данных со значениями целевых переменных.

Кроме того, для оценки качества работы алгоритмов могут использоваться различные метрики, такие как точность (precision), полнота (recall), F-мера и другие, которые могут быть выбраны в зависимости от конкретной задачи.

Также можно использовать кросс-валидацию, чтобы оценить обобщающую способность модели на новых данных. При кросс-валидации данные делятся на несколько частей, и каждый раз обучение и тестирование проводятся на разных подмножествах данных.

Кросс-валидация — используют для оценки качества работы модели. Он обширно применяется в машинном обучении, дает возможность сравнивать между собой различные модели и определять наилучшую для конкретной задачи [6].

В любом случае, для выбора тестовых данных следует принимать во внимание особенности конкретной задачи и возможные источники ошибок при работе алгоритмов на этих данных.

Точность — это доля ответов, которые алгоритм верно классифицировал относительно общего количества ответов, которые он предсказал. То есть, если алгоритм предсказал 10 объектов, из которых 8 оказались верными, то его точность составит 80%.

Полнота — это доля верно предсказанных ответов от общего числа правильных ответов. Если, например, всего было 20 правильных ответов, но алгоритм угадал лишь 16, то его полнота будет равна 80%.

Применяется несколько способов объединения precision и recall в агрегированный критерий качества. F-мера (в общем случае F_β) — среднее гармоническое precision и recall (1):

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \quad (1)$$

где

β – определяет вес точности в метрике, и при $\beta = 1$ – это среднее гармоническое (с множителем 2, чтобы в случае precision = 1 и recall = 1 иметь $F_1 = 1$);

F – может достигнуть максимума при полноте и точности, равными единице, также близка к нулю, когда один из аргументов близок к нулю [7].

Оценка сходства кода может быть использована для решения следующих прикладных задач информационной безопасности:

– обнаружение вредоносного кода: эффективный алгоритм оценки сходства кода может позволить обнаружить вредоносный код, который был изменен или скрыт за шифрованием или другими методами маскировки;

– анализ утечек данных: алгоритм оценки сходства кода может помочь выявить случаи кражи информации и утечки конфиденциальных данных. Также алгоритм может показать, какие файлы были скопированы или перемещены на другой носитель;

– выявление уязвимостей: алгоритм оценки сходства кода может использоваться для анализа кода на наличие уязвимостей. Например, он может выявить похожий уязвимый код, который уже был использован в атаках или которые может стать целью будущих атак;

– проверка подлинности: алгоритм оценки сходства кода может использоваться для проверки подлинности программного обеспечения. Например, он может помочь определить, является ли программа настоящей или фальсифицированной;

– анализ соблюдения правил безопасности: алгоритм оценки сходства кода может помочь анализировать соблюдение правил безопасности и стандартов в программном обеспечении. Например, он может выявить нарушение политик безопасности, таких как использование небезопасных функций или открытие портов без разрешения [8].

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ И ИХ ОБСУЖДЕНИЕ

Далее продемонстрирована работа нескольких базовых алгоритмов и показаны их недостатки.

Листинг 1 — Алгоритм SequenceMatcher [9]

```
      import difflib

      def compare_code(file1, file2):
          with open(file1, 'r') as f1, open(file2, 'r') as f2:
              code1 = f1.readlines()
              code2 = f2.readlines()

          matcher = difflib.SequenceMatcher(None, code1, code2)
          similarity_ratio = matcher.ratio()

          return similarity_ratio

      file1 = "file1.txt"
      file2 = "file2.txt"

      similarity = compare_code(file1, file2)
print(f"Similarity ratio: {similarity}")
```

Метод SequenceMatcher — это класс пакета difflib Python, который предоставляет функции и классы для сравнения последовательностей. Используется для сравнения различных файлов на схожесть.

В этом примере программа открывает и считывает содержимое файлов «file1.txt» и «file2.txt». Затем создает экземпляр класса SequenceMatcher, передавая в него строки кода из обоих файлов. Метод ratio() возвращает отношение схожести между этими строками, где 0 означает полное отсутствие сходства, а 1 — полное совпадение. Программа выводит коэффициент сходства между двумя файлами. Чем ближе значение к 1, тем более похожи коды в файлах. Однако, следует учесть, что такой алгоритм будет работать только на уровне строк и не учитывает семантику кода. Если требуется более точный анализ сходства кода, то могут потребоваться более сложные методы и библиотеки, такие как алгоритмы на основе графового анализа или машинное обучение.

Следующий алгоритм похож на первый, но чуть сложнее, хотя использует ту же самую библиотеку difflib.

Листинг 2 — Вторая версия difflib

```
import difflib

def compare_code(file1, file2):
    with open(file1, 'r') as f1, open(file2, 'r') as f2:
        code1 = f1.read()
        code2 = f2.read()

    matcher = difflib.SequenceMatcher(None, code1, code2)
    blocks = matcher.get_matching_blocks()

    similarities = []
    for block in blocks:
        if block.size > 0:
            similarity = code1[block.a:block.a+block.size]
            similarities.append(similarity.strip())

    return similarities

file1 = "file1.txt"
file2 = "file2.txt"

similar_lines = compare_code(file1, file2)
if similar_lines:
    print("Similar lines found:")
    for line in similar_lines:
        print(line)
else:
print("No similar lines found.")
```

В этом примере программа также открывает и считывает содержимое файлов «file1.txt» и «file2.txt». Затем создает экземпляр класса SequenceMatcher, чтобы получить сравнение между строками кода. Функция get_matching_blocks() возвращает список обнаруженных блоков, включающих совпадения и различия между файлами. Программа проходится по каждому блоку и, если размер блока больше 0, добавляет схожую строку в список similarities. Затем выводит найденные схожие строки. Этот пример программы позволяет найти не только коэффициент сходства между файлами кода, но и конкретные строки, которые совпадают или похожи друг на друга.

Сравнив эти 2 алгоритма, можно сделать вывод, что для базовой проверки кода на схожесть достаточно программы из 20 строк, которая сможет выявить явную схожесть кода, но для более глубокой проверки необходимо учитывать семантику кода и разрабатывать более сложные алгоритмы проверки [10].

ЗАКЛЮЧЕНИЕ

В заключении можно подчеркнуть важность анализа сходства кода и поиска его заимствований в современной разработке программного обеспечения. Этот метод помогает обнаружить повторяющиеся фрагменты кода и выявить возможные проблемы с плагиатом или небезопасными практиками программирования. Анализ сходства кода предоставляет разработчикам возможность оптимизировать процесс разработки, повысить эффективность и надежность кода. Поиск заимствований кода помогает командам разработчиков обнаружить повторное использование кода из различных источников, включая открытые источники, легаси-код или даже внутри компании. Это позволяет оптимизировать использование ресурсов и избежать проблем, связанных с авторскими правами или патентами.

Современные инструменты анализа сходства кода предлагают широкий набор функций, позволяющих проводить глубокий и точный анализ кода, учитывая его синтаксис, структуру и

логику. Такие инструменты помогают разработчикам быстро обнаруживать дублированный код, проводить рефакторинг и бороться с потенциальными уязвимостями. В итоге, осознанный подход к анализу сходства кода и поиску его заимствований является неотъемлемой частью современной разработки ПО. Это помогает повысить качество и надежность кода, оптимизировать его использование и обеспечить соблюдение авторских прав [11].

Список литературы

1. Бабкина А.А. Инструментальная поддержка поиска клонов в программном коде: Магистерская диссертация – Сибирский федеральный университет, 2017.
2. Саргсян С.С. Методы поиска клонов кода и семантических ошибок на основе семантического анализа программы: дис. канд. физико-математических Наук – Ин-т систем. программирования, 2016.
3. Выявление клонов [Электронный ресурс]. URL: https://newtechaudit.ru/clone-code/
4. Обработка данных для машинного обучения [Электронный ресурс]. URL: https://4brain.ru/aibasics/data.php
5. Какие бывают типы ошибок в моделях машинного обучения? [Электронный ресурс]. URL: https://qaa-engineer.ru/kakie-byvayut-tipy-oshibok-v-modelyah-mashinnogo-obucheniya/
6. Кросс-валидация [Электронный ресурс]. URL: https://academy.yandex.ru/handbook/ml/article/kross-validaciya
7. Чутка ликбеза: что такое f мера в машинном обучении? [Электронный ресурс]. URL: https://dzen.ru/a/ZIHJQbB6HGcQVC6i
8. Ищем уязвимости в коде: теория, практика и перспективы SAST [Электронный ресурс]. URL: https://www.securitylab.ru/analytics/483063.php
9. Алгоритм SequenceMatcher [Электронный ресурс]. URL: https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher
10. Вовченко Н.Г., Кузнецов Н.Г., Макаренко Е.Н. и др. Реализация ESG-принципов в стратегии устойчивого развития экономики России. – Ростов-на-Дону: Ростовский государственный экономический университет "РИНХ", 2022. – 508 с.
11. Нестеренко В.Р., Маслова М.А. Современные вызовы и угрозы информационной безопасности публичных облачных решений и способы работы с ними // Научный результат. Информационные технологии. – 2021. – Т. 6, № 1. – С. 48-54.

References

1. Babkina A.A. Instrumental support for the search for clones in the program code: Master's thesis – Siberian Federal University, 2017.
2. Sargsyan S.S. Methods of searching for code clones and semantic errors based on semantic analysis of the program: dis. Candidate of Physical and Mathematical Sciences – In-t Systems. programming, 2016.
3. Identification of clones [Electronic resource]. URL: https://newtechaudit.ru/clone-code/
4. Data processing for machine learning [Electronic resource]. URL: https://4brain.ru/aibasics/data.php
5. What are the types of errors in machine learning models? [electronic resource]. URL: https://qaa-engineer.ru/kakie-byvayut-tipy-oshibok-v-modelyah-mashinnogo-obucheniya/
6. Cross-validation [Electronic resource]. URL: https://academy.yandex.ru/handbook/ml/article/kross-validaciya
7. A little educational program: what is an f measure in machine learning? [electronic resource]. URL: https://dzen.ru/a/ZIHJQbB6HGcQVC6i
8. Looking for vulnerabilities in the code: theory, practice and prospects of SAST [Electronic resource]. URL: https://www.securitylab.ru/analytics/483063.php
9. Algoritm SequenceMatcher [Elektronnyj resurs]. URL: https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher

10. Vovchenko N.G., Kuznetsov N.G., Makarenko E.N. and others. Implementation of ESG principles in the strategy for sustainable development of the Russian economy. – Rostov-on-Don: Rostov State Economic University "RINH", 2022. – 508 p.

11. Nesterenko R.V., Maslova M.A. Modern challenges and threats information security public cloud making and methods of work with them // Research result. Information technologies. – T.6, №1, 2021. – P. 48-54. DOI: 10.18413/2518-1092-2021-6-1-0-6

Кузьминых Егор Сергеевич, студент четвертого курса кафедры Информационная безопасность Института информационных технологий
Ильина София Павловна, студент четвертого курса кафедры Информационная безопасность Института информационных технологий
Маслова Мария Александровна, старший преподаватель кафедры Информационная безопасность Института информационных технологий

Kuzminykh Egor Sergeevich, fourth-year Student of the Department Information security, Institute of Information Technologies
Ilina Sofia Pavlovna, fourth-year Student of the Department Information security, Institute of Information Technologies
Maslova Maria Alexandrovna, Senior Lecturer of the Department Information security Institute of Information Technologies