М.В. Аврискин [1], М.С. Воробьева [1,2]

[1] Тюменский государственный университет, г. Тюмень

[2] Научно-технический университет «Сириус», г.Сочи

УДК 004.912

АНАЛИЗ КОЛИЧЕСТВЕННЫХ ПРИЗНАКОВ ПРОГРАММНОГО КОДА

Аннотация: В статье рассматриваются различные количественные признаки программного кода, методы и инструменты для извлечения и анализа, что позволит сравнивать работы студентов, используя различные метрики кода.

Ключевые слова: интеллектуальный анализ программного кода, машинное обучение, анализ текстовых данных, визуализация.

Введение

Для улучшения качества образования всё чаще используются различные информационные технологии, в том числе для анализа работ студентов [1].

Анализ программного кода применяется для улучшения качества кода как в промышленной разработке, так и в простых личных проектах. Автоматизированный анализ содействует аудиту программного кода и борьбе с плагиатом, а также может использоваться в образовательных целях.

В то время, как программные средства анализа не могут заменить квалифицированного преподавателя. Но наличие сервиса для сбора и анализа работ позволит пользователю быстро находить предыдущие работы программиста, сравнивать работу по определенному заданию с другими решениями, используя различные метрики, такие как длина кода,

количество методов, цикломатическая сложность, количество используемых модулей, количество комментариев, длины строк.

Особенно актуально это может быть, когда проверка работ производится несколькими преподавателями в независимых группах, а с применением автоматизированного подхода появится возможность просматривать, анализировать, оценивать и сравнивать работы студентов.

Обзор исследований и инструментов анализа

Для анализа кода чаще всего используются статический анализ кода — анализ исходного кода без компиляции и запуска.

Одним из первых продуктов для анализа кода считается Unix-утилита «lint» [2], выпущенная в 1978 году и выполняющая анализ исходного кода на языке C. В настоящее время слова lint и linter стали применяться для описания любых утилит для анализа кода на возможные критические ошибки и соблюдение стиля, а также для описания самого процесса. Эти библиотеки, а также инструменты статического анализа, встроенные в среды разработки ориентированы на использование индивидуально разработчиками. В дополнение этим инструментам на рынке появляются системы, ориентированные на крупные команды разработчиков.

При этом инструменты анализа зачастую используют правила, заданные заранее разработчиком инструментов. Такому подходу противопоставлен интеллектуальный анализ кода — поиск закономерностей и правил в больших наборах данных с использованием машинного обучения. Этот подход позволяет создавать критерии оценки без значительных усилий со стороны разработчика, однако он является менее используемым на практике подходом из-за более сложной интерпретируемости пользователем получаемых моделей.

Зачастую в научных статьях статей посвященных интеллектуальному анализу кода рассматривается возможность использования стилометрии кода для определения авторства кода и проверке на плагиат.

Методы, описанные в статьях [3–6] опираются на использование стилометрии кода и дают возможность определять авторство исходного кода, даже если автор решал абсолютно другую задачу и при попарном сравнении образцы исходного кода являются различными.

В целом структура подхода к задаче определения с использованием стилометрии является неизменной. По образцам исходного кода выделяются стилометрические признаки, а затем производится классификация типичными методами машинного обучения [7].

Методы, приведенные в статье «Ступенчатый метод проверки исходного кода программы на плагиат» Стрельчёнка Г.В. [8], подразумевают попарное сравнение образцов исходного кода и не подходят для анализа больших объемов данных в связи с вычислительной сложностью подхода. Однако, эти методы могут быть использованы для более наглядного сравнения похожих образцов кода.

Разработка приложения для анализа программного кода

Для работы с программным кодом было спроектировано и разработано веб-приложение, позволяющее наглядно просматривать работы студента и сравнивать их с другими образцами кода.

Приложение основано на Фреймворке «ASP.NET Core» с использованием технологии «Razor Pages» для генерации HTML-страниц с применением языков программирования C#, HTML, JavaScript. Для хранения данных использована СУБД MariaDB и библиотека Entity Framework Core для связи веб-приложения с СУБД.

Исходные коды студентов для обработки взяты с образовательного портала (elearning.utmn.ru) — системы электронного обучения ТюмГУ, основанной на платформе Moodle. Для графической визуализации

применяется JavaScript-библиотека Highcharts, для отображения программного кода на веб-страницах — библиотека «highlight.js».

Вся работа с приложением происходит через веб-интерфейс, студенты могут отправлять свои работы по соответствующим заданиям для проведения анализа и просмотра отчета, а преподаватель может просматривать статистику по выбранному студенту или по всей группе.

После загрузки образца программного кода текст и метаданные сохраняются в базу данных в исходном виде. Сохранение исходных данных позволяет пересчитывать при дальнейшей необходимости извлекаемые признаки, а также позволяет преподавателю просматривать исходный код.

Для выделения признаков кода был создан класс CodeFeatures, экземпляр которого создается из файла с исходным кодом. При создании экземпляра класса с помощью библиотеки Microsoft.CodeAnalysis.CSharp строится абстрактное синтаксическое дерево, которое используется для нахождения численных значений искомых признаков. Полученный вектор признаков преобразуется в формат JSON и сохраняется в базу данных для возможности быстрого анализа в дальнейшем.

В приложении можно просматривать код отдельной работы студента с подсветкой синтаксиса, отображением стилометрических признаков кода, а также сравнения признаков с признаками других студентов и других работ этого же студента.

Для примера визуализации взяты следующие признаки:

- Доля строк с комментариями и доля комментариев в коде — позволяют понять, что в работе может быть очень мало, или, наоборот слишком много комментариев.
- Доля пустых строк — позволяет заметить, если программист не ставит отступы. При программировании рекомендуется разделять логические блоки в программе с помощью пустых строк.

- Средняя длина строки — позволяет заметить, в каких из экземпляров кода могут быть слишком длинные строки. Рекомендации по оформлению кода указывают на необходимость разбиение слишком длинных строк на несколько для удобочитаемости как на бумаге, так и на экране компьютера.
- Средняя длина идентификаторов — малые значения могут указывать на чрезмерное использование однобуквенных названий переменных, в то время как рекомендуется использовать в качестве названий слова, описывающие значение переменной.
- Средняя глубина вложенности фигурных скобок — большие значения указывают на слишком большую вложенность циклов, условных операторов и других конструкций. Это приводит к большим длинам строк и низкой читабельности кода в целом. Подобные блоки кода рекомендуется выносить в отдельные функции.

При открытии страницы, соответствующей образцу кода, отображается таблица со значениями признаков при выполнении других лабораторных работ данным студентом. При выборе одной из лабораторной работ, выполненных студентом, отображаются значения признаков по этой работе по сравнению с медианой и квартилями других работ по этому заданию (см. рис. 1). На примере по таблице видно, что студент Иванов Иван, начиная с третьей работы, перестал использовать комментарии при выполнении задания, что может затруднить проверку преподавателем.

Иванов Иван

Значения признаков

Признак	Лаб 61256	Лаб 62452	Лаб 63968	Лаб 67394	Среднее
Доля строк с комментариями	0.48	1.85	0.00	0.00	5.28
Доля пустых строк	11.88	35.19	13.12	7.79	14.11
Доля комментариев	0.27	3.17	0.00	0.00	9.30
Ср. длина строки	44.95	30.08	42.51	36.42	35.03
Ср. длина идентификатора	6.15	4.41	5.46	4.81	4.37
Ср. глубина {}	1.47	1.09	1.95	1.47	1.32

Рис. 1. Отображение таблицы со значениями признаков работ студента

При выборе отдельного признака отображается график динамики изменения этого признака со временем, отображающий значения признака из различных лабораторных работ.

Очевидно, абсолютные значения могут отражать характер выполняемого задания, а не только развитие студента. Для того чтобы снизить степень влияния характера задания на результат, рекомендуется оценка значений признака по сравнению с значениями других студентов по этой же лабораторной работе.

Поэтому, в дополнение к графику абсолютных значений, в приложении вычисляется и визуализируется отклонение значения от среднего по учебной группе.

Также отображается гистограмма распределения признака, взятого среди всех студентов группы, и значение признака текущего студента. Это позволяет оценить стиль кода группы в целом и то, насколько стиль студента отклоняется от среднего.

Например, на рисунке 2 можно видеть, что у данного студента в 5 из 6 работ длина строк ниже среднего по группе, а значит, что несоответствие правилам оформления, касающихся длины строк, маловероятно. При этом по гистограмме видно, что при оформлении кода работ у некоторых студентов группы использовались строки с длинами, значительно превышающие стандартные значения.

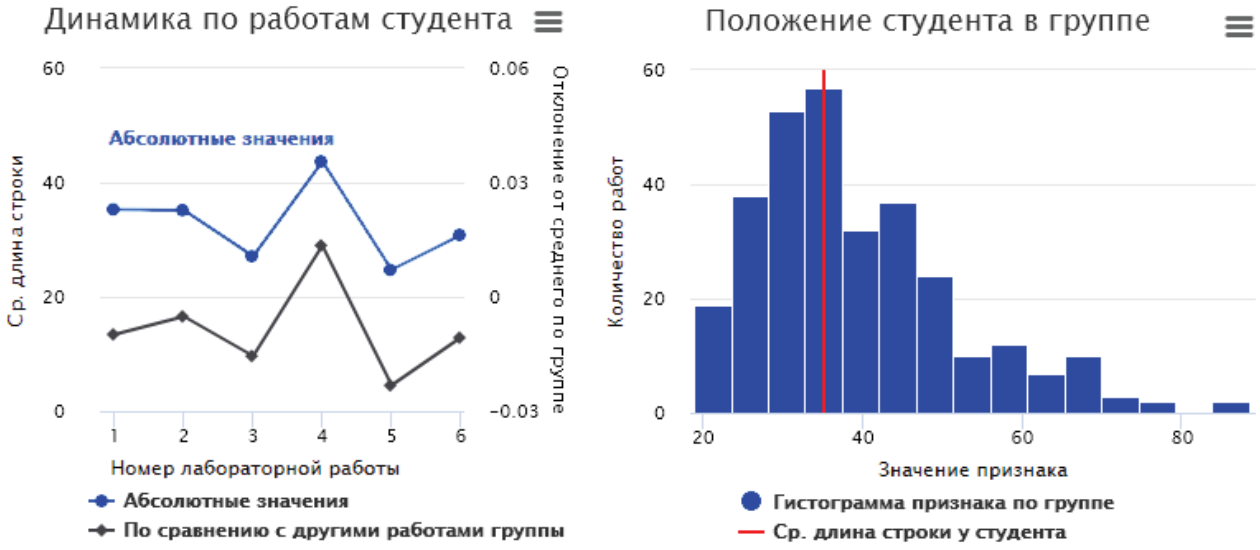

Рис. 2. Отображение значения признака «Средняя длина строки» для студента

Эти графики могут также помочь проиллюстрировать студенту особенности его кода. Например, студент может увидеть, что в своих работах он использовал чрезмерно длинные строки в коде, что может являться нарушением рекомендаций оформления кода [9].

Заключение

В результате проведенного исследования были изучены инструменты для статического анализа программного кода. Рассмотрены статьи, посвященные анализу программного кода с использованием стилометрии кода.

Разработанное веб-приложение для анализа программных кодов было протестировано на примере более 300 работ, написанных на языках C++ и C# студентами 1 курса направлений «Компьютерная безопасность» и «Математическое обеспечение и администрирование информационных систем» Института математики и компьютерных наук ТюмГУ.

Благодарности

Статья подготовлена в рамках разработки образовательного кейса для НТУ Сириус при финансовой поддержке РФФИ в рамках научного проекта № 19-37-51028.

СПИСОК ЛИТЕРАТУРЫ

1. Воробьева, М.С., Павлова Е.А. Разработка конструктора вариантов индивидуальных заданий на основе шаблонов / М. С. Воробьева, Е.А. Павлова // Современные исследования социальных проблем. – 2017. №8. – С. 214-217.
2. S. C. Johnson. Lint, a C Program. Bell Laboratories, Murray Hill. 1978.
3. Куртукова А.В., Романов А.С. Идентификация автора исходного кода методами машинного обучения // Труды СПИИРАН. 2019. № 3 (18). С. 742-766.
4. Wang, Ningfei; Ji, Shouling; Wang, Ting. Integration of static and dynamic code stylometry analysis for programmer de-anonymization / AISec 2018 — Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2018. Association for Computing Machinery, 2018. pp. 74-84.

5. Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. De-anonymizing programmers via code stylometry / 24th USENIX Security Symposium (USENIX Security), Washington, DC. 2015.
6. IvanKrsul, Eugene H.Spafford. Authorship Analysis: Identifying The Author of a Program / The COAST Project, Department of Computer Sciences, Purdue University, West Lafayette. 1995.
7. Аврискин М.В., Павлова Е.А. Разработка приложения для определения авторства программного кода / М.В. Аврискин, Е.А. Павлова // Математическое и информационное моделирование. Издательство
   ТюмГУ, 2019. С. 54–60.
8. Стрельчёнок, Г.В. Ступенчатый метод проверки исходного кода программы на плагиат / Санкт-Петербургский государственный университет, 2016 // URL: https://nauchkor.ru/pubs/stupenchatyy-metod-proverki-ishodnogo-koda-programmy-na-plagiat-587d36555f1be77c40d58cf6 (дата обращения: 15.03.2020).
9. Guido van Rossum, Barry Warsaw, Nick Coghlan. PEP 8 — Style Guide for Python Code / Python Software Foundation. 2001 // URL: https://www.python.org/dev/peps/pep-0008/ (дата обращения: 23.03.2020).