УДК 004.891

КЛАСТЕРИЗАЦИЯ ПРЕДСТАВЛЕНИЙ ТЕКСТОВ ПРОГРАММ НА ОСНОВЕ ЦЕПЕЙ МАРКОВА

Л. А. Демидова, д.т.н., профессор кафедры корпоративных информационных систем Института информационных технологий МИРЭА – Российского технологического университета, Москва, Россия; orcid.org/0000-0003-4516-3746, e-mail: demidova.liliya@gmail.com
П. Н. Советов, к.т.н., доцент кафедры корпоративных информационных систем Института информационных технологий МИРЭА – Российского технологического университета, Москва, Россия; orcid.org/0000-0002-1039-2429, e-mail: sovetov@mirea.ru
А. В. Горчаков, аспирант кафедры корпоративных информационных систем Института информационных технологий МИРЭА – Российского технологического университета, Москва, Россия; orcid.org/0000-0003-1977-8165, e-mail: worldbeater-dev@yandex.ru

Статический анализ текстов программ при помощи методов машинного обучения успешно применяется для поиска дубликатов фрагментов кода, плагиата, для генерации подсказок в редакторах кода. ***Целью данного исследования*** *является разработка метода кластеризации программных решений типовых задач, присланных через веб-интерфейс системы «Цифровой ассистент преподавателя» (ЦАП), для выявления и анализа наиболее общих подходов к решению. При разработке метода кластеризации учитывалась особенность системы ЦАП, которая обеспечивает формирование уникальных вариантов автоматически сгенерированные задач различных типов. При реализации метода кластеризации предлагается с целью векторизации выполнить преобразование текстов программ в представления на основе цепей Маркова, построение которых производится для деревьев абстрактного синтаксиса. Это позволяет учесть особенности системы ЦАП и выполнить кластеризацию по подходам к решению задач определённого типа. Применение разработанного метода кластеризации позволило выявить основные используемые способы решений автоматически сгенерированных задач в тысячах программ, присланных студентами курса программирования на языке Python РТУ МИРЭА в весеннем семестре 2022-го года.*

Ключевые слова: *анализ текстов программ, анализ программного кода, алгоритм кластеризации, цепи Маркова, деревья абстрактного синтаксиса.*

DOI: 10.21667/1995-4565-2022-81-51-64

Введение

Цифровизация мировой экономики приводит к расширению круга задач по разработке программного обеспечения, к росту востребованности специалистов, обладающих компетенциями в информационно-коммуникационных технологиях, к увеличению количества репозиториев с открытым исходным кодом. С целью ускорения и упрощения процесса разработки, а также для упрощения поддержки программных продуктов широко используются инструменты статического анализа кода. Исследования в области интеллектуальных алгоритмов анализа данных позволили автоматизировать решение таких задач статического анализа, как поиск схожих блоков кода в текстах программ, интеллектуальная генерация подсказок, поиск плагиата.

Дублирование подходов к решению задач, возникающих при разработке программного обеспечения, часто встречается как в сложных программных системах с обширной кодовой базой, над которой работает несколько специалистов, так и в репозиториях с открытым исходным кодом, не связанных друг с другом [1]. На практике программистами нередко заимствуется публично доступный исходный код, решающий схожую прикладную задачу, который затем адаптируется с целью интеграции в разрабатываемую систему. Однако разработчик также способен самостоятельно предложить и реализовать известный подход к решению

прикладной задачи, уже применяющийся в системе, над которой ведётся работа. Существуют способы обнаружения схожих шаблонов в текстах программ [1, 2] как для выполнения автоматического рефакторинга с целью избавления от дублирования кода, так и для поиска плагиата или обнаружения ошибок.

В университетской среде заимствование чужих решений со стороны студентов при сдаче текстов программ на проверку является серьёзной проблемой. Одним из способов решения данной проблемы является внедрение систем автоматического обнаружения плагиата в исходных кодах, присылаемых на проверку преподавателю. Также способом предотвращения плагиата со стороны студентов является генерация уникальных, различающихся вариантов типовых задач для каждого студента.

Известен способ кластеризации текстов программ с целью обнаружения заимствований, в котором программы представляются в виде наборов взвешенных ключевых слов некоторого языка программирования [2]. Наборы ключевых слов предварительно формируются для каждой программы, после чего попарно сравниваются с использованием коэффициента Жаккара, а кластеризация выполняется на основании попарных сходств при помощи модификации графового алгоритма кластеризации MajorClust [3]. В работе [4] описан схожий инструмент для обнаружения плагиата, основанный на преобразовании текстов программ на различных языках программирования к наборам токенов, не зависящих от языка программирования. После преобразования наборы токенов попарно сравниваются при помощи алгоритма RKR-GST [5]. Данные инструменты основаны на попарных сравнениях представлений текстов программ в целом. Также известно об алгоритмах, анализирующих деревья синтаксиса с целью обнаружения дубликатов кода [6].

Векторизация текстов программ позволяет расширить круг алгоритмов машинного обучения, которые могут быть применены для решения задач кластеризации или классификации исходного кода [7, 8]. Известно о решении задач многоклассовой классификации с целью определения языка программирования по тексту программы при помощи искусственных нейронных сетей [7]. Для выполнения предсказаний предварительно осуществляется векторизация текстов программ путём их преобразования к наборам ключевых слов и последующего обучения модели векторизации текста GloVe [9].

В работе [10] предложен способ векторизации текстов программ code2vec, основанный на построении деревьев абстрактного синтаксиса, извлечении набора путей между листьями для каждого из полученных синтаксических деревьев и глубоком обучении. Этот способ предполагает представление извлекаемых путей в виде троек, содержащих 2 терминальных символа и последовательность типов узлов абстрактного синтаксического дерева между терминальными символами. Каждому элементу тройки ставится в соответствие случайный вектор, сохраняемый в словаре. Векторы для каждой тройки конкатенируются и подаются на вход искусственной нейронной сети для получения векторного представления пути, после чего полученные векторы объединяются при помощи механизма внимания нейронной сети для формирования вектора текста программы. Такой способ использовался при решении задачи генерации рекомендаций для названий методов на основании анализа их внутренних реализаций [11]. Также известно об использовании алгоритма code2vec для автоматизации обнаружения ошибок в листингах программ [12].

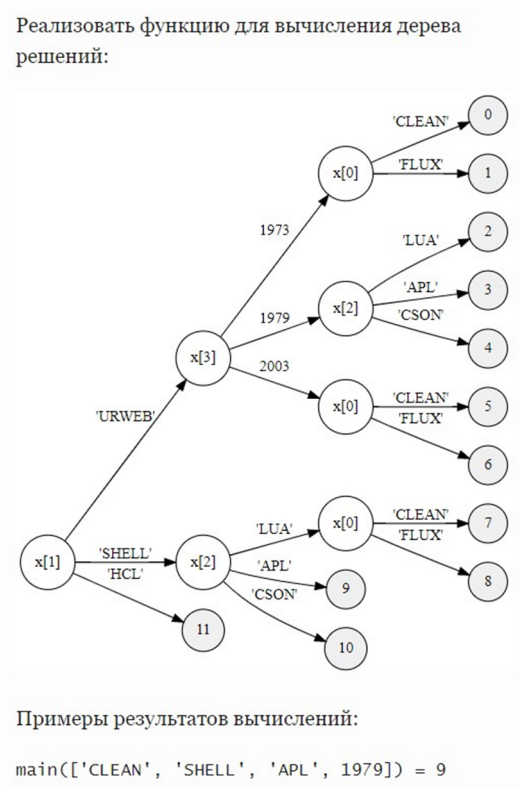
В данной статье предложен метод кластеризации программных решений типовых задач на основе представления деревьев абстрактного синтаксиса текстов программ в виде цепей Маркова. Описан опыт применения предложенного метода для кластеризации решений задач, отправленных студентами курса программирования на языке Python РТУ МИРЭА в весеннем семестре 2022-го года в систему «Цифровой ассистент преподавателя» (ЦАП) [13, 14]. Задачи в системе ЦАП порождаются генераторами случайных программ [15]. Таким образом, каждый студент курса получает индивидуальный набор уникальных задач нескольких типов. Общие способы решения задач одного и того же типа, но разных вариантов, могут иметь сходство, однако детали реализации будут различаться в зависимости от варианта, полученного студен-

том. Представление текстов программ в виде цепей Маркова позволяет учесть данную особенность и выполнить кластеризацию по наиболее общим подходам к решению задач заданного типа. Изучая выявленные кластеры и решения, являющиеся их типичными представителями, преподаватели могут оценить навыки студентов и пробелы в их знаниях. Более того, анализ текстов программ помогает понять качество самих генераторов задач. Плохой генератор задач – такой генератор, для которого возможны лишь однотипные решения.

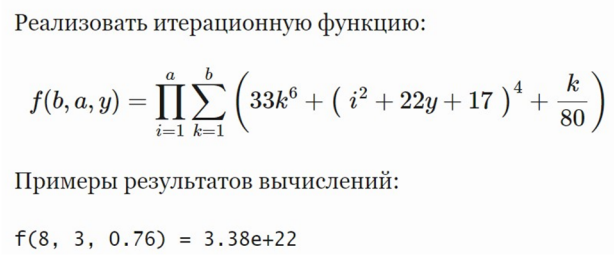
Цифровой ассистент преподавателя

Массовый характер современных университетских курсов по программированию влечет за собой ряд серьезных трудностей для преподавателя, включая сложность детальной проверки решений каждого студента, сложность разработки достаточного числа разнообразных задач в условиях, когда заимствование чужих решений без указания авторства со стороны студентов становится нормой.

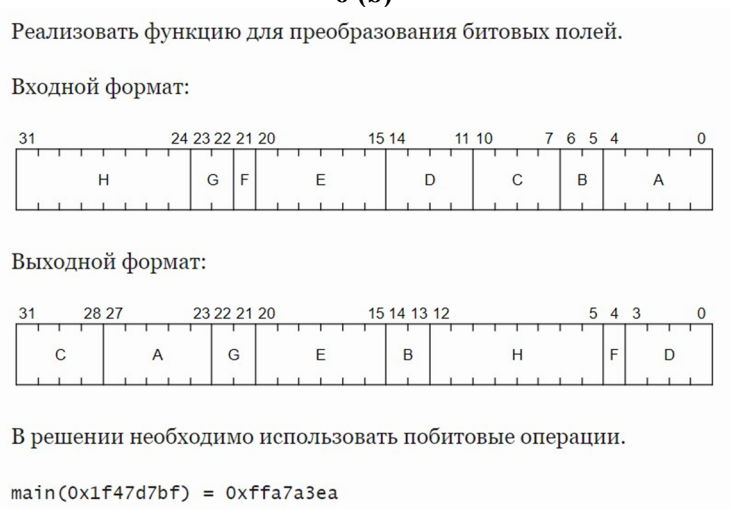
Система «Цифровой ассистент преподавателя» была разработана и внедрена в курс программирования на языке Python в РТУ МИРЭА с целью автоматизации обучения программированию [13, 14]. ЦАП автоматизирует такие виды преподавательской деятельности, как создание уникальных задач по программированию различных типов для каждого студента, проверка решений задач, ведение статистики успеваемости. Сборник индивидуальных задач для студентов генерируется заранее и представляет собой набор HTML-документов, полученных из автоматически созданных исходных файлов в формате Markdown, в которых использовалась LaTeX-разметка. Также заранее создаётся набор проверочных тестов для каждой задачи [14]. Примеры условий автоматически созданных задач представлены на рисунке 1.



а (a)



б (b)



в (c)

Рисунок 1 – Примеры условий уникальных задач, автоматически сгенерированных ЦАП, в формате HTML: а – задача на программное вычисление дерева решений, б – задача на перевод математической нотации в код, в – задача на преобразование битовых форматов данных

Figure 1 – Examples of formulating the unique tasks automatically generated by DTA in HTML format: a – task for automatic calculation of a decision tree, b – task for translating mathematical notation into code, c – task for converting between bitwise data formats

ЦАП содержит два блока задач на перевод математической нотации в код и на преобразование форматов данных. Отправка решений и учёт статистики успеваемости студентов осуществляется при помощи веб-интерфейса ЦАП, включающего страницы перечня групп, списка вариантов, состояния задачи. Автоматическая проверка отправленных студентами решений включает статический анализ текста программы и тестирование кода в изолированной среде. Тексты программ и результаты их автоматической проверки сохраняются в базе данных ЦАП с целью ведения статистики активности и последующего анализа полученных решений.

Векторизация и кластеризация текстов программ

Задача векторизации текста программы сводится к разработке алгоритма, реализующего отображение $f : P \to \mathbb{R}^m$, где P обозначает множество строк, содержащих текст программы, а m обозначает размерность целевого векторного пространства. Преобразование исходных текстов программ в векторы позволяет расширить круг алгоритмов машинного обучения, которые могут быть применены для анализа кода [8, 10]. Кроме того, становится доступным применение мер сходства объектов, определяемых над векторными пространствами.

При векторизации текстов программ, присылаемых на проверку системе ЦАП, с целью последующего применения метода или алгоритма кластеризации для выявления общих подходов к решению типовых задач, необходимо учитывать, что каждый студент решает свой уникальный вариант. В разных вариантах различаются такие параметры, как число операций, число условий, используемые функции и др. Примеры различных вариантов задачи одного и того же типа представлены на рисунке 2.

Реализовать функцию по рекуррентной формуле:

$$f_n = \begin{cases} 0.73, & n = 0; \\ \sin^2\left(75f_{n-1} - 42f_{n-1}^3 - f_{n-1}^2\right), & n \geq 1. \end{cases}$$

Реализовать функцию по рекуррентной формуле:

$$f_n = \begin{cases} -0.84, & n = 0; \\ 0.27, & n = 1; \\ \frac{89 - f_{n-2} - 42f_{n-1}^3}{84}, & n \geq 2. \end{cases}$$

Реализовать функцию по рекуррентной формуле:

$$f_n = \begin{cases} -0.82, & n = 0; \\ \operatorname{arctg}^3\left(70f_{n-1} - \frac{f_{n-1}^3}{4}\right), & n \geq 1. \end{cases}$$

а (a) **б (b)** **в (c)**

Рисунок 2 – Примеры различных вариантов задачи одного и того же типа
Figure 2 – Examples of different variants of tasks of the same type

Каждый вариант задачи из представленных на рисунке 2 может быть решен различными способами. При написании программ для типовых задач студенты не ограничены в выборе подходов к их решению. Полученные от студентов тексты программ, реализующие функции, представленные на рисунках 2 *а*, *б*, *в*, показаны на рисунках 3 *а*, *б*, *в* соответственно.

```
from math import sin

def main(n):
    if n == 0:
        return 0.73
    c = main(n - 1)
    b = 75*c - 42*c**3 - c**2
    return pow(sin(b), 2)
```

а (a)

```
def main(n):
    a = [-0.84, 0.27]
    for i in range(2, n + 1):
        x = 89 - a[i-2] - 42*a[i-1]**3
        a.append(x / 84)
    return a[n]
```

б (b)

```
from math import atan

def main(n):
    return -0.82 if n == 0 else
atan(70 * main(n-1) - (main(n-1)
** 3) / 4) ** 3
```

в (c)

Рисунок 3 – Примеры решений для вариантов задач, показанных на рисунке 2: а – решение при помощи рекурсивного вызова, б – решение при помощи цикла, в – тернарный оператор
Figure 3 – Examples of solutions for typical tasks shown in Fig. 2: a – solution using recurrent function calls, b – solution by converting recursion into a loop, c – ternary operator

Векторы, описывающие тексты программ, содержащие различающиеся способы решения разных вариантов задачи одного и того же типа (рисунок 3), должны быть значительно удалены друг от друга в целевом векторном пространстве. Напротив, векторы, описывающие тексты со схожими подходами к решению, не должны располагаться далеко друг от друга.

Векторизация текстов программ на основе деревьев абстрактного синтаксиса (англ. Abstract Syntax Tree, AST) позволяет учесть иерархические связи между элементами программы [8, 11], что выгодно отличает данный подход от подходов, основанных на сравнении наборов ключевых слов программ. Дерево абстрактного синтаксиса для программы на языке Python, представленной на рисунке. 3 *б*, полученное про помощи модуля AST [16] и визуализированное с помощью библиотеки graphviz [17], представлено на рисунке 4. При построении синтаксического дерева предлагается удалить незначащие узлы, такие как Import, Load, Store, alias, arguments, arg.

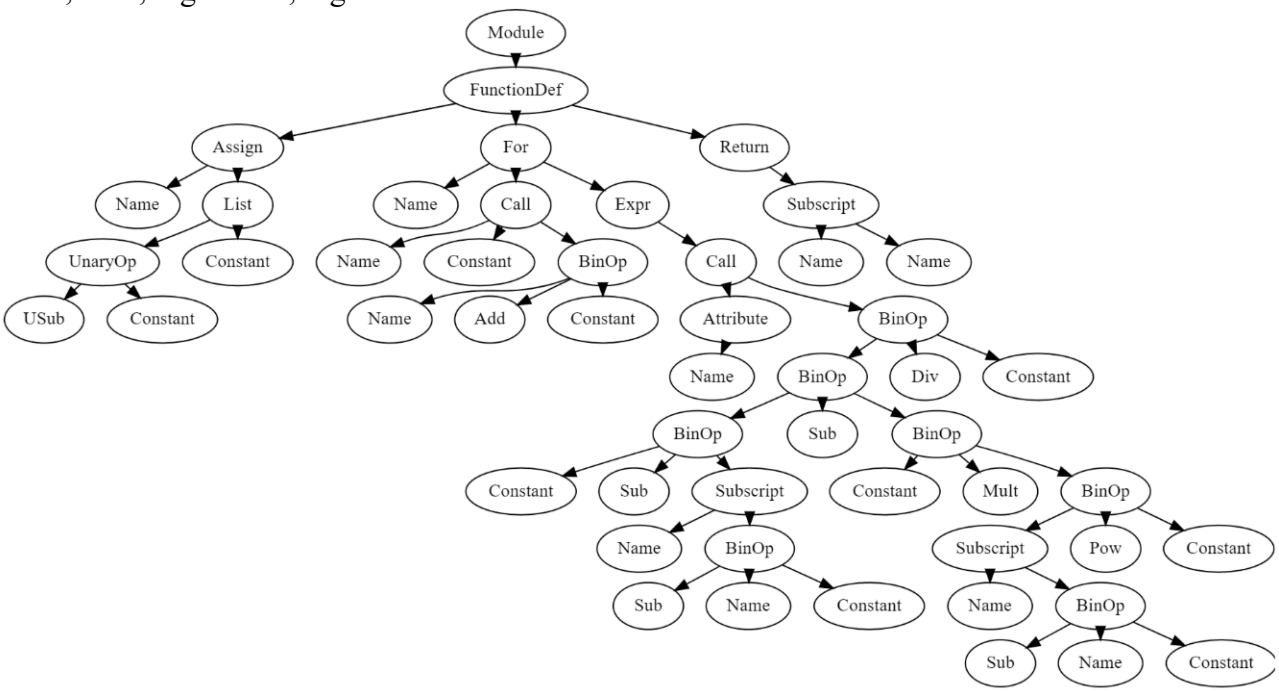


Рисунок 4 – Дерево абстрактного синтаксиса для программы, представленной на рисунке 3, *б*
Figure 4 – Abstract syntax tree for a program shown in Figure 3, *b*

Цепь Маркова – последовательность зависимых случайных величин $X_0, X_1, \ldots,$ причём условное распределение случайной величины $X_n, n \geq 1,$ зависит только от значения X_{n-1}. Предлагаемый подход к векторизации деревьев абстрактного синтаксиса, аналогичных представленному на рисунке 4, предполагает преобразование дерева к цепи Маркова с пространством состояний, состоящим из типов вершин, встречающихся в дереве. При преобразовании синтаксического дерева предлагается не учитывать узлы типов Module, keyword, заменить узел Return на узел Assign. Тогда ориентированный граф переходов для полученной цепи Маркова будет иметь вид, показанный на рисунке 5.

Матрица смежности для графа, представленного на рисунке 5, принадлежит пространству $\mathbb{R}^{18\times 18}$, поскольку пространство состояний цепи Маркова содержит 18 типов вершин, встречающихся в соответствующем дереве абстрактного синтаксиса. В зависимости от числа различных типов вершин t в исходном синтаксическом дереве, матрица смежности для графа переходов определена в пространстве $\mathbb{R}^{t\times t}$ и тривиально может быть преобразована к вектору, принадлежащему пространству $\mathbb{R}^m$, где $m = t^2$.

Таким образом, при реализации отображения $f : P \to \mathbb{R}^m$, выполняющего векторизацию текстов программ P преобразованием матриц смежности графов переходов цепей Маркова для деревьев абстрактного синтаксиса в m-мерные векторы, необходимо заранее определить множество учитываемых типов вершин графа переходов. Это множество, имеющее мощность m, может быть получено в процессе анализа деревьев абстрактного синтаксиса всех программных решений, входящих в набор текстов программ для векторизации.

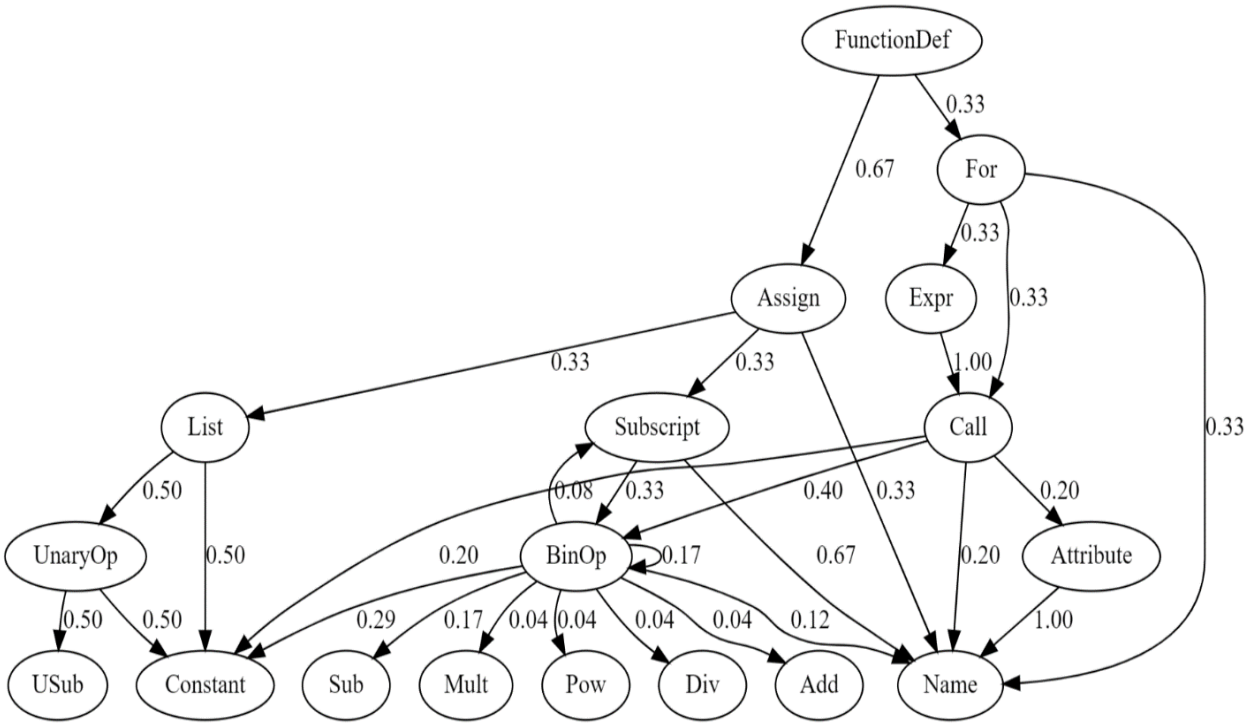


Рисунок 5 – Граф переходов цепи Маркова для дерева абстрактного синтаксиса (см. рисунок 4)
Figure 5 – Markov chain transition graph for abstract syntax tree (see Figure 4)

Результатом применения отображения $f : P \to \mathbb{R}^m$ к каждому объекту из набора исходных текстов, состоящего из n текстов программ, является матрица, принадлежащая пространству $\mathbb{R}^{n \times m}$. Строки такой матрицы кодируют n m-мерных векторных представлений текстов. Перед применением алгоритма кластеризации происходит преобразование набора m-мерных векторов текстов программ к матрице попарных сходств, принадлежащей пространству $\mathbb{R}^{n \times n}$, где n – число объектов в исследуемом наборе данных.

Поскольку весами рёбер ориентированного графа переходов цепи Маркова являются вероятности, а векторизация текстов программ выполняется методом преобразования деревьев абстрактного синтаксиса к цепям Маркова, для сравнения векторных представлений текстов программ предлагается использовать специализированную меру сходства двух распределений вероятностей Йенсена – Шеннона (англ. Jensen-Shannon Divergence), также известную как полное отклонение от среднего [18]. При построении матрицы попарных сходств в пространстве $\mathbb{R}^{n \times n}$, происходит сравнение каждой пары векторов $\vec{v}_i, \vec{v}_j$ из полученного набора векторных представлений, соответствующих текстам программ, причём $i \neq j$. Полученное значение записывается в матрицу попарных сходств с индексами i и j. Мера сходства Йенсена – Шеннона в этом случае имеет вид:

$$\mathrm{JSD}\left(\vec{v}_i, \vec{v}_j\right) = \frac{1}{2}\sum_{k=1}^{m} v_{ik} \log_2 \frac{v_{ik}}{\frac{1}{2}\left(v_{ik} + v_{jk}\right)} + \frac{1}{2}\sum_{k=1}^{m} v_{jk} \log_2 \frac{v_{jk}}{\frac{1}{2}\left(v_{ik} + v_{jk}\right)}, \quad (1)$$

где $\vec{v}_i$ и $\vec{v}_j$ – сравниваемые векторы, принадлежащие векторному пространству $\mathbb{R}^m$, v_{ik} – k-я компонента вектора $\vec{v}_i$, v_{ik} – k-я компонента вектора $\vec{v}_j$.

Значения меры Йенсена – Шеннона (1) принадлежат вещественному промежутку $[0, 1]$. Полученная матрица попарных сходств в пространстве $\mathbb{R}^{n \times n}$ подаётся на вход агломеративному алгоритму иерархической кластеризации. При выполнении иерархической кластеризации предлагается использовать метод средней связи, как компромиссный вариант между ме-

тодами одиночной и полной связи [19]. При использовании невзвешенного метода средней связи расстояние между двумя кластерами C_I и C_K полагается равным среднему расстоянию между объектами, принадлежащими данным кластерам:

$$\mathrm{R}_{avg}\left(C_I, C_K\right) = \frac{1}{\left|C_I\right| \times \left|C_K\right|} \sum_{\vec{v}_i \in C_I} \sum_{\vec{v}_k \in C_K} d\left(\vec{v}_i, \vec{v}_k\right), \tag{2}$$

где C_I и C_K – сравниваемые кластеры, $\left|C_I\right|$ – число объектов, отнесённых к кластеру C_I, $\left|C_K\right|$ – число объектов, отнесённых к кластеру C_K, $\vec{v}_i$ и $\vec{v}_k$ – объекты, принадлежащие кластерам C_I и C_K соответственно, $d\left(\vec{v}_i, \vec{v}_k\right)$ – расстояние между объектами $\vec{v}_i$ и $\vec{v}_k$, расстояние вычисляется заранее согласно формуле (1) и хранится в матрице попарных сходств. Под объектами в рассматриваемой задаче будем понимать векторные представления текстов программ.

Оптимальное число кластеров предлагается выбирать автоматически при использовании метрики индекса кластерного силуэта [20]. Эта метрика позволяет численно оценивать качество разбиения исходного набора данных алгоритмом кластеризации на заданное число кластеров: индекс кластерного силуэта позволяет определить, насколько каждый объект похож на другие объекты в своём кластере и насколько не похож на объекты из других кластеров. Формула для вычисления силуэта *i*-го объекта $\vec{v}_i$ имеет вид:

$$s\left(\vec{v}_i\right) = \begin{cases} 1 - \dfrac{a\left(\vec{v}_i\right)}{b\left(\vec{v}_i\right)}, \; a\left(\vec{v}_i\right) < b\left(\vec{v}_i\right), \\ 0, \; a\left(\vec{v}_i\right) = b\left(\vec{v}_i\right), \\ \dfrac{b\left(\vec{v}_i\right)}{a\left(\vec{v}_i\right)} - 1, \; a\left(\vec{v}_i\right) > b\left(\vec{v}_i\right), \end{cases} \tag{3}$$

где $\vec{v}_i$ – объект, силуэт которого оценивается, $a\left(\vec{v}_i\right)$ – среднее расстояние от $\vec{v}_i$ до остальных объектов кластера, к которому принадлежит $\vec{v}_i$, $b\left(\vec{v}_i\right)$ – минимальное среднее расстояние от $\vec{v}_i$ до всех объектов другого кластера из тех кластеров, к которым $\vec{v}_i$ не принадлежит.

Значение $a\left(\vec{v}_i\right)$ в формуле (3) оценивается как:

$$a\left(\vec{v}_i\right) = \frac{1}{\left|C_I\right| - 1} \sum_{\vec{v}_k \in C_I, k \neq j} d\left(\vec{v}_i, \vec{v}_k\right), \tag{4}$$

где $\vec{v}_i$ – объект, для которого ищется среднее расстояние до остальных объектов его кластера C_I, $\vec{v}_k$ – другой объект, также принадлежащий кластеру C_I, $\left|C_I\right|$ – мощность кластера, I – номер кластера, $d(\vec{v}_i, \vec{v}_k)$ – расстояние между $\vec{v}_i$ и $\vec{v}_k$, определяемое по формуле (1).

Значение $b\left(\vec{v}_i\right)$ в формуле (3) оценивается как:

$$b\left(\vec{v}_i\right) = \min_{I \neq K} \frac{1}{\left|C_K\right|} \sum_{\vec{v}_k \in C_K} d\left(\vec{v}_i, \vec{v}_k\right), \tag{5}$$

где $\vec{v}_i$ – объект, для которого ищется минимальное среднее расстояние до всех объектов другого кластера из тех, к которым $\vec{v}_i$ не принадлежит, $\vec{v}_k$ – объект из кластера C_K, к которому $\vec{v}_i$ не принадлежит, $\left|C_K\right|$ – мощность кластера, K – номер кластера, $d\left(\vec{v}_i, \vec{v}_k\right)$ – расстояние между объектами $\vec{v}_i$ и $\vec{v}_k$, определяемое по формуле (1).

Из формул (3), (4) и (5) следует, что значение индекса силуэта $s(\vec{v}_i)$ для *i*-й точки принадлежит вещественному промежутку $[-1,1]$. Для набора данных, разбитого на кластеры, индекс кластерного силуэта вычисляют как:

$$s_{avg}(V) = \sum_{i=1}^{n} s(\vec{v}_i), \tag{6}$$

где $\vec{v}_i$ – *i*-й объект, $s(\vec{v}_i)$ – индекс (3) для объекта $\vec{v}_i \in V$, n – число объектов в наборе V.

Значение индекса кластерного силуэта (6), равное 1, считается наилучшим, а значение, равное –1, считается наихудшим. При этом значения, близкие к 0, обозначают перекрывающиеся кластеры, а отрицательные значения показывают, что объект, вероятно, был отнесён к неправильному кластеру, поскольку имеет сходства с объектами из другого кластера.

Таким образом, последовательность действий при применении предложенного метода кластеризации текстов программ по подходам к решению уникальных автоматически сгенерированных задач имеет вид, представленный на рисунке 6.

Вход: P – множество текстов программных решений уникальных задач,
$k_{min} \in \mathbb{N}$ – минимальное число кластеров,
$k_{max} \in \mathbb{N}$ – максимальное число кластеров.

1. **установить** множество графов переходов цепей Маркова $G = \varnothing$.
2. **для каждого** текста программы $p_i \in P$ **выполнять**:
3. **построить** дерево абстрактного синтаксиса a_i для p_i (см. рисунок 4)
4. **удалить** из a_i вершины { Import, Load, Store, alias, arguments, arg, Module, keyword }
5. **заменить** в a_i вершины Return **на** вершины Assign
6. **построить** взвешенный граф переходов g_i цепи Маркова для a_i (см. рисунок 5)
7. $G \leftarrow G \bigcup \{g_i\}$
8. **конец цикла**
9. **установить** $h \in \mathbb{N}$ равным числу типов вершин во всех графах из множества G
10. **установить** множество векторных представлений текстов программ $V = \varnothing$
11. **для каждого** графа переходов цепи Маркова $g_i \in G$ **выполнять:**
12. **построить** матрицу смежности $\boldsymbol{m}_i \in \mathbb{R}^{h \times h}$ для взвешенного графа g_i
13. **преобразовать** матрицу $\boldsymbol{m}_i$ к вектору $\vec{v}_i \in \mathbb{R}^{h \cdot h}$
14. $V \leftarrow V \bigcup \{\vec{v}_i\}$
15. **конец цикла**
16. **построить** матрицу попарных сходств $M \in \mathbb{R}^{|V| \times |V|}$ векторов из V по формуле (1)
17. **установить** множество разбиений набора V на кластеры $C = \varnothing$
18. **для каждого** числа кластеров $k \in \mathbb{N}$ из промежутка $[k_{min}, k_{max}]$ **выполнять:**
19. **разделить** V на k кластеров C_k методом (2) на основе матрицы сходств M
20. **вычислить** индекс кластерного силуэта s_k по формуле (3) для разделения C_k
$C \leftarrow C \bigcup \{(C_k, s_k)\}$
21. **конец цикла**
22. **возвратить** разделение C_k с наибольшим индексом s_k из множества C

Рисунок 6 – Кластеризация представлений текстов программ на основе цепей Маркова
Figure 6 – Clustering of representations of source texts based on Markov chains

Экспериментальные исследования

Анализируемые наборы данных, представленные текстами программ на языке Python, для 9 типов автоматически сгенерированных задач содержат различное число объектов. Число объектов в наборах данных для задач 9 типов приведено в таблице 1. Также в таблице приведены размерности векторов, полученные при преобразовании матрицы смежности графа переходов цепи Маркова в процессе векторизации текстов программ.

Таблица 1 – Число текстов программ, решающих задачи разных типов
Table 1 – The number of objects in source code sets that solve problems of different types

Номер типа задачи	1	2	3	4	5	6	7	8	9
Число текстов программ	1301	1203	1172	1183	1130	672	1113	997	886
Размерность вектора	625	1764	1764	1936	1681	3481	2304	2704	3844

Номера типов с 1-го по 5-й в таблице 1 обозначают задачи, принадлежащие блоку задач на перевод математической нотации в код на языке Python (таблица 2), номера типов с 6-го по 9-й в таблице 1 обозначают задачи, принадлежащие блоку задач на преобразование форматов данных (таблица 2). Размерность m вектора для текста программы зависит от числа строк h в матрице смежности, соответствующего программе графа переходов состояний цепи Маркова, причём $m = h^2$. Так, число строк матрицы смежности h для задачи первого типа равно 25, а размерность вектора $m = 25^2 = 625$ (см. таблицу 1). Число строк в матрице смежности h определяется числом всех различных типов вершин деревьев синтаксического разбора, фигурирующих в наборе данных. Согласно табл. 1, число различных типов вершин в наборах программных решений растёт с усложнением задач, что может указывать на рост вариативности подходов к решению. Число присланных студентами программ, напротив, уменьшается с увеличением номера типа задачи.

Таблица 2 – Типы автоматически сгенерированных задач в ЦАП
Table 2 – The types of tasks automatically generated by DTA

Номер типа задачи	Задача
1.	Реализовать функцию.
2.	Реализовать кусочную функцию.
3.	Реализовать итерационную функцию.
4.	Реализовать рекуррентную функцию.
5.	Реализовать функцию, оперирующую векторами.
6.	Реализовать дерево решений.
7.	Реализовать функцию для преобразования битовых полей.
8.	Реализовать конечный автомат мили в виде класса.
9.	Реализовать разбор двоичного формата данных.

После векторизации текстов программ для каждого набора данных была построена матрица попарных сходств, в качестве меры сходства использовалась мера Йенсена – Шеннона согласно формуле (1). После этого матрица попарных сходств подавалась на вход агломеративному алгоритму иерархической кластеризации с методом средней связи (2), реализованному в библиотеке sklearn [21]. Для выбора оптимального числа кластеров использовался индекс кластерного силуэта (3). Графические зависимости индекса кластерного силуэта от числа кластеров для задач различных типов представлены на рисунке 7.

Для блока задач на перевод математической нотации в код оптимальное число кластеров выбиралось из целочисленного диапазона от 5 до 30. Для блока задач на преобразование форматов данных подходы к решениям более вариативны, поэтому оптимальное число кластеров выбиралось из целочисленного диапазона от 15 до 60. Выбранное на основании рисунка 7 число кластеров для наборов программных решений задач различных типов представлено в таблице 3. Также в таблице 3 приведены значения индекса кластерного силуэта (3) для наилучшего числа кластеров.

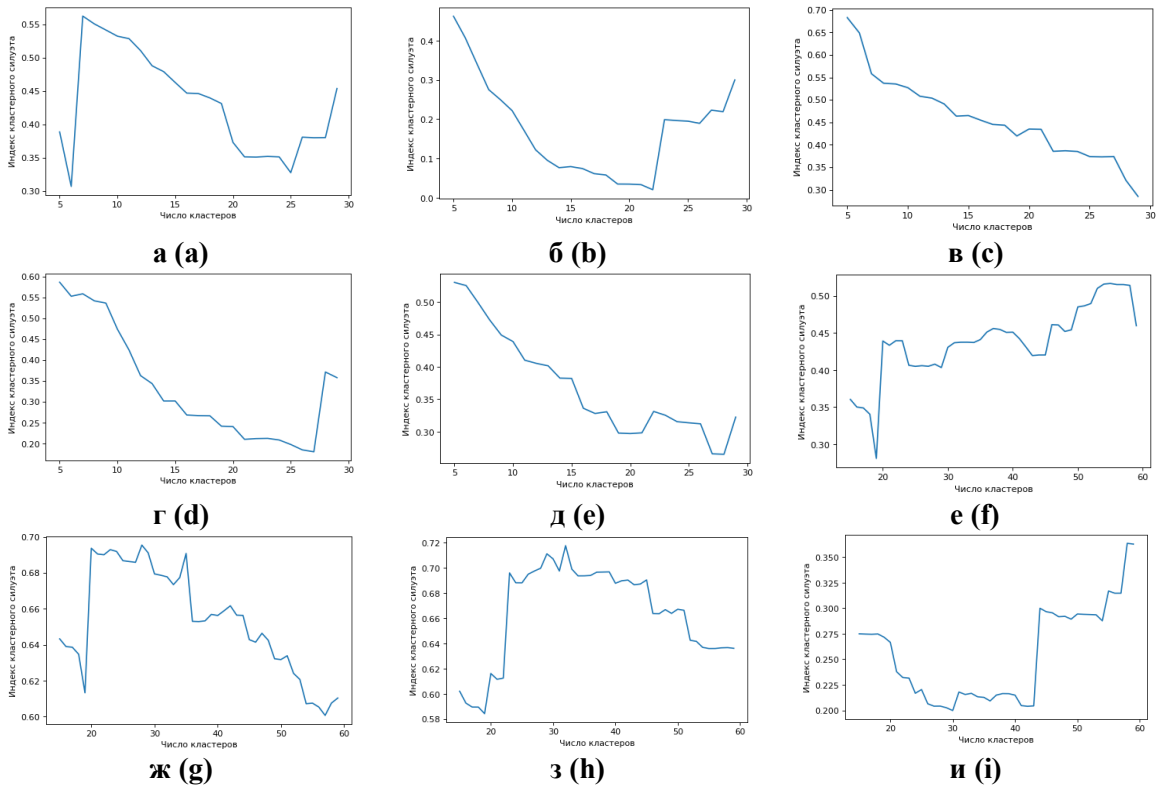


Рисунок 7 – Графические зависимости индекса кластерного силуэта (3) от числа кластеров для задач различных типов (см. табл. 2): (а) тип 1, (б) тип 2, (в) тип 3, (г) тип 4, (д) тип 5, (е) тип 6, (ж) тип 7, (з) тип 8, (и) тип 9
Figure 7 – Dependence of cluster silhouette score (3) on the number of clusters for the tasks of different types (see table 2): (a) type 1, (b) type 2, (c) type 3, (d) type 4, (e) type 5, (f) type 6, (g) type 7, (h) type 8, (i) type 9

Таблица 3 – Число объектов в наборах текстов программ, решающих задачи разных типов
Table 3 – The number of objects in source code sets that solve problems of different types

Номер типа задачи	1	2	3	4	5	6	7	8	9
Число кластеров	7	5	5	5	5	55	28	32	58
Индекс кластерного силуэта	0,56	0,46	0,68	0,58	0,53	0,51	0,69	0,71	0,36

Согласно таблице 3, во всех случаях значение индекса кластерного силуэта положительное. Это свидетельствует о том, что алгоритмом иерархической кластеризации объекты из наборов данных были успешно разделены на заданное число непересекающихся кластеров. Выбранное число кластеров растёт с усложнением задач. Для несложных задач на перевод математической нотации в код тексты программ разбиваются на сравнительно малое число кластеров, содержащих решения со схожими подходами. Для задач на преобразование форматов данных выявлено большее число кластеров, что может свидетельствовать о большей вариативности подходов к решению задач.

Примеры решений задач, принадлежащих различным кластерам, представлены на рисунке 8.

Решение, представленное на рисунке 8 *б*, принадлежит наиболее многочисленному кластеру, содержащему 1197 объектов из 1203 (таблица 1). Неожиданным для преподавателей курса программирования на языке Python стало выявление кластера, содержащего 3 объекта, к которому принадлежит решение, представленное на рисунке 8 *а*. Это решение не содержит условных операторов, а кусочная функция описывается словарём, ключи которого имеют логический тип данных. Интересен также кластер с решениями, аналогичными показанному на рисунке 8 *в*, решающими задачу в одну строчку тернарными операторами. Оставшиеся

два кластера содержали решения с контекстами модуля decimal и комбинированным присваиванием.

```
import math

def main(y):
    return {
        y < 88: pow((y + 1 + y * y * y / 22), 6) + 91,
        88 <= y < 138: 73 * pow(math.atan(y), 5) - 1,
        138 <= y < 234: 81 - 2*y*y*y,
        y >= 234: 1 + 83*(round(41*y*y*y - 0.02)**6)
    }[True]
```

```
from math import log, atan

def main(x):
    if x < 135:
        a = 37 + atan(x)**5 - 21*x**3 - 57*(5-x)**6
    if 135 <= x < 231:
        a = 18*log(x, 2)**7
    if 231 <= x < 280:
        a = (63*x*x + 15*x**3 + 24*x)**3
    if x >= 280:
        a = x**0.5**3 + abs(x**3 + x**2)
    return a
```

```
import math

def main(x):
    return 42 * (x + x ** 2) ** 3 - math.atan(x) ** 5 if x < 110 else \
        math.log10(x ** 3 - 9 * x ** 2) / 43 + math.ceil(x ** 2) ** 3 + \
        17 * x ** 6 if x < 199 else \
        37 * x ** 4 if x < 216 else \
        10 * x ** 3 - 4 * x - (1 + x ** 3) ** 7 if x < 279 else \
        (1 + 55 * x ** 2 + x / 88) ** 5
```

Рисунок 8 – Примеры принадлежащих различным кластерам решений задач на перевод математической нотации кусочной функции в код
Figure 8 – Examples of solutions for problems of translating the mathematical notation of a partial function into code, belonging to different clusters

Заключение

В результате проведённого исследования был разработан метод кластеризации текстов программ, являющихся решениями задач различных типов. Предложенный метод основан на получении для текста программы дерева синтаксического разбора, которое затем преобразуется к взвешенному ориентированному графу переходов между состояниями цепи Маркова для рассматриваемого синтаксического дерева. Вершинами в полученном графе переходов состояний цепи Маркова являются типы узлов дерева синтаксиса, а весами рёбер являются вероятности переходов между узлами. Матрицы смежности полученных графов переходов для текстов программ затем преобразуются в векторы, которые попарно сравниваются при помощи меры Йенсена – Шеннона. Затем к матрице попарных сходств применяется алгоритм иерархической кластеризации, при этом оптимальное число кластеров выбирается на основании значений индекса кластерного силуэта.

Предложенный метод кластеризации текстов программ позволил оценить подходы студентов к решению уникальных задач, порождаемых системой «Цифровой ассистент преподавателя» (ЦАП) [13, 14]. Выявлено, что выбранные подходы к решению задач студентами курса программирования на языке Python РТУ МИРЭА зависели от способов, разобранных преподавателями на практических занятиях, причём с усложнением задач растёт вариативность подходов к их решению. Дальнейшая работа может быть направлена на разработку интеграции предложенного метода кластеризации текстов программ в систему ЦАП для оценки подходов к решению в режиме реального времени преподавателями в течение семестра. Кроме того, целесообразно изучить возможность применения предложенного подхода для кластеризации ошибочных решений задач с целью оперативного оповещения студентов о наличии проблем в присланном в систему ЦАП коде, что может позволить автоматизировать

процесс помощи студентам и разгрузить преподавателей практических занятий, позволив им уделять больше времени иным активностям.

Библиографический список

1. **Marcus A., Maletic J. I.** Identification of high-level concept clones in source code // Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001). IEEE, 2001. pp. 107-114.

2. **Moussiades L., Vakali A.** PDetect: A clustering approach for detecting plagiarism in source code datasets // The computer journal. 2005, vol. 48, no. 6, pp. 651-661.

3. **Stein B., Niggemann O.** On the nature of structure and its identification // International Workshop on Graph-Theoretic Concepts in Computer Science. Springer, Berlin, Heidelberg, 1999. pp. 122-134.

4. **Kustanto C., Liem I.** Automatic source code plagiarism detection // 2009 10th ACIS International conference on software engineering, artificial intelligences, networking and parallel/distributed computing. IEEE, 2009. pp. 481-486.

5. **Wise M. J.** YAP3: Improved detection of similarities in computer program and other texts // Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education. 1996. pp. 130-134.

6. **Jiang L., Misherghi G., Su Z., Glondu S.** Deckard: Scalable and accurate tree-based detection of code clones // 29-th International Conference on Software Engineering (ICSE'07). IEEE, 2007. pp. 96-105.

7. **Azcona D., Arora P., Hsiao I., Smeaton A.** user2code2vec: Embeddings for profiling students based on distributional representations of source code // Proceedings of the 9th International Conference on Learning Analytics & Knowledge. 2019. pp. 86-95.

8. **Gilda S**. Source code classification using Neural Networks // 2017 14th international joint conference on computer science and software engineering (JCSSE). IEEE, 2017. pp. 1-6.

9. **Pennington J., Socher R., Manning C. D.** Glove: Global vectors for word representation // Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014, pp. 1532-1543.

10. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code // Proceedings of the ACM on Programming Languages. 2019, vol. 3, pp. 1-29.

11. **Jiang L., Liu H., Jiang H.** Machine learning based recommendation of method names: how far are we // 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019. pp. 602-614.

12. **Briem J. A., Smit J., Sellik H., Rapoport P.** Using distributed representation of code for bug detection. arXiv preprint arXiv:1911.12863. 2019.

13. **Андрианова Е. Г., Демидова Л. А., Советов П. Н.** «Цифровой ассистент преподавателя» в массовом профессиональном обучении для цифровой экономики // Российский технологический журнал. 2022. Т. 10. №. 3. С. 7-23.

14. **Sovietov P. N., Gorchakov A. V.** Digital Teaching Assistant for the Python Programming Course // 2022 2nd International Conference on Technology Enhanced Learning in Higher Education (TELE). IEEE, 2022, pp. 272-276.

15. **Sovietov P.** Automatic generation of programming exercises // 2021 1st International Conference on Technology Enhanced Learning in Higher Education (TELE). IEEE, 2021, pp. 111-114.

16. AST – Abstract Syntax Trees [Digital resource]. Python 3.10.5 Documentation. URL: https://docs.python.org/3/library/ast.html (accessed at: 23.06.2022)

17. **Gansner E. R., North S. C.** An open graph visualization system and its applications to software engineering // Software: practice and experience. 2000, vol. 30, no. 11, pp. 1203-1233.

18. **Dagan I., Lee L., Pereira F**. Similarity-based methods for word sense disambiguation // Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics. 1997.

19. **Griffiths A., Robinson L. A., Willett P.** Hierarchic agglomerative clustering methods for automatic document classification // Journal of Documentation. 1984, vol. 40, no. 3, pp. 175-205.

20. **Shutaywi M., Kachouie N. N.** Silhouette analysis for performance evaluation in machine learning with applications to clustering // Entropy. 2021, vol. 23, no. 6, pp. 759.

21. Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau, D., Brucher M., Perrot M., Duches-

nay E. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research. 2011, vol. 12, pp. 2825-2830.

UDC 004.891

CLUSTERING OF PROGRAM SOURCE TEXT REPRESENTATIONS BASED ON MARKOV CHAINS

L. A. Demidova, Dr. Sc. (Tech.), full Professor, Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University, Moscow, Russia;
orcid.org/0000-0003-4516-3746, e-mail: demidova.liliya@gmail.com
P. N. Sovietov, Ph.D. (Tech.), associated Professor, Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University, Moscow, Russia;
orcid.org/0000-0002-1039-2429, e-mail: sovetov@mirea.ru
A. V. Gorchakov, post-graduate student, Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University, Moscow, Russia;
orcid.org/0000-0003-1977-8165, e-mail: worldbeater-dev@yandex.ru

Static analysis of program sources using machine learning algorithms has been successfully applied to solve a number of common problems, including the search for duplicate code fragments, intelligent generation of hints, and the search for plagiarism. ***The aim of this study*** *is to develop a method for clustering software solutions to typical tasks sent through the web interface of Digital Teaching Assistant (DTA) system to identify and analyze the most common approaches to solving the tasks. Notably, it is necessary to take into account the peculiarity of DTA system, in which each student receives unique automatically generated variants of typical tasks. The method proposed for preliminary transformation of program sources into vectors based on Markov chains for abstract syntax trees allows taking into account the mentioned peculiarity of DTA system and performing clustering based on the approaches used to solve the tasks, not based on task variants. The application of the developed method made it possible to assess the gaps in skills and knowledge of students of the RTU MIREA Python programming course in spring semester 2022.*

Key words: *source code analysis, program code analysis, clustering algorithms, Markov chains, cluster profiling, abstract syntax trees, online education.*

DOI: 10.21667/1995-4565-2022-81-51-64

References

1. **Marcus A., Maletic J. I.** Identification of high-level concept clones in source code. *Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001). IEEE*, 2001, pp. 107-114.

2. **Moussiades L., Vakali A.** Detect: A clustering approach for detecting plagiarism in source code datasets. *The computer journal*. 2005, vol. 48, no. 6, pp. 651-661.

3. **Stein B., Niggemann O.** On the nature of structure and its identification. *International Workshop on Graph-Theoretic Concepts in Computer Science. Springer, Berlin, Heidelberg*, 1999. pp. 122-134.

4. **Kustanto C., Liem I.** Automatic source code plagiarism detection. *2009 10th ACIS International conference on software engineering, artificial intelligences, networking and parallel/distributed computing. IEEE,* 2009. pp. 481-486.

5. **Wise M. J.** YAP3: Improved detection of similarities in computer program and other texts. *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education.* 1996. pp. 130-134.

6. **Jiang L., Misherghi G., Su Z., Glondu S.** Deckard: Scalable and accurate tree-based detection of code clones. *29th International Conference on Software Engineering (ICSE'07). IEEE*, 2007, pp. 96-105.

7. **Azcona D., Arora P., Hsiao I., Smeaton A.** user2code2vec: Embeddings for profiling students based on distributional representations of source code. *Proceedings of the 9th International Conference on Learning Analytics & Knowledge.* 2019. pp. 86-95.

8. **Gilda S**. Source code classification using Neural Networks. *2017 14th international joint conference on computer science and software engineering (JCSSE). IEEE*, 2017, pp. 1-6.

9. **Pennington J., Socher R., Manning C. D.** Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014. pp. 1532-1543.

10. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*. 2019, vol. 3, p. 1-29.

11. **Jiang L., Liu H., Jiang H.** Machine learning based recommendation of method names: how far are we. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE*, 2019, pp. 602-614.

12. **Briem J. A., Smit J., Sellik H., Rapoport P.** Using distributed representation of code for bug detection. *arXiv preprint arXiv:1911.12863*. 2019.

13. **Andrianova E. G., Demidova L. A., Sovetov P. N.** Pedagogical design of a digital teaching assistant in massive professional training for the digital economy. *Russian Technological Journal*. 2022, vol. 10, no. 3, pp. 7-23.

14. **Sovietov P. N., Gorchakov A. V.** Digital Teaching Assistant for the Python Programming Course. *2022 2nd International Conference on Technology Enhanced Learning in Higher Education (TELE). IEEE,* 2022, pp. 272-276.

15. **Sovietov P.** Automatic generation of programming exercises. *2021 1st International Conference on Technology Enhanced Learning in Higher Education (TELE). IEEE*, 2021, pp. 111-114.

16. AST – Abstract Syntax Trees [Digital resource]. Python 3.10.5 Documentation. URL: https://docs.python.org/3/library/ast.html (accessed at: 23.06.2022)

17. **Gansner E. R., North S. C.** An open graph visualization system and its applications to software engineering // Software: practice and experience. 2000, vol. 30, no. 11, pp. 1203-1233.

18. **Dagan I., Lee L., Pereira F**. Similarity-based methods for word sense disambiguation. *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*. 1997.

19. **Griffiths A., Robinson L.A., Willett P.** Hierarchic agglomerative clustering methods for automatic document classification. *Journal of Documentation.* 1984, vol. 40, no. 3, pp. 175-205.

20. **Shutaywi M., Kachouie N. N.** Silhouette analysis for performance evaluation in machine learning with applications to clustering. *Entropy*. 2021, vol. 23, no. 6, pp. 759.

21. **Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau, D., Brucher M., Perrot M., Duchesnay E.** Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825-2830.