ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

УДК 004.891

АВТОМАТИЗИРОВАННЫЙ АНАЛИЗ ТЕКСТОВ ПРОГРАММ ПРИ ПОМОЩИ ПРЕДСТАВЛЕНИЙ НА ОСНОВЕ ЦЕПЕЙ МАРКОВА И МАШИН ЭКСТРЕМАЛЬНОГО ОБУЧЕНИЯ

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
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

Цифровизация экономики приводит к росту спроса на разработчиков программного обеспечения и, как следствие, к массовому характеру курсов по программированию. ***Целью данного исследования*** *является разработка модуля для анализа решений автоматически сгенерированных уникальных задач по программированию в системе «Цифровой ассистент преподавателя» (ЦАП), автоматизирующей массовый курс по программированию на языке Python в РТУ МИРЭА. Для векторизации текста программы предлагается выполнить построение дерева абстрактного синтаксиса, а затем преобразовать полученное дерево в цепь Маркова. Для классификации векторных представлений текстов программ предлагается применить машину экстремального обучения – вычислительно эффективную архитектуру искусственной нейронной сети. Разметка набора данных осуществляется алгоритмом иерархической кластеризации. Применение разработанного модуля позволяет автоматически определять способ решения сгенерированных с помощью ЦАП задач. Полученные сведения о способах решения могут использоваться преподавателями в течение семестра для выявления пробелов в знаниях и навыках учащихся. Статистика, полученная от классификаторов векторных представлений текстов программ, сообщается учащимся через веб-интерфейс системы ЦАП.*

Ключевые слова: *классификация текстов программ, анализ программного кода, алгоритм классификации, искусственная нейронная сеть, машина экстремального обучения, деревья абстрактного синтаксиса.*

DOI: 10.21667/1995-4565-2022-82-89-103

Введение

Массовый характер курсов по программированию, объясняемый высоким спросом на разработчиков программного обеспечения, приводит к повышению нагрузки на работников образовательных организаций. Для автоматизации таких рутинных видов преподавательской деятельности, как разработка задач для каждого студента, проверка решений студентов, был разработан ряд инструментов и программных систем [1-4]. Внедрение таких систем в образовательный процесс приводит к разгрузке преподавателей, но не к их самоустранению, позволяя творчески подходить к процессу преподавания, уделять меньше внимания рутинным действиям.

В современных условиях списывание со стороны учащихся становится нормой [3]. При автоматизации курсов по программированию необходимо уделять внимание проблеме поиска заимствований в решениях, присылаемых на проверку преподавателю или программной системе. Для решения данной проблемы были разработаны методы и алгоритмы поиска плагиата, основанные на применении алгоритмов кластеризации [5, 6], попарных сравнениях текстов программ и ранжировании результатов [7]. Также в работе [8] были предложены генераторы уникальных задач по программированию для полного предотвращения списывания со стороны студентов. При этом, как показано в [9], применение специализированных алгоритмов векторизации и кластеризации текстов программ к решениям автоматически сгенерированных задач позволяет выявить группы решений, в которых используются схожие подходы.

В данной статье рассматривается аналитический модуль системы «Цифровой ассистент преподавателя» (ЦАП) [3], генерирующей уникальные наборы задач по программированию разных типов для каждого студента [8] и автоматически проверяющей присланные решения. Аналитический модуль ЦАП включает алгоритмы кластеризации для выявления схожих способов решения в текстах программ [9] и алгоритмы классификации для быстрого определения подхода к решению задачи в момент загрузки нового решения в систему [10]. Задача выявления семантически схожих шаблонов в текстах программ не нова [11, 12], однако при разработке методов и алгоритмов анализа данных необходимо учитывать специфику предметной области, особенности наборов данных. Специализированные алгоритмы [9, 10], интегрированные в систему ЦАП, позволяют автоматически идентифицировать способ решения в присланной в ЦАП программе в случае её успешной проверки системой. Полученная статистика сообщается студенту через веб-интерфейс, используется преподавателями для устранения пробелов в знаниях студентов разных групп и для оценки качества генераторов задач, а также может быть использована для автоматического оценивания решений студентов в баллах.

Аналитический модуль системы «Цифровой ассистент преподавателя»

Система «Цифровой ассистент преподавателя» (ЦАП) автоматически генерирует задачи 11 различных типов для каждого студента курса [3, 9], генераторы решают задачу удовлетворения ограничениям по принципу generate and test [8]. Список типов задач, поддерживаемых системой ЦАП, представлен в таблице 1. На рисунке 1 приведены примеры сгенерированных задач на перевод математической нотации в код, относящихся к типу 3 (см. таблицу 1). Каждая программа, присылаемая студентом через веб-интерфейс ЦАП, сохраняется в базе данных (БД) системы, выделенный процесс проверяет полученные решения.

Таблица 1 – Описание типов задач, поддерживаемых системой ЦАП
Table 1 – Description of task types available in the DTA system

Тип задачи	Краткое описание
1	Реализовать функцию
2	Реализовать кусочную функцию
3	Реализовать итерационную функцию
4	Реализовать рекуррентную функцию
5	Реализовать функцию, оперирующую векторами
6	Реализовать функцию, вычисляющую дерево решений
7	Реализовать преобразование битовых полей
8	Реализовать разбор текстового формата
9	Реализовать конечный автомат Мили в виде класса
10	Реализовать преобразования табличных данных
11	Реализовать разбор двоичного формата данных

Реализовать итерационную функцию:

$$f(a,b,p)=\prod_{c=1}^{b}\sum_{j=1}^{a}(96(c^2-55j^3)-(8+j^3)^2)$$

а (a)

Реализовать итерационную функцию:

$$f(a,b,y)=\prod_{k=1}^{b}\sum_{c=1}^{a}\left(12(|c|)^2-\frac{\ln^6 k}{22}-\frac{y^3}{13}\right)$$

б (b)

Рисунок 1 – Примеры условий уникальных задач, относящихся к типу 3 (см. таблицу 1)
Figure 1 – Examples of formulations of unique tasks belonging to type 3 (see Table 1)

В конце семестра производится анализ присланных решений при помощи алгоритма кластеризации из аналитического модуля ЦАП для выявления схожих шаблонов в текстах программ с целью оценки качества генераторов задач и обучения классификаторов для следующего семестра. Обученные в конце предыдущего семестра классификаторы автоматически определяют способ решения по тексту программы для быстрой оценки пробелов в навыках и знаниях студентов преподавателями в течение семестра. Полученная от классификаторов статистика также сообщается студентам через веб-интерфейс ЦАП.

Последовательность применения рассматриваемых в данной статье алгоритмов векторизации текстов программ на основе цепей Маркова, а также алгоритмов кластеризации и классификации полученных векторных представлений показана на рисунке 2.

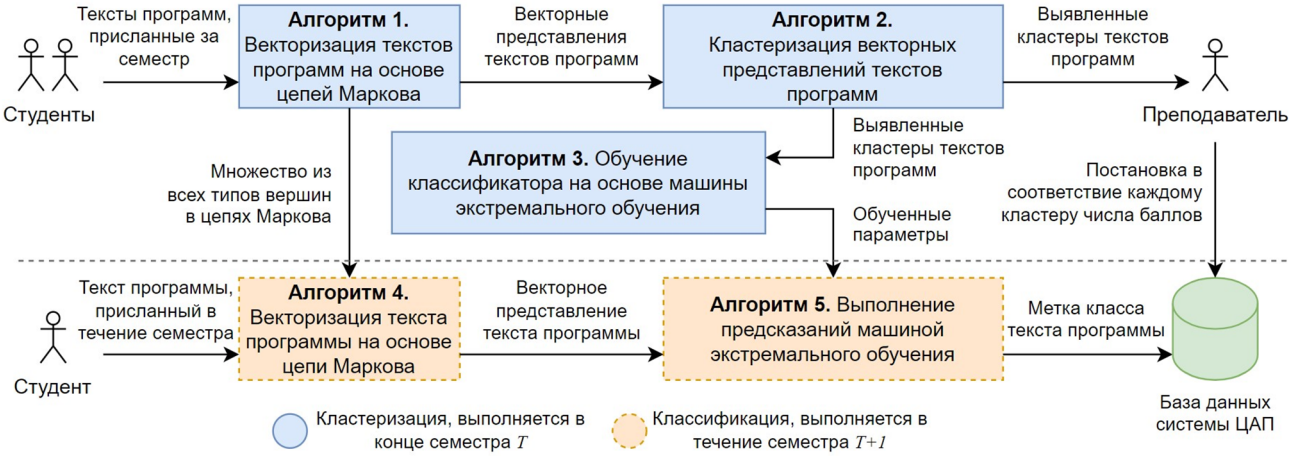


Рисунок 2 – Архитектура аналитического модуля ЦАП
Figure 2 – Architecture of the DTA system analytics module

Последовательное применение Алгоритма 1 и Алгоритма 2 (см. рисунок 2) позволило автоматически выявлять использованные подходы к решению уникальных автоматически сгенерированных задач в конце семестра РТУ МИРЭА с целью оценки качества генераторов задач и пробелов в навыках и знаниях учащихся массового курса по программированию на языке Python [9]. Применение Алгоритма 3 позволяет обучить алгоритмы классификации для каждого типа задачи, а применение Алгоритмов 4 и 5 позволяет выполнять быстрое определение способа решения в течение семестра в случае успешной проверки решения системой.

Перед началом нового семестра преподавателем может быть поставлено в соответствие каждой метке класса начисляемое количество баллов, что позволит системе ЦАП автоматически повышать или понижать оценку за присланное студентом решение в зависимости от кластера, к которому принадлежит способ решения, выбранный студентом.

Кластеризация текстов программ

Для применения большинства алгоритмов кластеризации и классификации необходимо, чтобы данные, подаваемые алгоритму на вход, были представлены в виде векторов. Широко используемым способом преобразования текстов на естественном языке к векторам является word2vec [13], для векторизации текстов программ применяется code2vec [14]. При реализации отображения $f: P \to \mathbb{R}^m$, выполняющего преобразование текстов программ из множе-

ства P в пространство вещественных чисел размерности m, в аналитическом модуле ЦАП необходимо учитывать особенности системы, генерирующей разные варианты задач разных типов.

Перед применением алгоритма кластеризации ЦАП преобразует множество P, включающее все полученные тексты программ, решающие автоматически сгенерированные задачи заданного типа, к векторам, принадлежащим $\mathbb{R}^m$, согласно Алгоритму 1. В процессе выполнения преобразования строк в векторы осуществляется построение дерева абстрактного синтаксиса (англ. Abstract Syntax Tree, AST) при помощи средств, доступных в стандартной библиотеке языка Python, графа переходов между состояниями цепи Маркова для AST, матрицы смежности для цепи Маркова [9]. При преобразовании AST текста программы к графу переходов цепи Маркова производится замена некоторых типов AST узлов, таблица 2 определяет список заменяемых узлов и узлов-заменителей для задач различных типов.

Выходом Алгоритма 1 является множество векторных представлений текстов программ V, каждый элемент которого принадлежит $\mathbb{R}^m$, и множество H, содержащее все различные типы вершин в графах переходов между состояниями цепей Маркова. Значение m в $\mathbb{R}^m$ определяется как квадрат числа типов вершин в множестве H, $m = |H|^2$.

Таблица 2 – Замены типов AST узлов при построении графов переходов цепей Маркова
Table 2 – Replacements of AST node types while building state transition graphs of Markov chains

Замены для типов задач с 1 по 11		Замены для типов задач с 6 по 11	
Заменяемое	Заменитель	Заменяемое	Заменитель
Constant, Attribute	Name	List, Tuple, Set	List
BoolOp, Call, UnaryOp, BinOp	Op	Lambda, JoinedStr, FormattedValue	Name
Lt, LtE	Less	Pass, Break, Continue	None
Gt, GtE	Greater	ExceptHandler	Try
AugAssign, AnAssign	Assign	IfExp	If
match_case, MatchStar, MatchAs, MatchOr, MatchSingleton, MatchSequence, MatchMapping, MatchClass	MatchValue	For, While, ListComp, SetComp, DictComp, GeneratorExp, comprehension	Loop
UAdd	Add		
USub	Sub		

Алгоритм 1 – Векторизация текстов программ на основе цепей Маркова
Algorithm 1 – Vectorization of program texts represented as Markov chains

Вход: $P = \{p_1, p_2, ..., p_n\}$ набор, содержащий n текстов программ.

1. **Определить** множество графов переходов цепей Маркова $G = \varnothing$.
2. **Определить** $R = \{\text{Import, Load, Store, alias, arguments, arg, Module, keyword}\}$.
3. **Для каждого** текста программы $p_i \in P$ выполнять:
4. **Построить** AST a_i для текста p_i средствами стандартной библиотеки Python.
5. **Удалить** из a_i вершины, принадлежащие множеству R.
6. **Заменить** вершины в a_i согласно таблице 2.
7. **Построить** граф переходов цепей Маркова g_i для a_i.
8. $G \leftarrow G \cup \{g_i\}$.
9. **Завершить цикл.**
10. **Определить** множество H, содержащее все различные типы вершин в графах из G.

11. **Определить** множество векторных представлений текстов программ $V = \varnothing$.
12. **Для каждого** графа переходов цепи Маркова $g_i \in G$ **выполнять:**
13. **Построить** матрицу смежности $\boldsymbol{m}_i \in \mathbb{R}^{|H|\times|H|}$ для взвешенного графа g_i.
14. **Преобразовать** матрицу $\boldsymbol{m}_i$ к вектору $\vec{v}_i \in \mathbb{R}^m$, где $m = |H|^2$.
15. $V \leftarrow V \cup \{\vec{v}_i\}$.
16. **Завершить цикл.**
17. **Возвратить** множество векторных представлений текстов программ V, множество H.

После преобразования набора P, содержащего тексты программ, решающих уникальные автоматически сгенерированные задачи определённого типа, к множеству векторных представлений текстов программ V, к V для иерархической кластеризации применяется Алгоритм 2. Сравнение каждой пары $\vec{v}_i$, $\vec{v}_j$ из V при построении матрицы попарных сходств M в Алгоритме 2 осуществляется при помощи меры сходства Йенсена – Шеннона (англ. Jensen-Shannon Divergence) [15, 9] согласно:

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

где $\vec{v}_i \in \mathbb{R}^m$ и $\vec{v}_j \in \mathbb{R}^m$ – векторы из множества V, m – размерность векторных представлений текстов программ, определяемая Алгоритмом 1, v_{ik} и v_{jk} – k-е компоненты сравниваемых векторов $\vec{v}_i$ и $\vec{v}_j$ соответственно.

Разделение векторов из множества V на кластеры производится агломеративным алгоритмом иерархической кластеризации на основе матрицы попарных сходств M, при разделении используется невзвешенный метод средней связи, в котором расстояние между двумя кластерами C_I и C_K равно усреднённому расстоянию между объектами, которые принадлежат данным кластерам:

$$R_{avg}(C_I, C_K) = \frac{1}{|C_I|\times|C_K|}\sum_{\vec{v}_i\in C_I}\sum_{\vec{v}_k\in C_K} d(\vec{v}_i,\vec{v}_k), \quad (2)$$

где C_I и C_K – кластеры, $|C_I|$ и $|C_K|$ – числа объектов, принадлежащих кластерам C_I и C_K, $\vec{v}_i$ и $\vec{v}_k$ – объекты, принадлежащие C_I и C_K соответственно, $d(\vec{v}_i,\vec{v}_k)$ – расстояние (1) между объектами $\vec{v}_i$ и $\vec{v}_k$, получаемое из матрицы попарных сходств M.

Алгоритм 2 также принимает на вход параметры $k_{\min} \in \mathbb{N}$ и $k_{\max} \in \mathbb{N}$, определяющие область поиска оптимального числа кластеров. Для оценки качества кластеризации используется индекс кластерного силуэта [16], значения индекса кластерного силуэта принадлежат вещественному промежутку [-1,1], значение 1 считается наилучшим, а значение -1 считается наихудшим. Алгоритм 2 возвращает разбиение на кластеры S_k с наибольшим индексом s_k.

Алгоритм 2 – Кластеризация векторных представлений текстов программ
Algorithm 2 – Clustering of vector representations of program texts

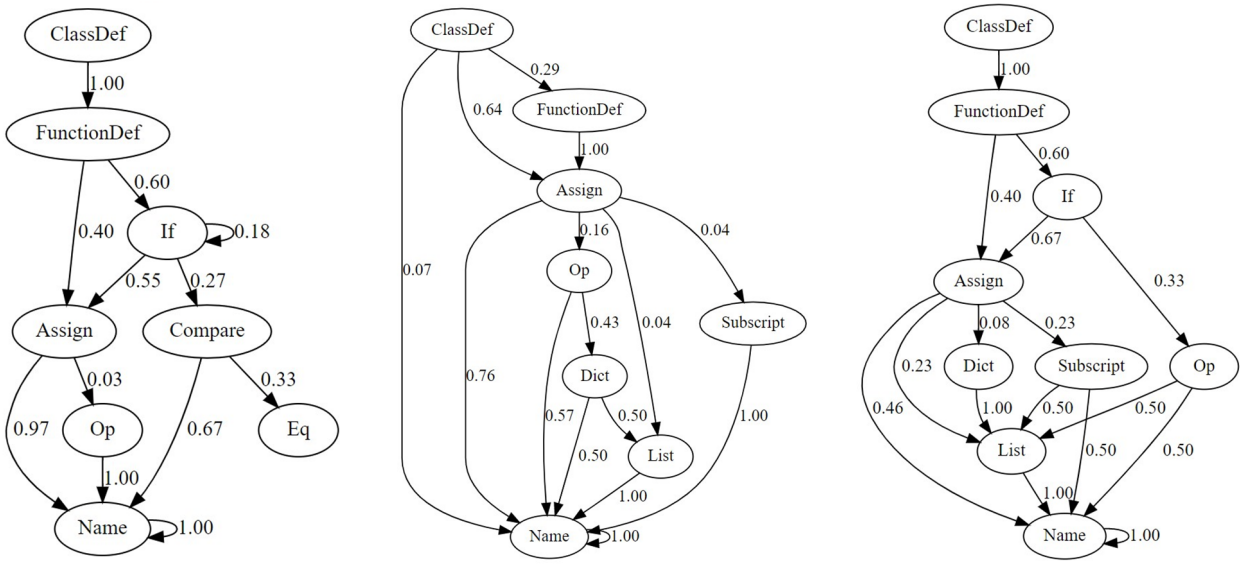
Вход: $V = \{\vec{v}_1,...,\vec{v}_i,...,\vec{v}_n\}$ набор, содержащий n векторных представлений текстов программ,
$k_{\min} \in \mathbb{N}$ – минимальное число кластеров,
$k_{\max} \in \mathbb{N}$ – максимальное число кластеров.

1. **Построить** матрицу попарных сходств $M \in \mathbb{R}^{|V|\times|V|}$ векторов из V по формуле (1).
2. **Установить** множество разбиений набора V на кластеры $S = \varnothing$.

3. **Для каждого** числа кластеров $k \in \mathbb{N}$ из промежутка $[k_{\min}, k_{\max}]$ **выполнять:**
4. **Разделить** V на k кластеров S_k по формуле (2) на основе матрицы сходств M.
5. **Вычислить** индекс кластерного силуэта s_k для разделения S_k.
6. $S \leftarrow S \cup \{S_k, s_k\}$.
7. **Завершить цикл.**
8. **Возвратить** разделение S_k с наибольшим индексом s_k из множества разделений S.

Лучшее разбиение векторных представлений текстов программ из множества V на кластеры затем используется для разметки набора данных для обучения классификатора. Примеры цепей Маркова для текстов программ, решающих уникальные задачи типа 9 (см. таблицу 1) и отнесённых к различным кластерам, представлены на рисунке 3.

Каждому вектору $\vec{v}_i \in V$, кодирующему текст программы p_i из множества всех текстов программ P, ставится в соответствие метка класса $y_i \in \mathbb{N}$, равная номеру кластера, к которому был отнесён вектор $\vec{v}_i$ в процессе выполнения Алгоритма 2. Результатом разметки набора данных V, разделённого Алгоритмом 2 на кластеры, является матрица $\boldsymbol{V}_L \in \mathbb{R}^{n \times m}$, строки которой кодируют векторы размерности m из множества V мощности n, и матрица $\boldsymbol{Y}_L \in \mathbb{R}^{n \times o}$, строки которой кодируют метки классов, представленные номерами кластеров, преобразованными к o-мерным векторам унитарным кодированием. Матрицы $\boldsymbol{V}_L$ и $\boldsymbol{Y}_L$, кодирующие размеченный набор данных, используются затем для обучения классификатора.



а (a) **б (b)** **в (c)**

Рисунок 3 – Примеры цепей Маркова для текстов программ, решающих различные задачи, относимые к типу 9 (см. таблицу 1) различающимися способами
Figure 3 – Examples of Markov chains for program texts that solve different problems belonging to type 9 (see Table 1) using varying approaches

Классификация текстов программ

Для классификации векторных представлений текстов программ в ЦАП предлагается использовать машину экстремального обучения. Машина экстремального обучения (англ. Extreme Learning Machine, ELM) [17] – это вычислительно эффективный алгоритм машинного обучения с учителем, искусственная нейронная сеть, обучающаяся без использования итеративных методов. Алгоритм ELM был разработан на основе таких алгоритмов, как нейронные

сети со случайными весами [18], случайные векторные функциональные сети [19], с помощью ELM был решён ряд прикладных задач [20–22]. Структура ELM представлена на рисунке 4.

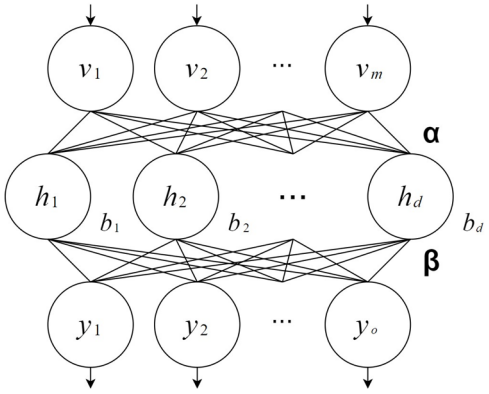


Рисунок 4 – Структура машины экстремального обучения (ELM)
Figure 4 – Extreme Learning Machine (ELM) structure

Как показано на рисунке 4, ELM является искусственной нейронной сетью с единственным скрытым слоем. ELM принимает на вход компоненты $v_1, v_2, \ldots, v_m$ вектора $\vec{v} \in \mathbb{R}^m$, кодирующего текст программы p, значение $m \in \mathbb{N}$ обозначает размерность входного вектора. Выходы $y_1, y_2, \ldots, y_o$ кодируют решения, принятые алгоритмом, причём в случае решения задачи классификации при помощи ELM выходной вектор $\vec{y}$ кодирует метку класса унитарным кодом. Скрытый слой содержит d скрытых нейронов $h_1, h_2, \ldots, h_d$. Связи между m нейронами входного и d нейронами скрытого слоя содержатся в матрице $\boldsymbol{\alpha} \in \mathbb{R}^{d \times m}$, связи между d нейронами скрытого и o нейронами выходного слоя содержит матрица $\boldsymbol{\beta} \in \mathbb{R}^{d \times o}$, значения $b_1, b_2, \ldots, b_d$ кодируют сдвиги для каждого из d нейронов скрытого слоя.

Алгоритм 3 описывает процесс обучения классификатора ELM (см. рисунок 4) на наборе данных, представленном матрицами $\boldsymbol{V}_L$ и $\boldsymbol{Y}_L$, содержащими векторные представления текстов программ и метки их классов, закодированные унитарным кодом.

Алгоритм 3 – Обучение машины экстремального обучения
Algorithm 3 – Extreme Learning Machine training process

Вход: $\mathbf{V}_L \in \mathbb{R}^{n \times m}$ – матрица, содержащая *n* строк, кодирующих *m*-мерные входные векторы,
$\mathbf{Y}_L \in \mathbb{R}^{n \times o}$ – матрица, содержащая *n* строк, кодирующих *o*-мерные выходные векторы,
$\gamma \in \mathbb{R}$ – параметр регуляризации,
$d \in \mathbb{N}$ – число скрытых нейронов,
σ – функция активации скрытого слоя.

1. **Инициализировать** $\boldsymbol{\alpha} \in \mathbb{R}^{d \times m}$ случайными числами с равномерным распределением.
2. **Инициализировать** $\vec{b} \in \mathbb{R}^d$ случайными числами с равномерным распределением.
3. **Вычислить** матрицу сдвигов $\mathbf{b} \in \mathbb{R}^{n \times d}$ клонированием вектора $\vec{b} \in \mathbb{R}^d$ *n* раз.
4. **Вычислить** выходную матрицу скрытого слоя $\mathbf{H} = \sigma(\mathbf{V}_L \boldsymbol{\alpha}^T + \mathbf{b})$.
5. **Инициализировать** диагональную матрицу $\mathbf{I} \in \mathbb{R}^{d \times d}$, элементы диагонали равны 1.
6. **Вычислить** псевдообратную матрицу $\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H} + \gamma \mathbf{I})^{-1} \mathbf{H}^T$.
7. **Вычислить** $\boldsymbol{\beta} \in \mathbb{R}^{d \times o}$ согласно $\boldsymbol{\beta} = \mathbf{H}^\dagger \mathbf{Y}_L$.
8. **Возвратить** входные веса $\boldsymbol{\alpha} \in \mathbb{R}^{d \times m}$, сдвиги $\vec{b} \in \mathbb{R}^d$, выходные веса $\boldsymbol{\beta} \in \mathbb{R}^{d \times o}$.

В Алгоритме 3 $\boldsymbol{\alpha}^T$ обозначает транспонированную матрицу весов соединений $\boldsymbol{\alpha}$ между входными и скрытыми нейронами, $\vec{b} = \{b_1, b_2, \ldots, b_d\}$ обозначает *d*-мерный вектор, кодирую-

щий веса сдвигов нейронов $h_1, h_2, \ldots, h_d$ скрытого слоя, $\mathbf{I}$ обозначает диагональную матрицу, элементы главной диагонали которой равны 1, $\mathbf{H}^T$ обозначает транспонированную выходную матрицу скрытого слоя $\mathbf{H}$, $\mathbf{H}^\dagger$ обозначает псевдообратную матрицу для матрицы $\mathbf{H}$. Скалярный параметр регуляризации γ, число скрытых нейронов d, функция активации скрытого слоя являются гиперпараметрами алгоритма ELM. В качестве σ часто используют сигмоидальную функцию активации нейронов скрытого слоя:

$$\sigma(x) = \frac{1}{1+e^{-x}}, \tag{3}$$

где x обозначает ячейку выходной матрицы скрытого слоя $\mathbf{H} \in \mathbb{R}^{n \times d}$ до активации.

Результатом обучения ELM согласно Алгоритму 3 являются матрицы весов соединений между нейронами, вектор сдвигов скрытого слоя.

Классификация текстов программ, которые не участвовали в процессе векторизации обучающего набора данных целиком согласно Алгоритму 1, кластеризации по Алгоритму 2 и обучения классификаторов по Алгоритму 3, необходима для реализации возможности быстрого определения способа решения в течение семестра системой ЦАП, в момент принятия текста программы модулем, ответственным за проверку решения.

Для определения метки класса текста программы *p*, не участвовавшей в процессах векторизации и кластеризации, описанных Алгоритмами 1 и 2, используется множество H, предварительно вычисленное в процессе выполнения Алгоритма 1 и сохранённое на диск. Множество H содержит все различные типы вершин, встречающиеся в графах переходов между состояниями цепей Маркова текстов программ, решающих задачи того же типа, что и программа *p*. Алгоритм 4 описывает процесс преобразования текста программы *p* в вектор $\vec{v}$.

Основное различие между Алгоритмом 1 и Алгоритмом 4 заключается в том, что Алгоритм 1 может быть применён только для векторизации набора текстов программ, решающих уникальные задачи определённого типа, что может быть полезно при решении задачи кластеризации с целью выявления наиболее часто используемых способов решения. Алгоритм 4, напротив, используется для векторизации одного текста программы на основе множества H, полученного от Алгоритма 1 и сохранённого на диск. Полученный от Алгоритма 4 вектор может быть подан на вход классификатору для быстрого определения способа решения.

Алгоритм 4 – Векторизация текста программы *p* на основе цепи Маркова
Algorithm 4 – Vectorization of program text p based on Markov chain

Вход: *p* – текст программы,
H – множество, содержащее все различные типы вершин, встречающихся в графах переходов цепей Маркова текстов программ, решающих задачи того же типа, что и программа *p*, данное множество получено в результате выполнения Алгоритма 1.

1. **Определить** *R* = {Import, Load, Store, alias, arguments, arg, Module, keyword}.
2. **Построить** AST *a* для текста *p* средствами стандартной библиотеки Python.
3. **Удалить** из *a* вершины, принадлежащие множеству *R*.
4. **Заменить** узлы в *a* согласно таблице 2.
5. **Построить** граф переходов цепи Маркова *g* для дерева *a*.
6. **Построить** матрицу смежности $\boldsymbol{m} \in \mathbb{R}^{|H| \times |H|}$ для взвешенного графа *g*.
7. **Преобразовать** матрицу $\boldsymbol{m}$ к вектору $\vec{v} \in \mathbb{R}^{|H|^2}$.
8. **Возвратить** векторное представление $\vec{v}$ для текста программы $\vec{v}$.

Алгоритм 5 описывает процесс классификации векторных представлений текстов программ машиной экстремального обучения, используя веса и сдвиги, полученные от Алгоритма 3 и предварительно сохранённые на диск. Результатом применения Алгоритма 5 к набору данных $\mathbf{V}_T \in \mathbb{R}^{n \times m}$, содержащему тексты программ, которые классификатор ранее не

видел, является матрица $\mathbf{Y}_T \in \mathbb{R}^{n \times o}$, кодирующая *o*-мерные предсказанные метки классов, закодированные унитарным кодом. В случае, если необходимо определить метку класса для единственной программы, число строк n в матрицах $\mathbf{V}_T$ и $\mathbf{Y}_T$ полагается равным 1, число столбцов m в матрице $\mathbf{V}_T \in \mathbb{R}^{n \times m}$ обозначает размерность векторного представления текста программы, число столбцов o в матрице $\mathbf{Y}_T \in \mathbb{R}^{n \times o}$ равно числу меток классов, поскольку используется унитарное кодирование.

Алгоритм 5 – Выполнение предсказаний машиной экстремального обучения
Algorithm 5 – Prediction making with Extreme Learning Machine

Вход: $\mathbf{V}_T \in \mathbb{R}^{n \times m}$ – матрица, содержащая *n* строк, кодирующих *m*-мерные входные векторы,

$\gamma \in \mathbb{R}$ – параметр регуляризации,

$d \in \mathbb{N}$ – число скрытых нейронов,

σ – функция активации скрытого слоя,

$\boldsymbol{\alpha} \in \mathbb{R}^{d \times m}$ – входные веса,

$\vec{b} \in \mathbb{R}^{d}$ – сдвиги скрытого слоя,

$\boldsymbol{\beta} \in \mathbb{R}^{d \times o}$ – выходные веса.

1. **Вычислить** матрицу сдвигов $\mathbf{b} \in \mathbb{R}^{n \times d}$ клонированием вектора $\vec{b} \in \mathbb{R}^{d}$ *n* раз.
2. **Вычислить** выходную матрицу скрытого слоя $\mathbf{H} = \sigma(\mathbf{X}_T \boldsymbol{\alpha}^T + \mathbf{b})$.
3. **Вычислить** матрицу, кодирующую выходные векторы $\mathbf{Y}_T = \mathbf{H}\boldsymbol{\beta}$, причём $\mathbf{Y}_T \in \mathbb{R}^{n \times o}$.
4. **Возвратить** матрицу $\mathbf{Y}_T$ из *n* строк, кодирующих *o*-мерные выходные векторы.

Вычислительный эксперимент

Поскольку в системе ЦАП используются генераторы задач 11 различных типов, перечень которых приведён в таблице 1, на вход модулю, реализующему Алгоритм 1, были поданы 11 наборов данных, содержащих тексты программ, сгруппированные по типу решаемой задачи. Каждый из наборов данных содержал более тысячи текстов программ (см. рисунок 5). Результаты векторизации каждого из 11 наборов данных Алгоритмом 1 были поданы на вход модулю, реализующему Алгоритм 2, с помощью которого каждый из наборов был разбит на кластеры. Параметры k_{min} и $k_{\max}$ в Алгоритме 2 были установлены равными 10 и 40 для всех типов задач, при этом в качестве лучших были выбраны разбиения с наибольшим индексом кластерного силуэта [9].

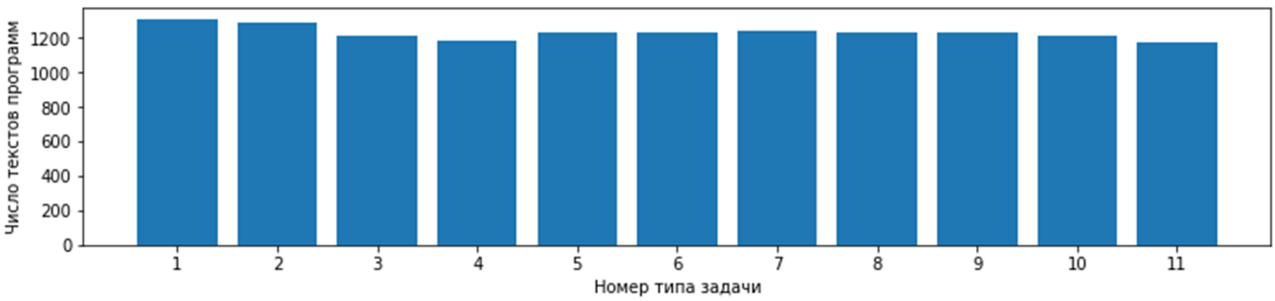


Рисунок 5 – Число текстов программ, решающих задачи разных типов (см. таблицу 1).
Figure 5 – The number of program texts solving tasks of different types (see Table 1)

При обучении классификаторов для обеспечения возможности быстрого определения способа решения по тексту программы необходимо, чтобы каждый из способов решения в обучающем наборе встречался достаточно часто для выявления алгоритмом классификации закономерностей в данных. Поэтому перед передачей размеченного алгоритмом кластеризации набора данных на вход модулю, реализующему Алгоритм 3, из набора удаляются кла-

стеры, содержащие менее 10 объектов, число 10 было выбрано экспериментально. Каждому из оставшихся кластеров в соответствие была поставлена метка класса, соответствующая номеру кластера. Полученные числа кластеров представлены на рисунке 6.

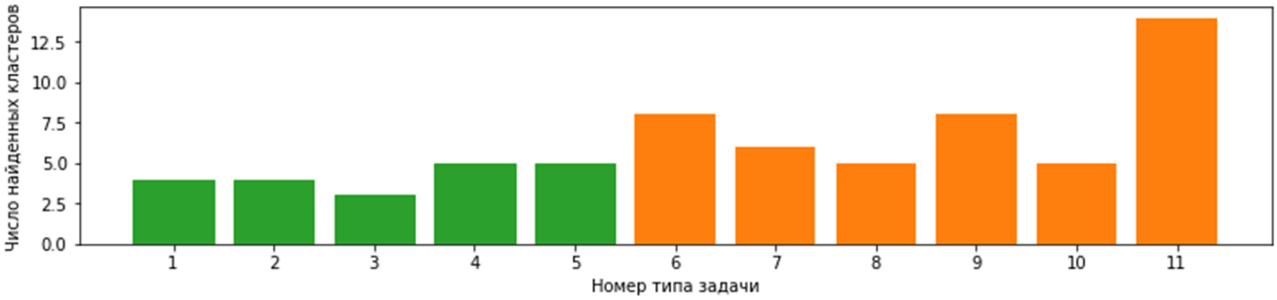


Рисунок 6 – Число выявленных кластеров для каждого из 11 наборов (см. таблицу 1).
Figure 6 – The number of clusters discovered for each of 11 datasets (see Table 1)

Зелёным на рисунке 6 обозначены типы задач, для которых осуществляются базовые замены AST узлов при построении цепей Маркова, оранжевым обозначены типы задач, для которых осуществляются также расширенные замены, такие типы узлов AST, как For, While и др., заменяются на Loop, такие узлы, как List, Tuple, Set заменяются на List (см. таблицу 2). Это позволяет объединять в один кластер решения, использующие разные формы записи циклов и различные структуры данных для реализации одного и того же подхода при решении уникальной задачи. Без дополнительных замен для задач с 6 по 11 число кластеров, включающих в себя менее 10 объектов, значительно возрастает. Типы задач в ЦАП ранжируются в порядке увеличения сложности их решения. Согласно рисунку 6, с увеличением сложности задачи также растёт вариативность часто применяемых подходов к их решению. Решения задачи типа 11 наиболее вариативны – присланные студентами тексты программ, реализующие разбор двоичного формата данных (см. таблицу 1), содержали как решения при помощи классов с внутренним состоянием, так и с помощью функций, лямбда-выражений, словарей, декораторов.

Размеченные наборы векторных представлений текстов программ с гарантией, что каждый класс содержит как минимум 10 объектов, были поданы на вход модулю, реализующему Алгоритм 3. Для оценки качества классификаторов ELM каждый из 11 наборов был разбит на обучающий и тестовый наборы, размеры наборов и числа классов приведены в таблице 3.

В качестве функции активации скрытого слоя σ в Алгоритме 3 использовалась сигмоидальная функция (3), а такие параметры, как число нейронов скрытого слоя d и коэффициент регуляризации γ, были выбраны поиском по сетке. Значения $d \in \mathbb{N}$ выбирались из промежутка [25, 250] с шагом 25, значения γ выбирались из множества $\{10^{-6}, 10^{-3}, 1\}$. Выбранные параметры для 11 классификаторов приведены в таблице 4.

Таблица 3 – Характеристика 11 наборов данных, подготовленных для обучения ELM
Table 3 – Properties of 11 datasets prepared for ELM-based classifier training

Тип задачи	1	2	3	4	5	6	7	8	9	10	11
Размер обучающего набора	918	903	847	830	863	862	871	860	861	847	821
Размер тестового набора	394	387	364	356	370	370	374	369	370	364	353
Число классов	4	4	3	5	5	8	6	5	8	5	14

Таблица 4 – Параметры классификаторов ELM для 11 наборов текстов программ
Table 4 – Parameters of ELM-based classifiers for 11 datasets containing program texts

Тип задачи	1	2	3	4	5	6	7	8	9	10	11
d	25	125	150	75	25	250	50	100	225	200	225
γ	10^{-6}	10^{-3}	10^{-6}	10^{-6}	10^{-6}	1	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-6}

В процессе оценки качества классификаторов каждый из 11 наборов данных был разбит на обучающий и тестовый наборы (см. таблицу 3) $k = 50$ раз, каждый раз были вычислены такие меры качества классификаторов, как Accuracy, Precision, Recall, F1 measure. Результаты оценки качества классификаторов на основе ELM приведены в таблице 5.

Таблица 5 – Оценка качества классификаторов ELM на 11 наборов данных
Table 5 – Quality assessment of ELM-based classifiers for 11 datasets

Тип задачи	1	2	3	4	5	6	7	8	9	10	11
Accuracy	100.00	99.82	99.93	99.95	99.98	98.67	99.89	97.05	99.35	98.23	98.24
Precision	100.00	99.72	99.95	99.30	99.76	91.40	99.54	90.08	98.26	88.95	95.66
Recall	100.00	99.82	99.96	99.98	99.99	96.07	99.25	96.18	98.37	94.62	97.74
F1 measure	100.00	99.77	99.95	99.59	99.83	92.84	99.34	92.14	98.24	90.75	96.30

Как видно из таблицы 5, классификаторы ELM достигают высокой точности классификации на тестовых данных, которые не использовались для обучения ELM. В таблице 6 приведено время, затрачиваемое на выполнение предсказаний ELM для тестовых наборов данных, числа объектов и классов в наборах для различных типов задач приведены в таблице 3.

Таблица 6 – Время, затрачиваемое на выполнение предсказаний ELM
Table 6 – Time elapsed while making predictions using ELM

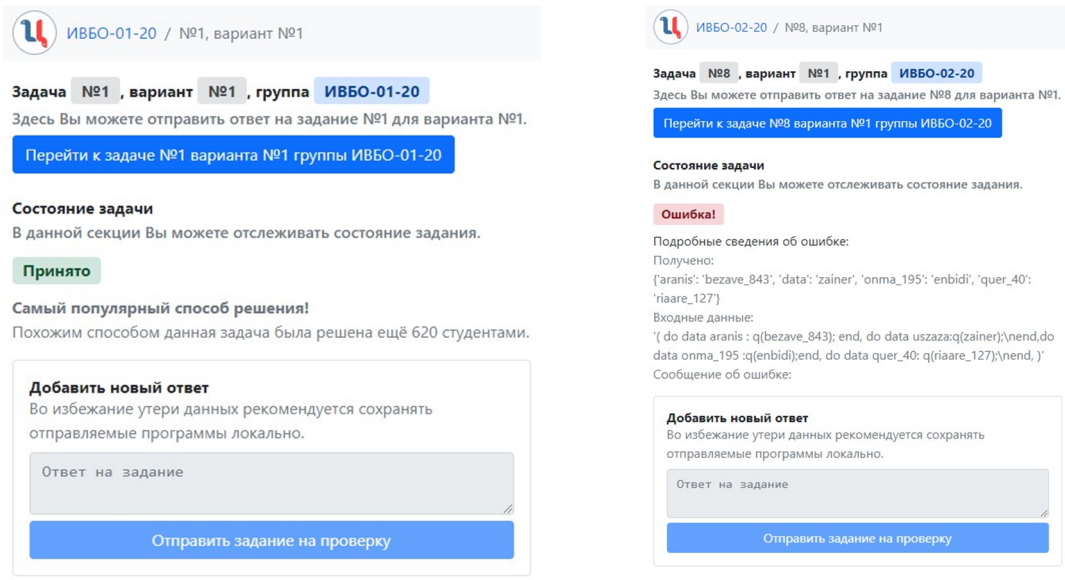
Тип задачи	1	2	3	4	5	6	7	8	9	10	11
Мат. ожидание, мс.	1.84	6.93	6.98	5.04	2.47	12.90	3.55	6.63	10.19	13.29	12.56
СКО, мс.	0.11	0.27	0.47	0.96	0.24	0.50	0.41	0.29	0.26	2.69	0.35

Из таблицы 6 видно, что ELM затрачивает всего несколько миллисекунд для определения меток классов нескольких сотен объектов (см. таблицу 3) в случае, если задача классификации характеризуется малым числом классов. При увеличенном числе классов в задачах типа 6, 9, 11 время классификации сотен объектов увеличивается до 10 миллисекунд. Алгоритм ELM был реализован при помощи языка программирования Python версии 3.10 и библиотеки numpy [23], характеристики ПК, на котором выполнялся эксперимент, приведены в таблице 7.

Таблица 7 – Характеристики ПК, на котором выполнялся эксперимент
Table 7 – Properties of the PC where experiments were conducted

Параметр	Процессор	Память
Значение	Intel® Core™ i3-7100U (2.40 ГГц, 2 ядра)	20 Гб DDR4 (2133 МГц)

Выходом Алгоритма 3 является тройка, содержащая матрицу весов соединений между входными и скрытыми нейронами, вектор сдвигов скрытого слоя, матрицу весов соединений между скрытыми и выходными нейронами. Значения этих трёх параметров сохраняются на диск и загружаются в оперативную память при необходимости выполнения предсказаний по Алгоритму 5. Для преобразования текста программы, не принадлежащего к набору данных, описанному в таблице, 3 и не участвовавшего в процессах векторизации и кластеризации, используется Алгоритм 4. В качестве параметра H в Алгоритм 4 передаётся список всех различных типов вершин, полученный при выполнении Алгоритма 1 и предварительно сохранённый на диск. При помощи последовательного применения модулей, реализующих Алгоритм 4 и Алгоритм 5 к новому тексту программы, присланному через веб-интерфейс ЦАП на проверку системе, способ решения автоматически сгенерированной задачи может быть быстро определён в момент проверки. Полученная при анализе текста программы статистика показывается студентам в веб-интерфейсе ЦАП после проверки и успешного принятия решения системой (см. рисунок 7).



а (a) **б (b)**

Рисунок 7 – Страница отправки решения ЦАП: а – задача принята; б – сообщение об ошибке
Figure 7 – DTA task submission page: a – the task is accepted; b – error message

Заключение

В статье был рассмотрен подход к автоматизированному анализу текстов программ, представленных цепями Маркова, при помощи машин экстремального обучения. Подход используется в аналитическом модуле системы ЦАП для выявления наиболее распространённых подходов к программному решению уникальных автоматически сгенерированных задач. Наиболее распространённые подходы к решению могут быть выявлены как в конце семестра посредством применения алгоритма векторизации (см. Алгоритм 1) и алгоритма кластеризации (см. Алгоритм 2) ко всем текстам программ, принятым системой [9], так и в течение семестра посредством предварительного обучения классификатора ELM согласно Алгоритму 3 и быстрого определения способа решения при помощи Алгоритма 4 для векторизации и Алгоритма 5 для классификации [10]. Разработанный аналитический модуль может применяться для анализа данных в системах, автоматизирующих массовые курсы по программированию, где используются генераторы уникальных задач.

Библиографический список

1. **Симуни М. Л., Соловьёв А. Ю., Шайтан В. И.** Автоматизированная проверка задач в курсе «Функциональное программирование» // Современные информационные технологии и ИТ-образование. 2016. Т. 12. № 3-1. С. 90-96.

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

3. **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.

4. **Tiam-Lee T.J., Sumi K.** Procedural generation of programming exercises with guides based on the student's emotion // 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2018. pp. 1465-1470.

5. **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.

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. **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.

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

9. **Демидова Л. А., Советов П. Н., Горчаков А. В**. Кластеризация представлений текстов программ на основе цепей Маркова // Вестник Рязанского государственного радиотехнического университета. 2022. № 81. С. 51-64.

10. **Demidova L. A., Gorchakov A. V.** Classification of Program Texts Represented as Markov Chains with Biology-Inspired Algorithms-Enhanced Extreme Learning Machines // Algorithms. 2022, vol. 15, no. 9, 329 p.

11. **Allamanis M., Sutton C.** Mining idioms from source code // Proceedings of the 22nd ACM Sigsoft International Symposium on Foundations of Software Engineering. 2014. pp. 472-483.

12. **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.

13. **Mikolov T., Chen K., Corrado G., Dean J.** Efficient estimation of word representations in vector space // arXiv preprint, arXiv:1301.3781. 2013.

14. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code // Proceedings of the ACM on Programming Languages. 2019, vol. 3, no. POPL, pp. 1-29.

15. **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. pp. 56-63.

16. **Shahapure K. R., Nicholas C.** Cluster quality analysis using silhouette score //2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA). IEEE, 2020, pp. 747-748.

17. **Huang G. B., Zhu Q. Y., Siew C. K.** Extreme Learning machine: Theory and applications // Neurocomputing, 2006, vol. 70, pp. 489-501.

18. **Schmidt W. F., Kraaijveld M. A., Duin R. P.** Feedforward Neural Networks with Random Weights // In Proceedings of the 11th IAPR International Conference on Pattern Recognition, Pattern Recognition Methodology and Systems, The Hague, The Netherlands, 30 August–3 September 1992; IEEE: Pistacaway, NJ, USA, 1992; vol. 2, pp. 1-4.

19. **Pao Y. H., Takefji Y.** Functional-link net computing: Theory, system architecture, and functionalities // Computer. 1992, vol. 25, pp. 76-79.

20. **Cheng C., Tay W. P., Huang G. B.** Extreme Learning Machines for Intrusion Detection // In Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, 10-15 June 2012; IEEE: Pistacaway, NJ, USA, 2012; pp. 1-8.

21. **Liu Y., Loh H. T., Tor S. B.** Comparison of Extreme Learning Machine with Support Vector Machine for Text Classification // In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Innovations in Applied Artificial Intelligence, Bari, Italy, 22-24 June 2005; Springer: Berlin, Germany, 2005; pp. 390-399.

22. **Демидова Л. А., Горчаков А. В.** Применение биоинспирированных алгоритмов глобальной оптимизации для повышения точности прогнозов компактных машин экстремального обучения // Russian Technological Journal, 2022. Т. 10, № 2, С. 59-74.

23. **Harris C. R., Millman K. J., van der Walt S. J. et al.** Array programming with NumPy // Nature, 2020, vol. 585, pp. 357-362.

UDC 004.891

AUTOMATED PROGRAM TEXT ANALYSIS USING REPRESENTATIONS BASED ON MARKOV CHAINS AND EXTREME LEARNING MACHINES

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
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

The digitalization of the economy leads to an increase in demand for software developers, and, as a result, to the massive nature of programming courses. ***The aim is*** *to develop a module for analyzing solutions to automatically generated unique programming tasks in the Digital Teaching Assistant (DTA) system, which automates a massive Python programming course at RTU MIREA. To vectorize a program text, it is proposed to build an abstract syntax tree, and then convert the resulting tree into a Markov chain. To classify vector representations of program texts, it is proposed to use an extreme learning machine - a computationally efficient architecture of an artificial neural network. The labeling of the data set is carried out by the hierarchical clustering algorithm. The use of the developed module made it possible to automate the determination of methods for solving automatically generated problems in real time in programs sent to the DTA. The obtained information about the methods of solution can be used by programming instructors during the semester to identify gaps in the knowledge and skills of students. Statistics obtained from classifiers of vector representations of program texts are reported to students through the web interface of the DTA system.*

Key words: *classification of program texts, code analysis, classification algorithm, artificial neural network, extreme learning machine, abstract syntax trees.*

DOI: 10.21667/1995-4565-2022-82-89-103

References

1. **Simuni M. L., Soloviov A. Y., Shaitan V. I.** Automated problem checking for functional programming course. *Modern Information Technologies and IT-Education*. 2016, vol. 12, no. 3-1. pp. 90-96.

2. **Andrianova E. G., Demidova L. A., Sovetov P. N.** «Cifrovoj assistent prepodavatelya» v massovom professional'nom obuchenii dlya cifrovoj ekonomiki. *Russian Technological Journal*. 2022, vol. 10, no 3, pp. 7-23. (in Russian).

3. **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.

4. **Tiam-Lee T.J., Sumi K.** Procedural generation of programming exercises with guides based on the student's emotion. *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2018. pp. 1465-1470.

5. **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.

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. **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.

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

9. **Demidova L. A., Sovietov P. N., Gorchakov A. V.** Klasterizaciya predstavlenij tekstov programm na osnove cepej Markova. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta*. 2022, no. 81, pp. 51-64. (in Russian).

10. **Demidova L. A., Gorchakov A. V.** Classification of Program Texts Represented as Markov Chains with Biology-Inspired Algorithms-Enhanced Extreme Learning Machines. *Algorithms*. 2022, vol. 15, no. 9, 329 p.

11. **Allamanis M., Sutton C.** Mining idioms from source code. *Proceedings of the 22nd ACM Sigsoft International Symposium on Foundations of Software Engineering*. 2014. pp. 472-483.

12. **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.

13. **Mikolov T., Chen K., Corrado G., Dean J.** *Efficient estimation of word representations in vector space*. arXiv preprint, arXiv:1301.3781. 2013.

14. **Alon U., Zilberstein M., Levy O., Yahav E.** code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*. 2019, vol. 3, no. POPL, pp. 1-29.

15. **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, pp. 56-63.

16. **Shahapure K. R., Nicholas C.** Cluster quality analysis using silhouette score. *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2020, pp. 747-748.

17. **Huang G. B., Zhu Q. Y., Siew C. K.** Extreme Learning machine: Theory and applications. *Neurocomputing.* 2006, vol. 70, pp. 489-501.

18. **Schmidt W. F., Kraaijveld M. A., Duin R. P.** Feedforward Neural Networks with Random Weights. In Proceedings of the 11th IAPR International Conference on Pattern Recognition, Pattern Recognition Methodology and Systems, The Hague, The Netherlands, 30 August–3 September 1992; *IEEE: Pistacaway, NJ, USA*, 1992; vol. 2, pp. 1-4.

19. **Pao Y. H., Takefji Y.** Functional-link net computing: Theory, system architecture, and functionalities. *Computer.* 1992, vol. 25, pp. 76-79.

20. **Cheng C., Tay W. P., Huang G. B.** Extreme Learning Machines for Intrusion Detection. *In Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, 10-15 June 2012; IEEE: Pistacaway*, NJ, USA. 2012; pp. 1-8.

21. **Liu Y., Loh H. T., Tor S. B.** Comparison of Extreme Learning Machine with Support Vector Machine for Text Classification. *In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Innovations in Applied Artificial Intelligence, Bari, Italy, 22-24 June 2005; Springer: Berlin, Germany*. 2005; pp. 390-399.

22. **Demidova L. A., Gorchakov A. V.** Primenenie bioinspirirovannyh algoritmov global'noj optimizacii dlya povysheniya tochnosti prognozov kompaktnyh mashin ekstremal'nogo obucheniya. *Russian Technological Journal,* 2022, vol. 10, no. 2, pp. 59-74. (in Russian).

23. **Harris C. R., Millman K. J., van der Walt S. J. et al.** Array programming with NumPy. *Nature. 2020*, vol. 585, pp. 357-362.