31. Efficient backprop / Y. A. LeCun, L. Bottou, G. B. Orr, K. R. Muller // Neural networks: Tricks of the trade. Berlin, Heidelberg: Springer, 2012. P. 9–48.

32. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks // Proc. of the thirteenth intern. conf. on artificial intelligence and statistics. Sardinia: Domus de Maria, 2010. P. 249–256.

33. The loss surfaces of multilayer networks / A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, Y. LeCun // Artificial Intelligence and Statistics. 2015. № 2. P. 192–204.

34. Saxe A. M., McClelland J. L., Ganguli S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks // arXiv preprint arXiv:1312.6120. 2013. URL: https://arxiv.org/abs/1312.6120 (дата обращения: 14.03.19).

35. Mishkin D., Matas J. All you need is a good init // arXiv preprint arXiv:1511.06422. 2015. URL: https://arxiv.org/abs/1511.06422 (дата обращения: 21.02.19).

A. S. Perkov, T. R. Zhangirov, A. A. Liss
Saint Petersburg Electrotechnical University «LETI»

N. Yu. Grigoryeva
Saint Petersburg SRCES RAS

L. V. Chistyakova
Center for Culture Collection of Microorganisms SPSU

COMPARISON OF NEURAL NETWORK TRAINING METHODS IN THE CLASSIFICATION PROBLEM

Based on the analysis of a series of feedforward artificial neural networks, a method has been developed for determining the optimal neural network architecture for the task of classifying cyanobacterial strains according to the fluorescence spectra. The analysis of six gradient methods of training neural networks and their parameters was carried out, the optimal number of neurons in the hidden layer for neural networks trained by each of the methods was found, various methods of initializing the weights of neurons and methods for splitting the initial sample into training, test and control samples were evaluated. The choice of the optimal architecture was carried out on the basis of the classification results, namely, on the basis of the classification accuracy graphs and the classification error graphs. The research was conducted on the example of recognition of 16 classes, representing 16 strains of cyanobacteria. A number of shortcomings were identified in the method of testing feedforward neural networks and directions for additional researches of neural networks for classification in terms of extending the testing methodology of their internal logic were determined.

Neural networks, training methods, machine learning, classification problems, initialization methods

УДК 519.688

И. А. Посов, В. Е. Допира
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Методы поиска плагиата в кодах программ

Изучены такие виды представления данных для поиска плагиата в кодах программ, как текст без преобразований, n-граммы и токены. На языке программирования Python реализовано разбиение текстов программ из массива решений студентов на токены. Сформулированы требования к алгоритмам обнаружения плагиата. Проведен обзор метрик для обнаружения плагиата в текстах программ. Выделены преимущества и недостатки для каждой метрики. Сравнение проведено по критериям: время, память, вероятность найти пару похожих программ, вероятность того, что найденная пара будет действительно похожа. После сравнения метрик: численных значений атрибутов, наибольшей общей подпоследовательности, расстояния Жаккара, расстояния Левенштейна и расстояния Колмогорова, между собой для реализации выбран расчет расстояния Левенштейна. На языке программирования Python реализован алгоритм расчета расстояния Левенштейна для списка токенов. Полученные результаты показывают, насколько тексты программ похожи между собой.

Плагиат, метрики для обнаружения плагиата, токен

Цифровой контент все чаще подвергается копированию. Люди цитируют друг друга, также соавторство порождает увеличение копий работ. Соответственно, возникает необходимость установления авторских прав на интеллектуальный труд, а также проверки новых работ на плагиат.

Под плагиатом понимается «умышленное присвоение авторства чужого произведения или использование в своих трудах чужого произведения без ссылки на автора» [1].

В учебной деятельности проблема обнаружения плагиата очень актуальна, потому что многие студенты копируют решения у однокурсников или берут из Интернета. В иностранных вузах контроль за списыванием студентов ведется постоянно.

Целью является реализация программного модуля, осуществляющего поиск плагиата в кодах программ. Объект исследования – методы обнаружения плагиата, а предмет – расстояние между текстами программ.

Поставлены следующие задачи:

1. Выделить способы представления данных для поиска плагиата в кодах программ.

2. Сформулировать требования к алгоритмам обнаружения плагиата.

3. Исследовать существующие методы поиска плагиата.

4. Выделить достоинства и недостатки существующих методов.

5. Реализовать алгоритм обнаружения плагиата в кодах программ, написанных на языке Octave.

Системы обнаружения плагиата также применяются в таких задачах, как указание точных источников заимствования информации, повышение точности поисковой выдачи, кластеризация кода по выбранным параметрам.

Нетривиальным будет нахождение такого решения, которое гарантированно обнаружит копирование чужого кода, включая небольшие участки в большом объеме данных. Необходимо также, чтобы код программы, в которым заменены имена переменных или их порядок, обнаруживался как скопированный при наличии совпадений. Высокие требования, безусловно, усложняют задачу, поэтому возникает необходимость изучить существующие методы обнаружения плагиата и проанализировать их, выделив преимущества и недостатки. А создание удобного программного инструмента для обнаружения плагиата представляет собой актуальную задачу.

Для выявления плагиата основным методом является расчет расстояния между текстами. Имеющиеся на рынке решения не позволяют работать с большим количеством коротких программ, не предоставляют пользователям удобного интерфейса и зачастую их функционал сильно ограничен.

Обзор алгоритмов обнаружения плагиата в кодах программ.

1. *Виды представления исходных кодов программ.* Входными данными модуля проверки на плагиат является текст программы. Для анализа на плагиат могут быть использованы исходный, объектный или исполняемый код программ. Был выбран исходный код, так как в нем содержится больше информации, характеризующей стилистические особенности конкретного автора. Первым шагом для реализации модуля обнаружения плагиата является выбор способа представления данных в кодах программ.

1.1. Исходный текст без преобразований. Можно рассматривать исходный код без преобразований. Такой метод представления данных крайне неэффективен, так как при анализе численных выражений признаков программы у разных программ могут быть получены близкие значения.

1.2. Элементы n-мерного пространства: N-граммы. N-грамма – это непрерывная подстрока из n элементов. Тексты программ разбиваются на N-граммы, и составляются списки слов, содержащие эти подстроки. Далее последовательно перебираются и сравниваются списки слов, которые содержат конкретные n-граммы. Если два слова совпадают с погрешностью, то, вероятно, для них найдется хотя бы одна общая n-грамма. Чтобы определить, насколько похожи слова в двух программах, следует задать метрику.

Параметр разбиения n выбирается пользователем. Выбор большого значения N ведет к тому, что слова длины, меньшей, чем n, могут быть не найдены.

Например, есть текст: «function [X, Y] = = task3(A, B)». После разделения на подстроки длины 4: «func», «unct», «ncti», «ctio», «tion» и т. д.

Можно улушить алгоритм хешированием по сигнатуре [2]. В качетве хеша используется группа символов из алфавита. Бит 1 на позиции i в хеше означает, что в исходном слове присутствует n-грамма из i-й группы алфавита. Такие хеши вычисляются для каждой программы. Далее в списки группируются слова, которые имеют одинаковое значение хеша. Составляется таблица соответствия хеша и списка слов. Алгоритм находит сначала списки слов, которые могут удовлетворять запросу, а затем ищет сами слова. Похожие слова будут располагаться в смежных ячейках, что позволяет осуществлять поиск эффективнее, без многократного чтения из разных позиций.

Таблица 1

Имя токена	Описание	Примеры
Идентификаторы	Имена переменных или функций, которые задает программист	*x*, tokenizer, digit
Ключевые слова	Имена индентификаторов, которые имеют специальное значение для компиляторов на языке программирования (не могут совпадать с именами переменных)	function, if, for
Разделители	Знаки пунктуации и парные разделители	{, (, ;
Операторы	Набор команд	+, =, <
Литералы	Фиксированные значения некоторого типа данных	True, 1, String
Комментарии	Строка или блок, содержащие пояснения к коду	# решаем систему уравнений

Значение хеша нечувствительно к порядку символов, удалению или вставке одного символа. При этом хеширование по сигнатуре очень чувствительно к выбору параметров и хеш-функции.

1.3. Список токенов. Токенизация – это процесс преобразования текста в последовательность строк (лексических единиц) с определенным значением (токенов) [3].

Токены состоят из имени токена и его значения, которое не обязательно может присутствовать. Имя токена представляет собой категорию лексической единицы. Выделяют следующие общие имена (см. табл. 1).

Токенизация позволяет сохранять существенные и игнорировать легко модифицируемые детали кода программы.

2. *Требования к алгоритму.* При сравнении алгоритмов выдвигаются критерии:

2.1. Сложность (оценка времени) алгоритмов поиска плагиата.

2.2. Оценка памяти алгоритма.

2.3. С какой вероятностью конкретная пара похожих программ будет найдена алгоритмом.

2.4. С какой вероятностью пара программ, определенные алгоритмом как похожие, действительно является похожей.

Первые два критерия показывают затраты на реализацию алгоритма. Вторые два связаны непосредственно с обнаружением плагиата в текстах программ и показывают вероятность найти правильные решения.

Для того чтобы определить, насколько программы похожи между собой, требуется ввести метрику.

3. *Обзор аналогов: метрики для обнаружения плагиата в текстах программ.*

3.1. Численное значение атрибутов.

Сравнение численных выражений признаков, иначе говоря, атрибутов программы, относится к атрибутным методам поиска плагиата. Программы будут считаться похожими, если полученные численные выражения атрибутов близки, например, количество операторов или операндов в программе, их количество относительно длины программы. Оценка схожести сводится к сравнению численных значений или их векторов, получаемых с помощью анализа исходного текста программ.

Можно комбинировать несколько признаков так, чтобы программа была представлена не одним значением, а вектором. Две программы будут считаться похожими, если соответствующие значения из составленных из них векторов равны или близки. Можно уточнить метод, проведя частотный анализ количества операторов в тексте или анализ их последовательности.

При высокой вероятности найти похожие программы вероятность того, что найденное решение действительно будет похожим, достаточно низкая.

Преимущества:

• Простота реализации.

• Подходит для разных языков программирования.

Недостатки:

• Не связанные между собой характеристики исходного кода плохо описывают его в целом.

• Разные программы могут получать близкие характеристики.

• Квадратичная зависимость времени от длины исходного кода программы.

3.2. Наибольшая общая подпоследовательность. Задача сводится к тому, чтобы найти все такие наибольшие последовательности, которые являются подпоследовательностями нескольких последовательностей из разных текстов программ.

Подпоследовательность получается из некоторой конечной последовательности, если удалить из нее некоторое множество ее элементов, в том числе и пустой. Например, ftion является подпоследовательностью последовательности function. Считается, что последовательность является общей подпоследовательностью двух последовательностей, если она является подпоследовательностью обеих последовательностей. Требуется для двух последовательностей найти все общие подпоследовательности наибольшей длины.

Большинство алгоритмов нахождения наибольшей общей подпоследовательности используют подход динамического программирования и работают за квадратичное время при линейной памяти [4], [5].

Вероятность найти похожие программы очень высока. Но при этом достаточно высока и вероятность того, что найденные решения не будут похожими из-за того, что подпоследовательностями могут являться короткие или часто встречающие слова.

Преимущества:

• Простота реализации.

• Подходит для разных языков программирования.

• Нечувствительность к изменению текстов.

Недостатки: для одних и тех же слов наибольшие общие подпоследовательности могут быть не единственны.

3.3. Расстояние Жаккара. Коэффициент (индекс) сходства Жаккара сравнивает элементы из двух текстов и определяет, какие из них являются общими, а какие – различными. Для улучшения метода обычно текст исходного кода программ представляют в виде *n*-грамм. Коэффициент Жаккара равен количеству общих элементов, деленному на количество всех элементов в обоих текстах, или, иначе говоря, пересечению двух текстов, деленному на их объединение. Мера показывает, насколько похожи два текста, и принимает значения от 0 до 1.

Расстояние Жаккара показывает, насколько различны два текста. Оно является дополнением к коэффициенту Жаккара и может быть найдено вычитанием коэффициента Жаккара из 1.

Метод работает за квадратичное время при линейной памяти. Вероятность найти похожие программы и того, что они будут действительно похожи, высока.

Преимущества: легко интерпретировать.

Недостатки:

• При малых наборах данных может дать неверный результат.

• Возможны совпадения при токенизированном представлении программ, но их отсутствие – в исходных кодах программ.

• Из-за небольшого количества уникальных *k*-граммов в больших программах многие совпадения, не содержащие в себе таких *k*-граммов, не будут добавлены в решение.

• Вставка или изменение найденного блока может привести к игнорированию той части блока, в которой не содержится уникальный *k*-грамм.

3.4. Расстояние Левенштейна. Для того чтобы найти функцию расстояния между двумя словами, метрику, наиболее часто применяют расстояние Левенштейна. Расстояние Левенштейна является дистанцией редактирования, т. е. минимальным количеством операций вставки, удаления или замены одного символа на другой, необходимых для превращения одной строки в другую.

Сначала текст программ разбивается на токены. Далее составляется матрица, которая заполняется расстояниями между каждым токеном.

Алгоритм имеет временную сложность O(mn) и потребляет O(mn) памяти, где m и n – длины сравниваемых строк. Можно также оптимизировать алгоритм отсечением Укконена, и свести временную сложность к O (n + d ^ 2), где n – длина более длинной строки, d – расстояние редактирования, а память к O(min(m, n)), где m и n – длины сравниваемых строк.

Вероятность найти похожие программы очень высокая, а найденные решения с высокой вероятностью действительно будут похожими.

Преимущества: токенизированное представление данных.

Недостатки:

• При перестановке местами слов или их частей могут получаться большие расстояния.

• Расстояния между совершенно разными короткими словами оказываются небольшими, а между очень похожими длинными словами — значительными.

3.5. Расстояние Колмогорова. При реализации алгоритмов колмогоровской сложности сначала текст разделяется на токены, далее идет поиск длиннейшей неточно повторяющейся подстроки, заканчивающейся в текущем символе, кодируется указателем на предыдущее размещение и сохраняет информацию о внесенных поправках. В результате будут найдены неточно совпавшие пары подстрок.

Расстояние между строками показывает, сколько информации содержит строка, и вычисляется

$$d(x,y) = 1 - \left(\frac{K(x) - K(x \vee y)}{K(xy)} \right),$$

где $K(x)$ – колмогоровская сложность строки x [7]. Чем ближе функция расстояния к 0, тем более схожи программы. Сложность Колмогорова в строке x

является минимальной длиной программы, необходимой для вычисления конкретной строки x.

Колмогоровская сложность применяется к строкам, но не применяется к программам. Было доказано, что невозможно вычислить колмогоровскую сложность строки [8].

Преимущества:

• Универсальность: две программы, близкие относительно любой другой метрики, будут близкими и относительно данной.

• Быстрота.

• Преимущества токенизированного представления данных.

• Общие маленькие подстроки игнорируются, поэтому алгоритм не чувствителен к малым, случайно совпавшим участкам кода.

• При разбиении совпавшего участка кода на несколько частей вставкой или перестановкой одного или нескольких блоков кода функция схожести слабо изменяется.

• Алгоритм нечувствителен к перестановкам больших фрагментов кода.

Недостатки:

• Возможноть совпадения токенизированного представления программ, но несовпадения в исходных кодах программ.

• Невозможность использования на практике.

Сравнение описанных алгоритмов представлено в табл. 2.

Наилучшее время показывает расстояние Колмогорова, однако его невозможно реализовать на практике. Затраты памяти наихудшие у расстояния Левенштейна. Следует отметить, что для выполнения задач важно качество преобразований текста, а затраты менее важны. Наивысшую вероятность найти похожие программы показывает расстояние Левенштейна.

Для успешного обнаружения плагиата данные разбираются с учетом:

• чувствительности к пробелам и символам. При сопоставлении текстов программ следует игнорировать дополнительные пробелы, комментарии, а также имена переменных;

• подавления шума. Обнаружение коротких совпадений не показательно, так как одно слово может встречаться в двух программах, потому что они написаны на одном языке. Все совпадения должны быть достаточно большими, чтобы считать, что программы скопированы;

• независимости от положения. Перестановка переменных или порядка строк в программе не должна влиять на обнаруженные совпадения. Добавление или удаление символов в тексте также не должно влиять на набор совпадений.

Исходный код программы разделяется на токены, так как это представление даст наиболее соответствующий требованиям результат. Многие компиляторы используют разделение текста на лексемы, а затем обрабатывают их список с помощью конечного автомата, преобразующего лексемы в токены. Библиотеки языка программирования Python позволяют обрабатывать текст без поиска лексем и преобразования их в токены, а делают токенизацию за один шаг.

Так как расстояние Левенштейна удовлетворяет требованиям, имеет наивысшую вероятность найти правильный результат, то реализован будет он.

Работа алгоритма будет осуществляться по схеме:

1. Преобразование нежелательных различий между документами.

2. Задание метрики (критерия близости).

3. Реализация нахождения расстояния Левенштейна.

4. Составление матрицы расстояний для определения похожести текстов.

Программа должна показывать, насколько коды программы похожи между собой.

Таблица 2

Метрика	Время	Память	Вероятность найти похожие программы	Вероятность, что похожие программы действительно похожи
Численные значения атрибутов	Квадратичное	Линейная	Высокая	Низкая
Наибольшая общая подпоследовательность	Квадратичное	Линейная	Очень высокая	Низкая
Расстояние Жаккара	Квадратичное	Линейная	Высокая	Высокая
Расстояние Левенштейна	Квадратичное	Квадратичная, сводится к $O(\min\{m,n\})$	Очень высокая	Высокая
Расстояние Колмогорова	Сверхлинейное	–	–	–

Реализован модуль, который сначала разбивает каждую программу из массива решений студентов по предмету «Математические пакеты» за 2015 и 2017 гг. на токены с помощью библиотеки tokenize. Далее сформированы списки из токенов для каждой программы. Реализован алгоритм нахождения расстояния Левенштейна для сравнения каждой программы с каждой. Результат записывается в файл формата csv, где содержатся названия сравниваемых программ и расстояния между ними.

В результате выполнения работы был разработан программный модуль, осуществляющий поиск плагиата в кодах программ. Исходный код программы разделяется на токены, так как токенизация позволяет сохранять существенные и игнорировать легко модифицируемые детали кода программы. Для того чтобы этот программный продукт удовлетворял требованиям (чувствительности к пробелам и символам, подавления шума, независимости от положения), проведен обзор метрик для обнаружения плагиата в текстах программ. Расстояние Левенштейна удовлетворяет требованиям и имеет наивысшую вероятность найти правильный результат: вероятность найти похожие программы, а также того, что похожие программы действительно похожи, очень высока. Недостатком решения являются затраты памяти на расчет расстояния Левенштейна. Но следует отметить, что для выполнения задач важно качество преобразований текста, а затраты менее важны. Преимуществом является открытый исходный код.

Программный модуль был протестирован на реальных данных: решениях задач студентов по программированию за 2015, 2017 гг. Расстояние Левенштейна между текстами программ соответствует степени их похожести.

Для наглядной демонстрации результатов планируется построить кластеры, которые объединяют группы похожих решений.

СПИСОК ЛИТЕРАТУРЫ

1. Бобкова О. В., Давыдов С. А., Ковалева И. А. Плагиат как гражданское правонарушение // Патенты и лицензии. 2016. № 7. С. 31–37.

2. Бойцов Л. М. Использование хеширования по сигнатуре для поиска. 2000. №. 7. С. 135–154.

3. Schütze H., Manning C. D., Raghavan P. Introduction to information retrieval / Cambridge University Press, 2008. Vol. 39. P. 19–47.

4. Ullman J. D., Aho A. V., Hirschberg D. S. Bounds on the complexity of the longest common subsequence problem // J. of the ACM (JACM). 1976. Vol. 23, № 1. P. 1–12.

5. Lueker G. S. Improved bounds on the average length of longest common subsequences // J. of the ACM (JACM). 2009. Vol. 56, № 3. P. 17.

6. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. Академии наук. 1965. Т. 163, №. 4. С. 845–848.

7. Vitanyi P. M. B., Li M. An introduction to Kolmogorov complexity and its applications. Heidelberg: Springer, 1997. Vol. 34, №. 10. С. 187–254.

8. Верещагин Н. К., Успенский В. А., Шень А. Колмогоровская сложность и алгоритмическая случайность. М.: МЦНМО, 2013. Т. 3, №. 3. С. 556.

I. A. Posov, V. Ye. Dopira
Saint Petersburg Electrotechnical University «LETI»

PLAGIARISM SEARCHING METHODS TO IN PROGRAM CODES

Types of data representation for plagiarism search in program codes have been explored, such as text without transformations, n-grams, and tokens. The splitting of program texts from an array of student solutions into an array of tokens has been implemented using the Python programming language. The requirements for plagiarism detection algorithms are formulated. And the metrics for the detection of plagiarism in the texts of programs are reviewed. The advantages and disadvantages for each metric are highlighted. The comparison was made according to the criteria: time, memory, the probability of finding a pair of similar programs, the probability that the found pair will be really similar. The metrics: the numerical values of the attributes, the longest common subsequence, the Jaccard distance, the Levenshtein distance, and the Kolmogorov distance have been compared. The calculation of the Levenshtein distance is chosen for implementation. The Python programming language has been used for implementation of the Levenshtein distance calculation algorithm for a list of tokens. The results show how the texts of the programs are similar to each other.

Plagiarism, plagiarism detection metrics, token