УДК 519.25: 004.8

А.В. Куртукова, А.С. Романов

Моделирование архитектуры нейронной сети в задаче идентификации автора исходного кода

Предлагаются новые гибридные архитектуры нейронных сетей для решения задачи идентификации автора исходного кода. Рассматриваются модели на основе популярных однонаправленных и двунаправленных сверточно-рекуррентных архитектур. Установлено, что наиболее эффективной моделью является гибридная нейронная сеть, которая достигает точности 97% и демонстрирует независимость от языка, на котором программирует автор.
Ключевые слова: модель, машинное обучение, исходный код, идентификация, глубокие нейронные сети.
doi: 10.21293/1818-0442-2019-22-3-37-42

Решения задачи идентификации автора исходного кода [1] представляют большой интерес как для отечественных, так и для иностранных исследователей в областях прикладной лингвистики, компьютерной криминалистики в качестве инструмента для идентификации авторов-вирусописателей, а также защиты интеллектуальной собственности как меры оценки оригинальности программных решений, защищенных авторскими правами.

Развитие информационных технологий влечет за собой создание новых и совершенствование уже существующих методов [2–9] обработки текстовых данных и принятия решений. При этом стоит учитывать опыт и результаты, полученные применением традиционных подходов, зарекомендовавших себя при решении аналогичных и смежных задач интеллектуального анализа текста.

В ранних трудах, посвященных решению поставленной задачи, как правило, применялись подходы на основе классических алгоритмов машинного обучения и статистики. Исследование Wang N. и Ji S. [10] посвящено методу, объединяющему статический и динамический анализы исходных кодов. Под динамическими признаками подразумевались такие признаки, как вызовы функций, выделение памяти и т.п. Использование такого метода при решении задачи определения автора исходного кода позволяет избежать необходимости полного переобучения при увеличении тренировочной выборки, а также потери точности на больших авторских корпусах. Точность метода – 94% на корпусе из 23 авторов.

Большую популярность среди исследователей имеет метод SCAP [11], применяющий для разграничения авторского инварианта статистические профили программистов, включающие в себя определенный ряд метрик (стиль комментариев, предпочитаемые типы данных, особенности именования идентификаторов и т.п.). На основе полученных профилей обучается классификатор машины опорных векторов (SVM). Такой метод позволяет получить заключение о принадлежности неизвестного образца исходного кода тому или иному профилю за короткий промежуток времени, однако имеет недостаточно высокую точность – 80% для 34 авторов, программирующих на JavaScript. В работе [12] представлена программная реализация методики SCAP, а также приведены результаты экспериментов для языков Java и C++. Точность идентификации для этих языков составила 91%.

Метод SVM использовался также в работе [13]. На вход классификатору подавался результат работы традиционного для анализа текста алгоритма машинного обучения – абстрактных синтаксических деревьев (АСД), учитывающих различные стилометрические признаки, в том числе признаки, устраняющиеся при рефакторинге программного кода. К последним относятся длинные методы, преобладание операторов включения над условными, большое количество комментариев и другие признаки, демонстрирующие степень отклонения программиста от парадигм написания кода. Точность такого подхода составила 75% на корпусе из 9 программистов, пишущих на Java.

Более современными подходами к идентификации автора исходного кода являются различные архитектуры нейронных сетей (НС). Авторы методики [14] предлагают осуществлять обучение НС на наборе признаков, полученных при помощи методов роя части (МРЧ) и обратного распространения ошибки. Так, тренировочные наборы включают в себя лексические, структурные (стиль расстановки скобок, предпочтение символов табуляции пробельным), а также синтаксические признаки, извлекаемые АСД. Классификация обученной авторами НС продемонстрировала точность 91% на корпусе из 40 авторов, программирующих на Java.

Наилучшие результаты при определении автора исходного кода демонстрируют алгоритмы глубокого обучения. Работа [15] посвящена подходу на основе ансамбля глубокой рекуррентной нейронной сети (RNN) и алгоритма случайного леса. Эксперименты, проведенные авторами, подтвердили способность такой архитектуры осуществлять идентификацию автора-программиста обфусцированного кода с точностью порядка 95%. Тренировочный корпус авторов работы включает в себя образцы 142 авторов, программирующих на языке C++.

Особого внимания заслуживают двунаправленные RNN – наиболее эффективные среди глубоких

архитектур и фундаментальные для обработки больших данных, представленных в текстовом виде. Архитектура двунаправленной LSTM (BiLSTM) была выбрана исследователями B. Alsulami и E. Dauber [16] и позволила им добиться точности идентификации автора исходного кода 96% для 25 Python-программистов и 85% для 10 программистов, пишущих на C++.

На основе проведенного анализа можно сделать вывод о высокой эффективности глубоких архитектур НС, а также ансамблей классификаторов, комбинирующих в себе традиционные алгоритмы машинного обучения и различные модели НС.

Таким образом, была сформулирована цель исследования – смоделировать и реализовать глубокую архитектуру НС, позволяющую идентифицировать автора исходного кода с высокой точностью независимо от языка программирования.

Постановка задачи

Разрабатываемая математическая модель должна быть классифицирующей, описывающей процесс определения принадлежности автора анонимного образца исходного кода программы конкретному классу, исходя из поданных ей на вход обучающих данных.

Пусть имеется множество исходных кодов программ $\mathbf{S}=\{s_1,...,s_k\}$, где каждый исходный код рассматривается как вектор признаков $\mathbf{X}=\{x_1,...,x_n\}$, и множество авторов $\mathbf{A}=\{a_1,...,a_l\}$. Для подмножества исходных кодов $\mathbf{S}'=\{s_1,...,s_m\}\subseteq\mathbf{S}$, где $m<k$ – авторы известны, существует множество пар «исходный код-автор» $(s_i,a_j)\in\mathbf{D}\subseteq\mathbf{S}'\times\mathbf{A}$, где $s_i\in\mathbf{S}'$, $a_j\in\mathbf{A}$. Решение задачи состоит в идентификации автора, принадлежащего множеству **A**, являющегося истинным автором анонимных образцов исходных кодов подмножества $\mathbf{S}''=\mathbf{S}/\mathbf{S}'$.

Данную задачу следует рассматривать как задачу многоклассовой классификации. Так, множество **A** состоит из множества предопределенных классов и их меток, множество **D** включает в себя исходные кода для обучения классификатора, а множество **S''** содержит классифицируемые исходные коды.

Цель состоит в разработке модели, решающего поставленную задачу – нахождение целевой функции $\mathbf{F}:\mathbf{S}\times\mathbf{A}\to[0,1]$, которая относит некоторый исходный код из множества **S** к его истинному автору. Значение функции описывается как степень принадлежности объекта классу, где 1 соответствует полное соответствие целевому классу, а 0, напротив, полное несоответствие.

Моделирование архитектур нейронных сетей

При анализе исходных кодов следует учитывать особенности искусственных текстов, в частности, семантические и синтаксические правила, структурные признаки, а также парадигмы программирования. Этим обусловлен выбор сверточно-рекуррентной архитектуры (CRNN) для решения поставленной задачи. С целью выявления большого количества информативных признаков следует обрабатывать исходные коды на символьном уровне. Такой вид представления данных позволит CRNN «охватывать» как объемные языковые конструкции, так и локальные особенности написания программного кода.

Наиболее сложным процессом при моделировании архитектуры CRNN является выбор самой эффективной для идентификации автора исходного кода архитектуры. В рамках данного исследования было решено смоделировать следующие CRNN:

– CNN-SimpleRNN;
– SeparableCNN-SimpleRNN;
– CNN-GRU/LSTM;
– CNN-BiGRU/BiLSTM.

Перечисленные архитектуры имеют различные математические аппараты и существенные отличия в процессах осуществляемого обучения.

В общем случае формирование выходной карты признаков скрытого слоя *l* CNN архитектурой можно описать следующим образом:

$$h_j^l = f(\textstyle\sum_i x_i^{l-1} * k_j^l + b_j^l), \qquad (1)$$

где f – функция активации; b_j – коэффициент сдвига для карты признаков; k_j – ядро свертки номер *j*; x_i^{l-1} – карта признаков предыдущего слоя; * – операция свертки.

Работа как простых CNN, так и отделимых CNN (SeparableCNN) заключается в параллельном анализе *n*-грамм, где *n* определяется размером фильтра свертки. Различие данных архитектур состоит лишь в способности отделимых CNN раскладывать ядро свертки на меньшие ядра, что в некоторых случаях может дать прирост точности за счет более глубокого анализа.

Состояние простой RNN в момент времени *t* формально может быть задано как

$$s_t = f(a_t) = f(b + \mathbf{W}_{s_{t-1}} + \mathbf{U}_{x_t}). \qquad (2)$$

Тогда выход простой RNN будет выглядеть как:

$$y_t = h(o_t) = h(c + V_{s_t}), \qquad (3)$$

где *f*() – нелинейность RNN (как правило, сигмоидальная функция), *h* – функция активации выходного слоя (для мультиклассовой классификации – Softmax), *b*, *c* – внутренние параметры, **W** – матрица весов для перехода между скрытыми состояниями, **U** – матрица входов, **V** – матрица выходов.

На практике простой RNN не всегда хватает для «запоминания» далеко расположенных в текстовой последовательности зависимостей. Наиболее распространенным подходом к решению такой проблемы является использование долгой краткосрочной памяти (LSTM), основанной на гейтах (узлах) входа, забывания и выхода. Гейт забывания f_t функционирует на основе сцепленных и переданных сигмоидальной функции на предыдущем шаге значений. На выходе она выводит значение f_t в диапазоне [0;1]. Затем f_t и c_{t-1} поэлементно умножаются: если получается значение 0, то f_t исключается из c_{t-1}, если 1, то включается. Гейт забывания можно описать как

$$f_t = \sigma(\mathbf{W}_f \cdot [h_{t-1}, \mathbf{x}_t] + b_f), \quad (4)$$

где $\mathbf{W}_f$ – весовая матрица забывания, $\mathbf{h}_t$ – вектор скрытого состояния в момент t, $\mathbf{x}_t$ – входной вектор в момент t, b_f – внутренний параметр забывания; σ – сигмоидальная функция.

При работе фильтра обновления происходит сцепление входного значения и представления из предыдущего шага. Функция тангенса позволяет получить несколько значений, из которых с помощью сигмоидальной функции производится выбор включаемых в c_{t-1} значений:

$$i_t = \sigma(\mathbf{W}_i \cdot [h_{t-1}, \mathbf{x}_t] + b_i), \quad (5)$$

$$\tilde{C}_t = \tan h(\mathbf{W}_c \cdot [h_{t-1}, \mathbf{x}_t] + b_c), \quad (6)$$

где $\mathbf{W}_c$, $\mathbf{W}_i$ – весовые матрицы обновления и входа, b_c, b_i – внутренние параметры обновления и входа.

Полученные значения могут использоваться как выходные. Состояние C_t ячейки передается в функцию тангенса и с помощью поэлементного умножения выбирается значение, которое будет включено в c_{t-1}.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t, \quad (7)$$

$$o_t = \sigma(\mathbf{W}_o [h_{t-1}, \mathbf{x}_t] + b_o), \quad (8)$$

$$h_t = o_t \cdot \tan h(C_t), \quad (9)$$

где $\mathbf{W}_o$ – весовая матрица выхода, b_o – внутренние параметры выхода.

Эффективность использования двунаправленных архитектур была доказана исследователями экспериментально [16]. Этим обусловлена необходимость их моделирования в рамках решения поставленной задачи.

Особенность BiLSTM состоит в наличии двух типов связей: одна направлена вперед во времени, что дает возможность учитывать представления с предыдущих шагов, а другая направлена обратно, что позволяет учитывать представления шагов, расположенных впереди. Таким образом, с целью реализации BiLSTM математический аппарат однонаправленной LSTM дополняется аналогичными гейтами и фильтрами, однако ориентированными на шаг во времени t+1. На примере гейта забывания

$$f_t = \sigma(\mathbf{W}_f \cdot [h_{t+1}, \mathbf{x}_t] + b_f). \quad (10)$$

Недостатком LSTM- и BiLSTM-архитектур является их требовательность к вычислительным ресурсам. При недостатке вычислительных ресурсов данные НС могут быть замещены своими более простыми аналогами – управляемыми рекуррентными блоками (GRU) и их двунаправленной модификацией (BiGRU).

В GRU используются гейты u_t обновления как альтернатива комбинации входного и забывающего гейтов LSTM и перезагрузки r_t как способ переноса памяти от одного шага к другому.

Гейт обновления u_t отбирает информацию, которая должна пойти дальше, из предыдущих шагов с помощью сигмоидальной функции

$$f_u = \sigma(\mathbf{W}_u \cdot [h_{t-1}, \mathbf{x}_t] + b_u). \quad (11)$$

Гейт перезагрузки можно представить как:

$$r_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{W}_{hr} h_{t-1} + b_r), \quad (12)$$

где $\mathbf{W}_u$, $\mathbf{W}_r$ – весовые матрицы обновления и перезагрузки, b_u, b_r – внутренние параметры обновления и перезагрузки.

Следовательно, выход блока h_t является комбинацией предыдущего выхода h_{t-1} и кандидата в выход h_t, являющегося зависимым от гейта перезагрузки:

$$\tilde{h}_t = \tan h(\mathbf{W}_{x\tilde{h}} \cdot \mathbf{x}_t + \mathbf{W}_{h\tilde{h}} [r_t \cdot h_{t-1}]), \quad (13)$$

$$h_t = (1 - u_t) \cdot \tilde{h}_t + h_{t-1}). \quad (14)$$

Принцип работы двунаправленной GRU (BiGRU) аналогичен BiLSTM.

Постановка эксперимента и его результаты

Для получения достоверных результатов работы каждой модели была собрана база исходных кодов программ с ресурса GitHub [17]. В базу были включены как наиболее популярные согласно индексу TIOBE [18], так и только набирающие популярность языки программирования. Подробная информация о собранной экспериментальной базе представлена в табл. 1.

Таблица 1

Экспериментальная база

Язык	Количество кодов	Количество авторов	Средняя длина кода, символов
C++	12366	72	988
Java	39708	73	2409
JS	18735	69	397
Python	16783	57	532
C	17274	62	1162
C#	19378	71	638
Ruby	19150	58	304
PHP	17158	80	374
Swift	12672	74	775
Go	14067	81	816
Groovy	14002	68	167
Kotlin	15274	72	301
Perl	11189	61	251

Рассмотренные модели были реализованы на языке программирования Python с использованием библиотек для глубокого анализа данных Keras [19] и TensorFlow [20].

Оценка качества моделей производилась посредством перекрестной проверки по 10 блокам (кроссвалидации). Такая процедура позволяет избежать проблему чрезмерного разброса оценок при проверке и получить достоверный результат. Точность на каждом из блоков определяется как

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}, \quad (15)$$

где TP – истинно-положительное решение, TN – истинно отрицательное решение, FP – ложно положительное решение, FN – ложно отрицательное решение. Данные показатели рассчитываются на основе таблицы сопряженности, где содержится информация о том, сколько раз система приняла верное и сколько раз неверное решение по исходным кодам заданного класса.

Результаты валидации описанных моделей на корпусе из 10 авторов, ограниченном 30 файлами и 500 символами, представлены в табл. 2.

В каждую из моделей была включена CNN-часть, позволяющая всем моделям вести параллельную обработку униграмм, триграмм и пентаграмм символов поданного на вход исходного кода. Наибольшая точность достигается моделями с BiLSTM и BiGRU, позволяющими производить обработку таких *n*-грамм в прямом и обратном порядке. Для данных моделей были проведены дополнительные эксперименты с корпусами, включающими в себя образцы 5 и 20 авторов. Результаты представлены в табл. 3.

Таблица 2

Результаты валидации моделей

Язык	CNN-RNN, %	CNN-GRU, %	CNN-BiGRU, %	CNN-LSTM, %	CNN-BiLSTM, %	SCNN-RNN, %
C++	77	71,8	87,5	83	65	64
Java	58	74	92	76,4	96,4	66
JS	60,9	67	88	73	83	59,9
Python	42	74	92	82	92	60,5
C	57	73,2	93	76	96	58
C#	73	69,8	95	85,2	95,2	61,8
Ruby	68	62	89	72,8	92	47,6
PHP	75	68,5	88,3	84,3	82	58
Swift	44,5	72	94	80	62	44
Go	55	81,4	86	84	90	50
Groovy	49,3	78	96	92,7	97	60
Kotlin	25	71,5	85	89,1	67,9	38
Perl	62	69	91	87	59	55

Таблица 3

Результаты экспериментов с архитектурами

Архитектура	CNN-BiGRU			CNN-BiLSTM		
Авторы (кол-во)	5	10	20	5	10	20
Java, %	97,2	92	90	98,5	96,4	93,5
C, %	96,1	93	89	97	96	74
C++, %	92	87,5	81,5	93	65	59
Python, %	95,2	92	90	100	92	72
C#, %	95,5	95	93,3	96	95,2	56
JS, %	91,5	88	86,2	94	83	73
PHP, %	92	88,3	82,8	96	82	45
Ruby, %	92,7	89	86	100	92	90
Swift, %	98,3	94	89,2	95,2	62	49
Go, %	93,1	86	83	94	90	62
Groovy, %	99	96	92,5	97,9	97	75
Kotlin, %	91	85	80,9	92	67,9	55
Perl, %	96,3	91	87	98	59	57

Для вынесения конкретных рекомендаций по применению модели на основе гибридной нейронной сети (HNN) были проведены эксперименты, направленные на выявление влияния объема корпуса (количества исходных кодов и их длин) на итоговый результат классификации. Результаты данных экспериментов над HNN с рекуррентной частью BiGRU представлены на рис. 1 и 2. Данные графики демонстрируют точности кроссвалидации на различных по объему выборках для 10 и 20 авторов.

Заключение

В данной статье для решения поставленной задачи были рассмотрены шесть моделей на основе различных CRNN. Научная новизна работы заключается в применении ранее неопробованных в задачах интеллектуального анализа текста архитектур НС, выявлении наиболее эффективных из них и дальнейшем внедрении в программную систему для идентификации автора исходного кода.

Сравнительно низкая точность моделей (~71%), включающих в себя однонаправленные рекуррентные архитектуры (SimpleRNN, GRU, LSTM), обусловлена их неспособностью в отличие от двунаправленных сетей анализировать символьную последовательность как в прямом, так и в обратном порядке.

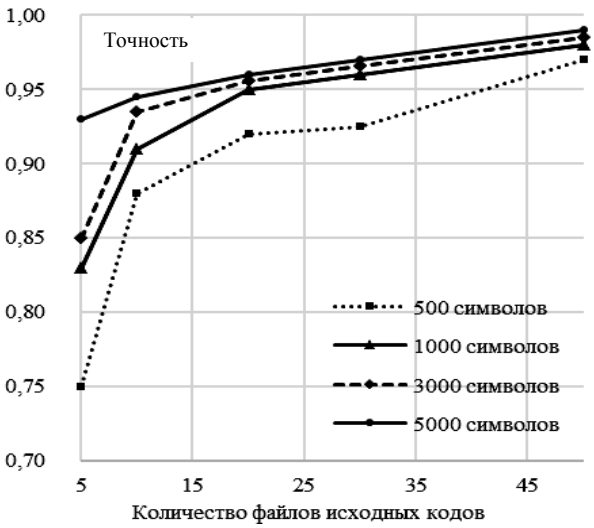

Рис. 1. Эксперимент с 10 авторами

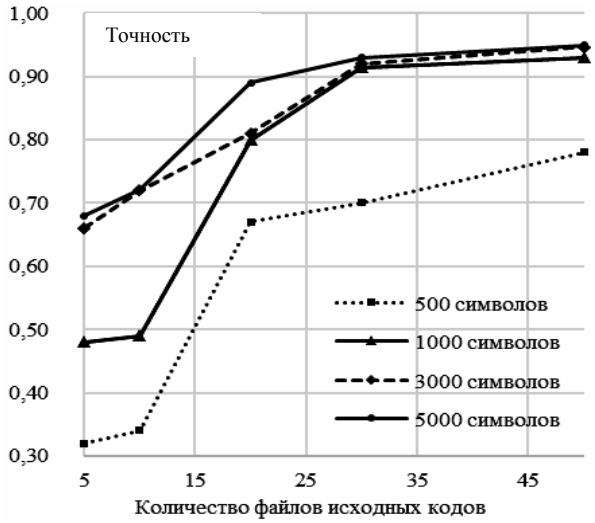

Рис. 2. Эксперименты с 20 авторами

Наибольшая точность (97%) была достигнута моделью, комбинирующей в себе архитектуры многослойной CNN и BiLSTM, однако данная модель оказалась неприменимой к большим по объему авторским корпусам.

Таким образом, хотя модель CNN-BiGRU и демонстрирует менее точный в сравнении с моделью CNN-BiLSTM результат, ввиду наличия у последней большего числа встроенных фильтров и операций, эксперименты показали ее эффективность при увеличении тренировочной выборки, а также независимость от языка программирования, на котором пишет автор.

Исследование, проведенное с использованием модели CNN-BiGRU, позволило определить оптимальный для получения достоверных результатов объем тренировочной выборки:

– для достижения точности 90% и выше на корпусе 10 авторов необходимо применять не менее 10 файлов объемом от 1 000 символов;

– на корпусе 20 авторов необходимо применять не менее 30 файлов объемом от 1 000 символов.

Уменьшение количества файлов должно сопровождаться увеличением объема одного исходного кода и, напротив, уменьшение объема должно сопровождаться уменьшением количества исходных кодов в корпусе.

Следует отметить, что все реализованные модели работают с не обработанными предварительно исходными кодами, а не с пространством признаков, выделенных экспертом. Это позволяет моделям выявлять закономерности, не контролируемые программистом на подсознательном уровне, а значит делает их устойчивыми к намеренному запутыванию кода.

Литература

1. Куртукова А.В. Идентификация автора исходного кода методами машинного обучения / А.В. Куртукова, А.С. Романов // Труды СПИИРАН. – 2019. – № 18(3). – С. 741–765.

2. Анализ тональности текстов с использованием методов машинного обучения / А.С. Романов, А.В. Куртукова, Р.В. Мещеряков, М.И. Васильева // Сб. трудов конф. «The II International Conference R. Piotrowski's Readings LE & AL'2017», СПб., 27 ноября 2017 г. – М.: Jeusfeld c/o Redaktion Sun SITE, Informatik V., 2018. – С. 86–95 [Электронный ресурс]. – Режим доступа: http://ceur-ws.org/Vol-2233, свободный (дата обращения: 12.06.19).

3. Романов А.С. Обобщенная методика идентификации автора неизвестного текста / А.С. Романов, А.А. Шелупанов, С.С. Бондарчук // Доклады ТУСУР. – 2010. – № 1(21), ч. 1. – С. 108–112.

4. Романов А.С. Методика проверки однородности текста и выявления плагиата на основе метода опорных векторов и фильтра быстрой корреляции / А.С. Романов, Р.В. Мещеряков, З.И. Резанова // Доклады ТУСУР. – 2014. – № 2(32). – С. 264–269

5. Исхакова А.О. Модель процесса формирования инвариантов классов текстов / А.О. Исхакова // Доклады ТУСУР. – 2016. – Т. 19, № 3. – С. 76–80.

6. Костюченко Е.Ю. Обработка естественной информации на основе аппарата нейронных сетей / Е.Ю. Костюченко // Доклады ТУСУР. – 2009. – № 1(19), ч. 2. – С. 54–56.

7. Чемерилов В.В. Система автоматического разрешения омографии на основе семантической связи слов смежных предложений в текстовом отрывке / В.В. Чемерилов, А.С. Фадеев // Доклады ТУСУР. – 2018. – Т. 21, № 3. – С. 42–48.

8. Mäntylä M.V. The Evolution of Sentiment Analysis / M. Viking, D. Graziotin, M. Kuutila // A Review of Research Topics, Venues, and Top Cited Papers. – 2018. – ArXiv abs/1612.01556.

9. Burrows S. Application of information retrieval techniques for source code authorship attribution / S. Burrows, A. Uitdenbogerd, A. Turpin // 14th International Conference on Database Systems for Advanced Applications. – 2009. – P. 699–713.

10. Wang N. Integration of Static and Dynamic Code Stylometry Analysis for Programmer De-anonymization / N. Wang, S. Ji // Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security. – 2018. – P. 74–84.

11. Wisse W. Scripting DNA: Identifying the JavaScript Programmer / W. Wisse, C.J. Veenman // Digit. Investig. – 2015. – № 15. – P. 61–71.

12. Tennyson M.F. ASAP: A Source Code Authorship Program // International Journal on Software Tools for Technology Transfer. – 2019. – P. 1–14.

13. Zia T. Source Code Author Attribution Using Author's Programming Style and Code Smells / T. Zia, M. Ilyas // I.J. Intelligent Systems and Applications. – 2017. – № 5. – P. 27–33.

14. Yang X. Authorship attribution of source code by using backpropagation neural network based on particle swarm optimization / X. Yang, Q. Li, Y. Guo, M. Zhang // PLoS ONE. – 2017. – № 12(11) 95 [Электронный ресурс]. – Режим доступа: https://doi.org/10.1371/journal.pone.0187204, свободный (дата обращения: 14.06.19).

15. Abuhamad M. Large-Scale and Language-Oblivious Code Authorship Identification / M. Abuhamad, T. AbuHmed, A. Mohaisen, D. Nyang // Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. – 2018. – P. 101–114.

16. Alsulami B. Source Code Authorship Attribution using Long Short-Term Memory Based Networks / B. Alsulami, E. Dauber, R. Harang, S, Mancoridis, R. Greenstadt // Proceedings of the 22nd European Symposium on Research in Computer Security. – 2017. – P. 65–82.

17. Github [Электронный ресурс]. – Режим доступа: https://github.com/, свободный (дата обращения: 17.06.19).

18. Index TIOBE [Электронный ресурс]. – Режим доступа: https://www.tiobe.com, свободный (дата обращения: 22.06.19).

19. Keras [Электронный ресурс]. – Режим доступа: https://keras.io/, свободный (дата обращения: 25.06.19).

20. TensorFlow [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/, свободный (дата обращения: 25.06.19).

Куртукова Анна Владимировна
Студентка каф. безопасности информационных систем (БИС) Томского государственного университета систем управления и радиоэлектроники (ТУСУР)
Ленина пр-т, д. 40, г. Томск, 634050
Тел.: +7-905-991 6713
Эл. почта: av.kurtukova@gmail.com

Романов Александр Сергеевич
Канд. техн. наук, доцент каф. безопасности информационных систем (БИС)
Томского государственного университета систем управления и радиоэлектроники (ТУСУР)
Ленина пр-т, д. 40, г. Томск, 634050
Тел.: + 7 (382-2) 41 34 26
Эл. почта: alexx.romanov@gmail.com

Kurtukova A.V., Romanov A.S.

Modeling the neural network architecture to identify the author of the source code

The paper proposes new hybrid architectures of neural networks to solve the problem of identifying the author of the source code. Models based on popular unidirectional and bidirectional convolutional-recurrent architectures are considered. The most effective model achieves an accuracy of 97% and demonstrates independence from the language in which the author programs, which makes it better in comparison with analogues.

Keywords: model, machine learning, source code, identification, deep neural networks.
doi: 10.21293/1818-0442-2019-22-3-37-42

References

1. Kurtukova A.V., Romanov A.S. [Identification author of source code by machine learning methods]. *SPIIRAS Proceedings*, 2019, vol. 18, no. 3. pp. 741–765 (in Russ.).

2. Romanov A.S., Vasilieva M.I., Kurtukova A.V., Meshcheryakov R.V. [Sentiment Analysis of Text Using Machine Learning Techniques]. *Proceedings 2nd International Conference «R. PIOTROWSKI'S READINGS LE & AL'2017»* (Saint-Petersburg, 2017). Saint-Petersburg, Informatik V Publ., 2018, pp. 86–95 (In Russ.). Available at: http://ceur-ws.org/Vol-2233 (Accessed: June 12, 2019).

3. Romanov A.S., Shelupanov A.A., Bondarchuk S.S. Generalized authorship identification technique]. *Proceedings of TUSUR University*, 2010, vol. 1, no. 21, pp. 108–112 (in Russ.).

4. Romanov A.S., Meshcheryakov R.V., Rezanova Z.I. [Plagiarism detection and text homogeneity checking technique based on one-class support machine and fast correlation-based filter]. *Proceedings of TUSUR University*, 2014, vol. 2, no. 32, pp. 264–269 (in Russ.).

5. Iskhakova A.O. [Model to set up the texts class invariants]. *Proceedings of TUSUR University*, 2016, vol. 19, no. 3, pp. 76–80 (in Russ.).

6. Kostyuchenko E.Y. [Processing of the natural information on the basis of neural networks]. *Proceedings of TUSUR University*, 2009, vol. 1, no. 19, pp. 54–56 (in Russ.).

7. Chemerilov V.V., Fadeev A.S. [System of automatic homography resolution based on the semantic connection words of adjacent sentences in a text passage]. *Proceedings of TUSUR University*, 2018, vol. 21, no. 3, pp. 42–48 (in Russ.).

8. Mäntylä M.V., Graziotin D., Kuutila M. The Evolution of Sentiment Analysis. *A Review of Research Topics, Venues, and Top Cited Papers*, 2018, *ArXiv* abs/1612.01556.

9. Burrows S. Uitdenbogerd A., Turpin A. Application of information retrieval techniques for source code authorship attribution. *14th International Conference on Database Systems for Advanced Applications*, 2009, pp. 699–713.

10. Wang N., Ji S. Integration of Static and Dynamic Code Stylometry Analysis for Programmer De-anonymization. *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, 2018, pp. 74–84.

11. Wisse W., Veenman C.J. Scripting DNA: Identifying the JavaScript Programmer. *Digit. Investig*, 2015, no. 15, pp. 61–71.

12. Tennyson M. F. ASAP: A Source Code Authorship Program. *International Journal on Software Tools for Technology Transfer*, 2019, pp. 1–14.

13. Zia T., Ilyas M. Source Code Author Attribution Using Author's Programming Style and Code Smells. *I.J. Intelligent Systems and Applications*, 2017, no. 5, pp. 27–33.

14. Yang X, Li Q., Guo Y., Zhang M. Authorship attribution of source code by using backpropagation neural network based on particle swarm optimization. *PLoS ONE*, 2017, vol. 12, no. 11. Available at: https://doi.org/10.1371/journal.pone.0187204 (Accessed: June 14, 2019).

15. Abuhamad M., AbuHmed T., Mohaisen A., Nyang D. Large-Scale and Language-Oblivious Code Authorship Identification. *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 101–114.

16. Alsulami B., Dauber E., Harang R., Mancoridis S., Greenstadt R. Source Code Authorship Attribution using Long Short-Term Memory Based Networks. *Proceedings of the 22nd European Symposium on Research in Computer Security*, 2017, pp. 65–82.

17. Github. Available at: https://github.com/ (Accessed: June 17, 2019).

18. Index TIOBE. Available at: https://www.tiobe.com/ (Accessed: June 22, 2019).

19. Keras Documentation. Available at: https://keras.io/ (Accessed: June 25, 2019).

20. TensorFlow Documentation. Available at: https://www.tensorflow.org/ (Accessed: June 15, 2019).

Anna V. Kurtukova
Student, Department of Information System Security of Tomsk State University of Control Systems and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
Phone: +7-905-991 6713
Email: av.kurtukova@gmail.com

Aleksandr S. Romanov
Ph.D., Associate professor, Department of Information System Security of Tomsk State University of Control Systems and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
Phone: + 7 (382-2) 41 34 26
Email: alexx.romanov@gmail.com