УДК 519.25: 004.8 **DOI** 10.15622/sp.2019.18.3.741-765

А.В. Куртукова, А.С. Романов

ИДЕНТИФИКАЦИЯ АВТОРА ИСХОДНОГО КОДА МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ

Куртукова А.В., Романов А.С. **Идентификация автора исходного кода методами машинного обучения.**

Аннотация. Статья посвящена анализу проблемы определения автора исходного кода, которая представляет интерес для исследователей в области информационной безопасности, компьютерной криминалистики, оценки качества образовательного процесса, защиты интеллектуальной собственности.

Представлен подробный анализ современных решений проблемы. Предлагаются две новые методики идентификации на основе алгоритмов машинного обучения: машины опорных векторов, фильтра быстрой корреляции и информативных признаков; гибридной сверточно-рекуррентной нейронной сети.

Эксперименты проводились на базе исходных кодов, написанных на Java, C++, Python, PHP, JavaScript, C, C# и Ruby. Данные были получены с веб-сервиса для хостинга IT-проектов Github. Общее количество исходных кодов превышает 150 тысяч образцов, средняя длина каждого из которых составляет 850 символов. Размер корпуса — 542 автора.

С помощью перекрестной проверки по 10 блокам оценена точность разработанных методик для различного количества авторов. Для наиболее популярного языка программирования Java проведен дополнительный ряд экспериментов с количеством авторов от 2 до 50 и приведены графики зависимости точности идентификации от размера корпуса.

Анализ результатов показал, что методика на основе гибридной нейронной сети способна достигать точности 97%, что является наилучшим результатом на сегодняшний день. Методика на основе машины опорных векторов позволила добиться точности 96%. Гибридная нейронная сеть оказалась точнее машины опорных векторов в среднем на 5%.

Ключевые слова: автор исходного кода, глубокое обучение, нейронная сеть, SVM, HNN.

1. Введение. Идентификация автора исходного кода программы является актуальной задачей и подразумевает определение особого авторского стиля программиста, в частности выявление индивидуальных привычек, профессиональных приемов и методов написания кода программного обеспечения (ПО).

Сформулируем задачу идентификации автора исходного кода следующим образом. Пусть имеется множество исходных кодов $S = \{s_1, ..., s_k\}$ и множество авторов-программистов $P = \{p_1, ..., p_l\}$. Для некоторого подмножества исходных кодов $S' = \{s_1, ..., s_m\} \subseteq S$, где $m<k$ — авторы известны, существует множество пар «исходный код-автор» $(s_i, p_j) \in D \subseteq S' \times P$, где $s_i \in S'$, $p_j \in P$. Необходимо установить, какой автор-программист из множества P является истинным автором анонимных образцов исходных кодов $S'' = S / S'$.

В данной постановке задачу идентификации автора можно рассматривать как задачу классификации с несколькими классами. В таком случае множество P состоит из множества предопределенных классов и их меток, D включает в себя тренировочные образцы, а множество S'' содержит классифицируемые образцы.

Целью является построение классификатора, который решает поставленную задачу — нахождение некоторой целевой функции $F : S \times P \to [-1,1]$, относящей случайный исходный код множества S к его истинному автору. Значение функции интерпретируется как степень принадлежности объекта классу: 1 соответствует полностью положительному решению, -1 — отрицательному. При этом каждый текст рассматривается как вектор признаков $X = \{x_1,...,x_n\}$.

Полученные решения задачи находят широкое практическое применение в различных отраслях:

1. Авторские права на ПО. Стремительная информатизация общества влечет за собой множество нарушений авторских прав и бесконтрольный плагиат исходных кодов программных продуктов. Использование автоматизированных систем, способных точно идентифицировать автора ПО, в судебных разбирательствах, предметом которых являются споры об интеллектуальной собственности, может позволить избежать серьезных убытков в коммерческой сфере.

2. Информационная безопасность. Развитие криптографических технологий делает возможной почти полную анонимность в сети Интернет, что порождает рост количества киберпреступлений. Различные вирусы, трояны, руткиты, кибератаки ставят под угрозу корректное функционирование даже самых защищенных систем, что делает проблему определения автора вредоносного ПО важным аспектом форензики.

3. Образовательный процесс. Выявление плагиата в студенческих работах является важным аспектом образовательного процесса. Методики идентификации автора исходного кода позволяют проверять работы студентов на плагиат по дисциплинам, связанным с программированием.

Исходя из актуальности, в разработке новых подходов к идентификации автора исходного кода и совершенствовании уже существующих заинтересованы как отечественные, так и зарубежные исследователи.

Подход авторов работы [1] состоит в применении методов роя частиц (МРЧ) и обратного распространения ошибки. Нейронная сеть (НС) обучается на вычисленном с помощью этих методов наборе признаков. Такой набор включает лексические, структурные и синтаксические метрики. Предложенный подход был оценен на ис-

ходных кодах 40 авторов, программирующих на языке Java. Его точность составила 91%.

В статье [2] предлагается усовершенствовать тривиальную для задач определения авторства методику, которая основана на абстрактных синтаксических деревьях (АСД), добавив ансамбль с глубокой НС. С этой целью авторами были опробованы рекуррентные архитектуры НС: обычная (LSTM) и двунаправленная (BiLSTM) модели с долгой краткосрочной памятью. Точность такой методики для выборки из 25 python-программистов составила 92% для LSTM и 96% для BiLSTM; для выборки из 10 авторов, программирующих на языке C++, 80% и 85% соответственно.

В статье [3] демонстрируется совокупное применение *n*-грамм, структурных метрик, полученных из АСД, статистических метрик, а также популярного метода SCAP (авторского профиля исходных кодов). Извлеченные признаки представляются в виде вектора объектов в многомерном пространстве. Классификация производилась посредством машины опорных векторов (SVM) с линейным ядром. Оценка точности классификации проводилась с помощью 5-фолдовой кроссвалидации. Результатом применения методики стала точность 80% для 34 авторов, программирующих на JavaScript.

Суть работы [4] заключается в применении метода стилометрии, нечетких АСД и «запахов кода», указывающих на авторские недочеты в программе. Полученные признаки использовались авторами для обучения SVM, алгоритма J48, наивного байесовского классификатора, KNN (k ближайших соседей) классификатора. Точность, полученная SVM, составила 75% на выборке из 9 авторов, программирующих на Java.

Методика [5] основывается на алгоритмах классификации случайных деревьев и нечетких АСД. Такая методика позволила авторам получить точность 90% для программистов, пишущих на Python. В дальнейшем методика была усовершенствована калибровочными кривыми для анализа неполных и некомпилируемых образцов кода, что позволило авторам получить точность 73% при наличии лишь одного образца исходного кода автора, программирующего на C++ [6].

Статья [7] посвящена методу, объединяющему статический (ключевые слова, структурные элементы и др.) и динамический (вызовы функций, выделяемая память и др.) анализы стилометрии. Преимущества такого метода состоят в применении небольшого количества созданных вручную функций, возможности расширения исследуемой выборки без переобучения и устойчивости результата к изменению количества авторов, участвующих в эксперименте. Точность, демонстрируемая методом, достигает 94% на корпусе из 23 авторов.

В статье [8] предлагается система идентификации автора исходного кода на основе глубокого обучения (DL-CAIS), позволяющая осуществлять идентификацию независимо от языка программирования и обфускации. Алгоритм глубокого обучения включает в себя многослойную рекуррентную НС (RNN) и полносвязные слои. Затем полученное с помощью НС представление подается на классификатор random forest (случайный лес) с целью масштабирования. Эксперименты показали точность идентификации 95,2%.

В статье [9] используется глубокое обучение для извлечения признаков исходных кодов, представленных в виде *n*-грамм. Полученные признаки преобразуются с помощью стекового автоэнкодера, позволяющего скрытым слоям выявлять наиболее сильные взаимосвязи. Далее они подаются на вход SVM для классификации. Результатом кросс-валидации стала точность 95%.

Рассмотренные научные труды позволяют сделать вывод о безусловной эффективности различных методов машинного обучения (МО) при решении задачи идентификации автора исходного кода. Более того, традиционные и статистические методы демонстрируют точность, сопоставимую популярным алгоритмам глубокого обучения.

Однако традиционные алгоритмы МО, не подразумевающие использование глубоких архитектур, не позволяют системе работать с данными без предобработки и требуют экспертных знаний по программированию у исследователя.

Данная статья посвящена анализу отличительных особенностей, достоинств и недостатков двух наиболее популярных среди исследователей подходов: SVM как классификатора, проверенного временем и демонстрирующего в случае верного выбора информативных признаков хороший результат в различных задачах [10]; и глубокой НС, способной выделять информативные признаки самостоятельно, как современного метода принятия решения. На их основе разработаны две новые методики идентификации автора исходного кода:

1. Машины опорных векторов, фильтра быстрой корреляции и информативных признаков.

2. Гибридной нейронной сети HNN.

Рассмотрим описание разработанных методик и полученные результаты.

2. Методика на основе SVM, фильтра быстрой корреляции и информативных признаков. Многообразие методов МО обусловлено большим количеством средств, на основе которых они функционируют: математическая статистика, методы оптимизации, теория вероятностей и другие. Эти методы различны в своей реализации, однако

наиболее распространенными являются методы обучения с учителем и без учителя. К первому типу относится SVM.

Модели SVM подобны классическому перцептрону. Применение их ядровых преобразований позволяет обучать сети радиальных базисных функций и перцептроны с сигмоидальной функцией активации, веса НС которых определяются путем решения задачи квадратичного программирования с линейными ограничениями. Тогда как обучение стандартной искусственной НС подразумевает решение задачи невыпуклой минимизации без ограничений. Также SVM дает возможность прямой работы с векторным пространством высокой размерности и избавляет от необходимости предварительного анализа и снижения количества измерений.

Обобщенная методика идентификации автора исходного кода на основе рассмотренной модели SVM представлена на рисунке 1.

В отличие от моделей глубокого обучения, SVM не способна распознавать информативные признаки самостоятельно. Поэтому возникает необходимость их выделения из исходных кодов обучающей выборки. Метрики, используемые в методике, были определены на основе работы [4] и включали в себя: лексические (таблица 1) и структурные (таблица 2) признаки с высокой разделяющей способностью, а также признаки «плохого кода» (таблица 3) как способа повышения разделяющей способности классификатора.

Таблица 1. Набор лексических признаков

Признак	Описание
keywordToLength	Отношение числа ключевых слов к длине кода
TernaryToLength	Отношение числа вхождений тернарного оператора к длине кода
TokensToLength	Отношение числа токенов к длине кода
CommentsToLength	Отношение числа комментариев к длине кода
LiteralsToLength	Отношение числа литералов к длине кода
MacrosToLength	Отношение числа директив препроцессора к длине кода
FunctionsToLength	Отношение числа функций к длине кода
avgLineLength	Средняя длина строки
avgParams	Среднее число параметров среди всех функций
nestingDepth	Наивысшая степень вложенности операторов и циклов

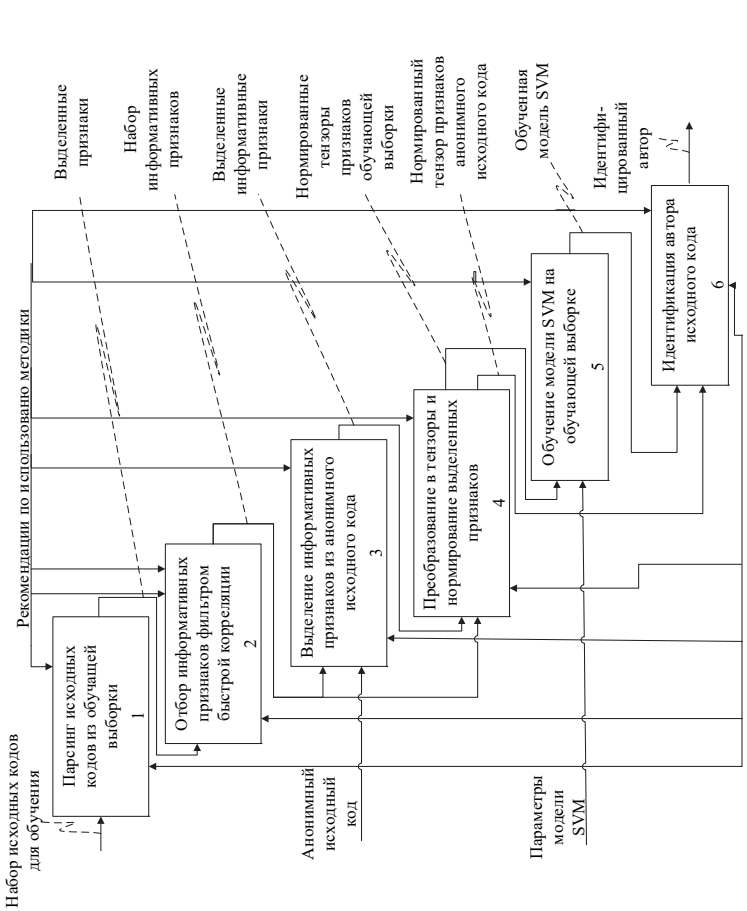

Рис. 1. Обобщенная методика идентификации автора исходного кода на основе SVM

Лексические признаки выражают степень приверженности автора к определенным конструкциям, а также демонстрируют статистику использования функций и глубину вложенности циклов. Кроме того, в данный набор включен такой признак, как отношение числа ключевых слов к длине исходного кода, что дает представление о предпочтениях автора в синтаксических конструкциях, а также об уровне его профессиональной квалификации.

Набор структурных признаков отражает склонность автора кода к применению определенного типа пробельных символов — табуляций или пробелов. В некоторых языках программирования отступы обязательны, так как они помогают компилятору понять, где начинаются и заканчиваются блоки команд. Пример такого языка — Python. В языке C++, напротив, отступы применяются по усмотрению автора кода.

Таблица 2. Набор структурных признаков

Признак	Описание
TabsToLength	Отношение числа символов табуляции к длине кода
SpacesToLength	Отношение числа символов пробела к длине кода
EmptyLinesToLength	Отношение числа пустых строк к длине кода
whiteSpaceRatio	Соотношение между числом пробельных символов и числом не пробельных символов
newLineBeforeOpenBrace	Бинарное значение, определяющее предшествуют ли большинству фигурных скобок символы новой строки
tabsLeadLines	Бинарное значение, определяющее начинается ли большинство строк кода с отступов

Таблица 3. Набор «запахов кода»

Признак	Описание
longParamList	Среднее число параметров в методах классов
longMethod	Среднее количество строк в методах
longMethodShare	Отношение числа длинных методов к общему числу методов
switchLeadStat	Отношение числа операторов Switch к общему числу условных операторов
featureEnvy	Отношение числа методов, обращающихся к данным другого класса чаще, чем к собственным, к общему числу методов
classMediator	Отношение числа классов, делегирующих большую часть методов другому классу, к общему числу классов
dataClass	Отношение числа классов, содержащих исключительно атрибуты и методы get и set, к общему числу классов

Признаки проблем кода являются показателями несоответствия кода парадигмам программирования и описывают совокупность отступлений авторов от стандартных принципов разработки ПО. Сюда же следует относить и комментирование кода в том случае, если число комментариев превышает определенный порог.

Особенность используемых наборов признаков состоит в том, что они могут быть вычислены в ходе анализа исходного кода. Код может быть некомпилируемым, неполным, содержащим синтаксические или программные ошибки. Недостатком является подверженность части лексических и структурных признаков сознательному контролю программиста.

Определенные признаки из представленных наборов могут оказаться малоинформативными для конкретной задачи идентификации автора исходного кода и негативно сказаться на разделяющей способности модели SVM. Этим обусловлена необходимость отбора информативных признаков до начала процесса обучения классификатора.

Фильтр быстрой корреляции начинает работу с полным множеством доступных для анализа признаков и использует меру симметричной неопределенности для определения зависимостей между признаками:

$$SU(X,Y) = 2\left[\frac{H(X) - H(X \mid Y)}{H(X) + H(Y)}\right] = SU(Y,X), \quad (1)$$

где *H(X), H(Y)* — энтропии случайных величин, имеющих *i* и *j* состояний, *H(X|Y)* — условная энтропия:

$$H(X \mid Y) = -\sum_j P(y_j) \sum_i P(x_i \mid y_i) \log_2 (P(x_i \mid y_j)), \quad (2)$$

где $P(x_i)$, $P(y_i)$ — априорные вероятности для всех значений X и Y, $P(x_i \mid y_j)$ — апостериорная вероятность X при известной Y.

Чем ближе значение *SU* к нулю, тем ниже зависимость признаков друг от друга. Путем исключения малоинформативных признаков производится поиск подмножества, лучше всего описывающего предметную область.

Перед подачей на вход классификатора SVM тензоры, сформированные на основе выделенных из обучающей выборки и анонимного исходного кода информативных признаков, были нормализованы.

Принцип работы SVM состоит в построении гиперплоскости в пространстве признаков высокой размерности таким образом, чтобы зазор между опорными векторами, крайними точками двух классов, был максимальным. Отображение исходных данных в пространство,

где разделяющая их поверхность будет линейной, осуществляется с помощью ядрового преобразования:

$$\left(\Phi(x),\Phi(x')\right)=k\left(x,x'\right), \tag{3}$$

где $\left(\Phi(x),\Phi(x')\right)$ — скалярное произведение между распознаваемым образцом и тренировочными образцами, k — некоторое отображение исходного пространства в пространство со скалярным произведением (пространство достаточной для линейной разделимости размерности).

Тогда функция, выполняющая классификацию, выглядит следующим образом:

$$f(x)=\left\{\sum_{i=1}^{l}\alpha_i y_i k(x_i,x)\right\}+b, \tag{4}$$

где α — оптимальный коэффициент, k — ядро, y — метка принадлежности некоторому классу, b — параметр, обеспечивающий выполнение второго условия Каруша — Куна — Такера для всех входных образцов, которые соответствуют множителям Лагранжа, лежащим не на границах.

Оптимальный коэффициент α определяется максимизацией целевой функции:

$$W(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}y_i y_j\alpha_i\alpha_j k(x_i,x_j), \tag{5}$$

где условие максимизации:

$$\sum_{i=1}^{l}\alpha_i y_i=0, \tag{6}$$

в положительном квадранте $0\le\alpha_i\le C$, $i=\overline{1,l}$.

Параметр регуляризации C определяет соотношение между количеством ошибок обучающего множества и величиной зазора.

Реализация описанной модели осуществлялась с помощью библиотеки LIBSVM [11], позволяющей обучать, проводить классификацию и оценивать эффективность SVM-модели.

Параметры для обучения классификатора SVM выбирались исходя из рекомендаций [12]:

– тип SVM — многоклассовая классификация;

– алгоритм обучения — метод последовательной оптимизации;

– ядро — сигмоидальное;

– параметр регуляризации $C = 1$;

– допустимый уровень ошибки — 0,00001.

Обучение модели производилось методом последовательной оптимизации (SMO). На каждом шаге данного алгоритма решается минимально возможная задача — SMO выбирает два множителя Лагранжа для совместной оптимизации, находит оптимальные значения и обновляет SVM для получения новых оптимальных значений на входе. Преимущество выбранного алгоритма состоит в возможности аналитической оптимизации двух множителей Лагранжа без помощи численных методов квадратичного программирования.

Для многоклассовой классификации применялась стратегия выбора решения «каждый против каждого». Такая стратегия строит классификаторы для каждой пары классов, а к полученному в итоге множеству ответов применяет мажоритарное голосование — класс, выбранный большинством классификаторов, является конечным решением.

Обученная на основе информативных признаков модель SVM использовалась в дальнейшем для решения практических задач идентификации автора исходного кода. Результатом работы модели являлось принятое множеством бинарных классификаторов решение относительно принадлежности анонимного исходного кода к одному из классов обучающей выборки.

3. Методика на основе гибридной нейронной сети. Глубокое обучение [13-17] моделирует абстрактное мышление человека с целью принятия решений относительно поставленных задач. Понятие «глубины» определяется как наличие в архитектуре более чем одного скрытого слоя. Современные архитектуры глубоких НС значительно превосходят традиционные методы МО во многих областях:

1. Обученные глубокие модели НС на практике оказываются проще, чем традиционные, что избавляет исследователей от необходимости выделения информативных признаков.

2. Гибкость глубоких моделей НС заключается в возможности их многократного применения, а также в беспрепятственном их дообучении без необходимости перезапуска.

3. Обучение глубоких моделей НС может происходить циклично по небольшим наборам данных — преимуществом является масштабируемость.

4. Процесс низкоуровневых вычислений, осуществляемый при обучении глубоких моделей НС, может быть распараллелен с помощью графических ускорителей (GPU) или тензорных процессоров (TPU), что позволяет уменьшить временные затраты.

Наиболее популярными архитектурами глубоких НС являются RNN, сверточные НС (CNN), а также гибридные НС (HNN), представляющие собой комбинации различных архитектур [14].

Обобщенная методика идентификации автора исходного кода на основе HNN представлена на рисунке 2.

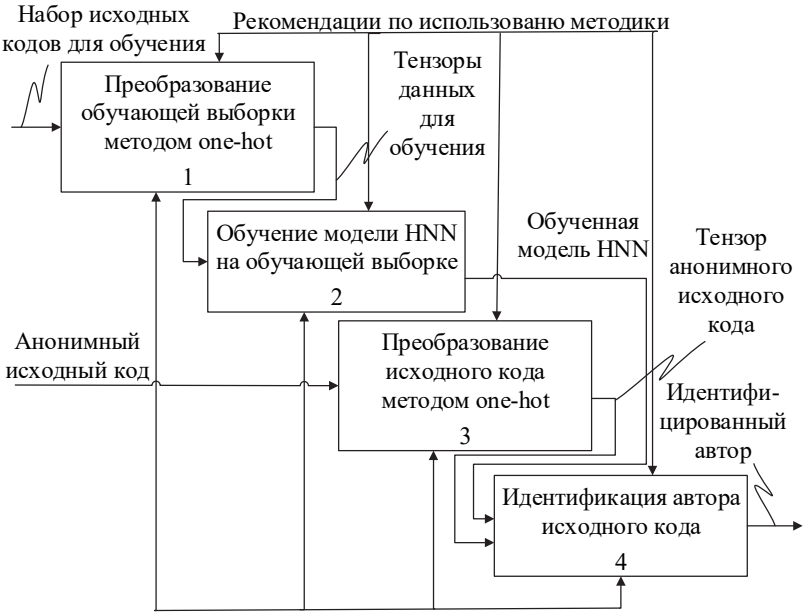

Рис. 2. Обобщенная методика идентификации автора исходного кода на основе HNN

Так как НС способны принимать на входе исключительно числовые тензоры различных рангов, возникла необходимость решения вопроса преобразования данных в тензорный вид.

С этой целью применялся метод прямого кодирования (one-hot encoding). Принцип его работы, представленный на рисунке 3, заключается в создании для каждого символа тензора первого ранга (вектора) из 256 нулей и единицы на позиции элемента с индексом, равным коду символа.

Данный метод использовался для преобразования в тензорный вид обучающей и тестовой выборок, а также анонимного исходного кода. Преобразование производилось на символьном уровне. Подлежащие преобразованию выборки были сформированы по следующим

параметрам: максимальная длина исходного кода — 500, максимальное количество исходных кодов одного автора — 20.

Символ	One-hot вектор
i	[105] 0,0,0,…,0,0,1,0,…,0,0,0,0
n	[110] 0,0,0,…,1,0,0,0,…,0,0,0,0
t	[116] 0,0,0,…,0,1,0,0,…,0,0,0,0
[space]	[32] 0,0,0,…,0,0,0,1,…,0,0,0,0
m	[109] 0,0,0,…,0,0,1,0,…,0,0,0,0
a	[97] 0,0,0,…,1,0,0,0,…,0,0,0,0
i	[105] 0,0,0,…,0,1,0,0,…,0,0,0,0
n	[110] 0,0,0,…,0,0,0,1,…,0,0,0,0

Рис. 3. Принцип работы one-hot кодирования

На полученных методом one-hot кодирования данных обучающей выборки было произведено обучение модели HNN.

В качестве инструмента для реализации архитектуры HNN выбрана библиотека для глубокого анализа данных Keras [18], предназначенная для создания и обучения практически любых архитектур нейросетей.

Выполнение низкоуровневых операций с тензорами и дифференцирования происходило с помощью наиболее распространенного бэкенда TensorFlow, обладающего высоким качеством и способностью к масштабированию.

При выборе архитектуры НС учитывались особенности исходных кодов как объекта исследования: лексические, синтаксические и семантические правила, парадигмы различных языков, которые могут так или иначе влиять на процесс идентификации программиста.

Исходный код представляет собой последовательность символов, что обуславливает необходимость применения рекуррентной архитектуры. Такие сети способны запоминать состояния, получаемые от обработки предыдущих элементов.

Авторские признаки исходных кодов, выявляемые НС при обучении, различны по размеру. Сверточные НС предназначены для распознавания локальных и глобальных признаков.

В общем случае выходное значение $h^{(t)} \in R^n$ скрытого слоя RNN на шаге обучения t можно представить как:

$$h^{(t)} = f(Wx^{(t)} + Uh^{(t-1)} + b^h), \quad (7)$$

где $x^{(t)} \in R^m$ — входной вектор в момент t, $W \in R^{mxn}$, $U \in R^{nxn}$, $b^h \in R^n$ — обучаемые параметры НС, f — функция активации.

Скрытый слой классической CNN формирует выходное значение (карту признаков) слоя l следующим образом:

$$h_j^l = f(\sum_i x_i^{l-1} * k_j^l + b_j^l), \quad (8)$$

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.

На рисунке 4 представлены следующие слои HNN:

– conv1D — одномерный сверточный слой. Может иметь различный масштаб для учета всех размерностей признакового пространства;

– concatenate — объединяющий слой. Применяется во избежание проблем с обработкой и обучением на новых данных;

– bidirectional(gru) — слой, объединяющий две независимые RNN: входная последовательность подается в обычном порядке для одной сети и в обратном для другой. Выходы обеих сетей объединяются на каждом шаге;

– dropout — слой, противодействующий переобучению. При поступлении нового объекта входные сигналы отключаются с заданной вероятностью;

– dense — полносвязный слой для классификации данных.

В качестве рекуррентной составляющей HNN применялись двунаправленные управляемые рекуррентные нейроны (BiGRU) [19] — вариация RNN, учитывающие состояния как предыдущих шагов во времени, так и будущих.

Сверточная составляющая HNN реализована сетью GoogleNet, также известной как Inception-v1 [20]. Особенность данной архитектуры заключается в параллельной работе слоев со свертками разных размерностей для дальнейшей их конкатенации в конечный результат.

Архитектура HNN, представленная на рисунке 1, не нуждается в предварительном определении набора авторских признаков. Более того, сверточно-рекуррентная НС способна находить информативные признаки самостоятельно.

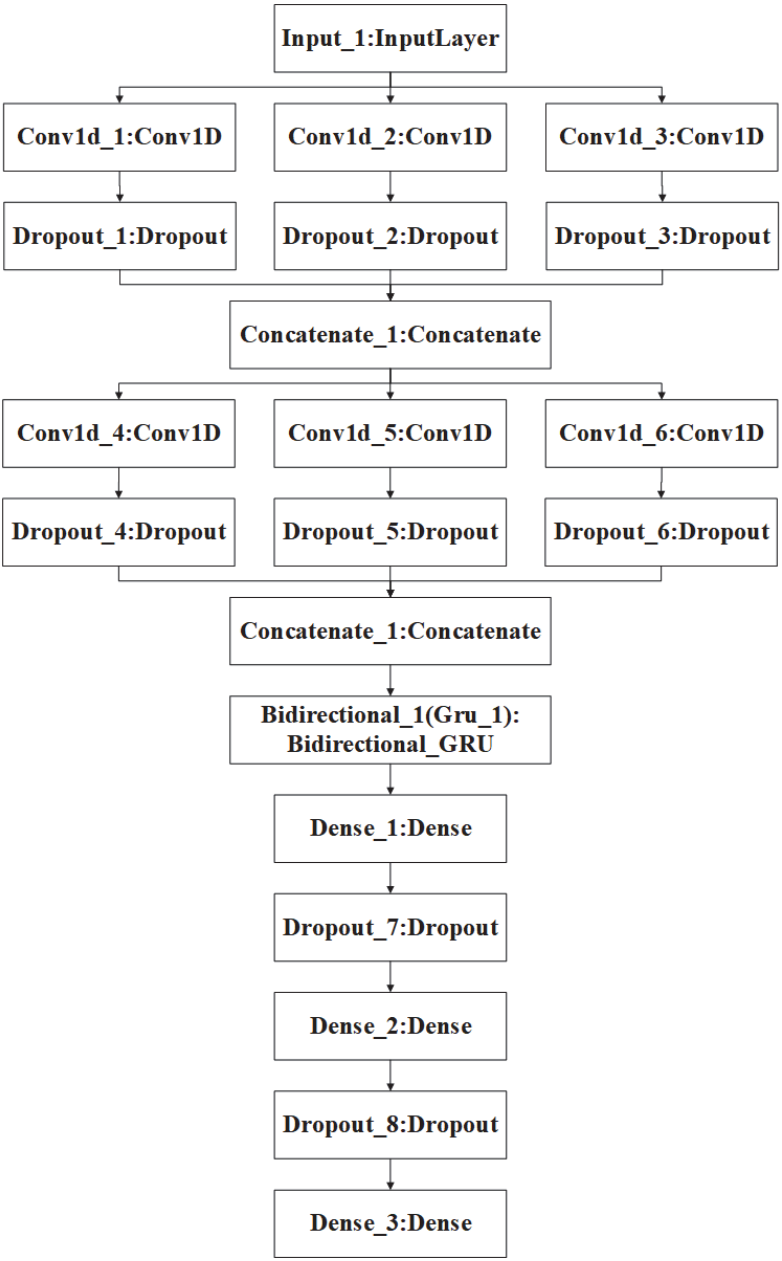

Рис. 4. Модель гибридной нейронной сети

Подбор характеристик и параметров НС является важнейшей процедурой, определяющей дальнейший ход обучения и конечный результат.

Минимизация ошибки, получаемой при обучении модели НС, происходит в процессе оптимизации смещений внутренних параметров и синаптических весов. Для этого использовался метод адаптивного шага обучения (Adadelta) [21], учитывающий историю значений градиента и изменения весов следующим образом:

$$\theta_{t+1} = \theta_t - \frac{RMS[\triangle\theta]_{t-1}}{RMS[g]_t} g_t , \tag{9}$$

где $RMS[\triangle\theta]_t$ — среднеквадратичная ошибка обновления параметров, g_t — градиент на момент *t*, θ_t — параметры сети на момент *t*, а среднеквадратичная ошибка обновления параметров определяется как:

$$RMS[\triangle\theta]_t = \sqrt{E[\triangle\theta^2]_t + \varepsilon} , \tag{10}$$

где $E[\triangle\theta^2]_t$ — скользящее среднее на момент *t*, ε — сглаживающий параметр.

Преимущество выбранного оптимизатора заключается в автоматической настройке скорости обучения без ее уменьшения. Результат работы НС должен иметь эффективное ограничение, то есть соответствующее вероятностное распределение на выходе последнего слоя.

Такое распределение было определено функцией активации Softmax [22] — обобщенной логистической функцией для многомерного случая, позволяющей трактовать выходные значения нейронов как вероятность принадлежности целевому классу.

Активатор Softmax позволяет избежать проблему затухания градиентов, свойственную алгоритму обратного распространения ошибки при его использовании в глубоком обучении:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} , \tag{11}$$

где σ — стандартная экспоненциальная функция, z_i — элемент входного вектора *z*, *k* — количество целевых классов.

По формуле (11) вычисляется экспонента входного значения и сумма экспоненциальных значений всех значений на входах. Данное отношение определяет выход функции Softmax. Координаты сформированного вектора являются вероятностями принадлежности объекта классу *i*. Вектор-столбец *z* при этом рассчитывается как:

$$z = \omega^T x - \theta \,, \tag{12}$$

где x — вектор-столбец признаков объекта размерности $\boldsymbol{M}\times 1$, ω^T — транспонированная матрица весовых коэффициентов признаков, имеющая размерность $\boldsymbol{K}\times\boldsymbol{M}$ — вектор-столбец с пороговыми значениями размерности $\boldsymbol{K}\times 1$, где $\boldsymbol{K}$ — количество классов объектов, а $\boldsymbol{M}$ — количество признаков объектов.

Согласно поставленной задаче и выбранной функции активации, была подобрана функция потерь. В данном случае использована categorical crossentropy, характеризующая потери при неправильном принятии решений на основании полученных данных.

Процедура регуляризации *i*-го нейрона происходила с помощью прореживания (dropout):

$$O_i = X_i a\left(\sum_{k=1}^{d_i} \omega_k x_k + b\right), \tag{13}$$

где X_i — случайная величина, k — всевозможные выходные значения, a — функция активации.

Таким образом, прореживание обнуляет часть элементов вектора и позволяет предотвратить процесс переобучения.

Обученная на основе рассмотренных характеристик и параметров модель НС применялась для решения практических задач идентификации автора исходного кода. Результатом работы модели являлось принятое на основе распределения вероятностей решение относительно принадлежности анонимного исходного кода к одному из классов обучающей выборки.

4. Результаты экспериментов. Процесс обучения любой модели и дальнейшая ее оценка подразумевают использование репрезентативных данных. Их сбор был произведен с крупнейшего веб-сервиса для хостинга IT-проектов GitHub [23]. Были отобраны языки, входящие в десятку наиболее популярных языков программирования.

Информация о корпусе представлена в таблице 4.

Таблица 4. Наборы данных для эксперимента

Язык	Количество кодов в наборе	Количество авторов в наборе	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

В исследовании был проведен эксперимент, направленный на выявление наиболее эффективной в рамках поставленной задачи архитектуры рекуррентной части НС. Выбор осуществлялся между современными моделями: GRU (управляемые рекуррентные блоки) и BiGRU. Популярные архитектуры LSTM и BiLSTM не были рассмотрены, так как они включают в себя больше фильтров и операций, чем GRU и BiGRU, а следовательно, требуют больше вычислительных ресурсов. Эксперимент был проведен на корпусе авторов, пишущих на языке C++. Его результаты приведены в таблице 5.

Таблица 5. Оценка точности GRU и BiGRU архитектур

Архитектура	5 авторов	10 авторов	20 авторов
GRU	83,4%	81%	79,8%
BiGRU	92%	91,5%	90%

Архитектура GRU решает проблему потери информации. Входной фильтр такой сети определяет, сколько информации из предыдущего слоя будет храниться в нейроне. Выходной фильтр определяет, сколько информации получат следующие слои. Полученный результат свидетельствует о недостаточности хранящихся в ячейках памяти GRU предыдущих шагов для эффективной идентификации.

Наибольшая точность идентификации была достигнута на модели HNN совместно с BiGRU. Такой результат обусловлен способностью BiGRU сети маркировать каждый элемент последовательности на основе не только прошлого, но также будущего контекста.

Предложенные архитектуры HNN и SVM были экспериментально оценены на наборе данных (таблица 4). Результаты экспериментов для модели SVM — таблица 6, для HNN — таблица 7.

Таблица 6. Оценка точности SVM-классификатора

Язык	5 авторов	10 авторов	20 авторов
C++	88,1%	87,6%	87%
Java	90,5%	90%	86,2%
JS	89%	87,4%	80%
Python	96%	90,1%	85,2%
C	91,2%	86,6%	85%
C#	90%	87%	82,5%
Ruby	88,5%	82,7%	76%
PHP	91%	86%	84,3%

Таблица 7. Оценка точности HNN-классификатора

Язык	5 авторов	10 авторов	20 авторов
C++	92%	91,5%	90%
Java	97,2%	93%	88,7%
JS	91,5%	82,1%	76%
Python	95,2%	92%	91,6%
C	96,1%	95,3%	94%
C#	95,5%	87,6%	83,4%
Ruby	94,7%	81,7%	77%
PHP	92%	89%	85,9%

Для наиболее популярного языка программирования Java эксперимент был расширен — получены оценки для корпусов из 30, 40 и 50 авторов. На их основе были построены графики зависимости точности от количества авторов для HNN (рисунок 5) и для SVM (рисунок 6).

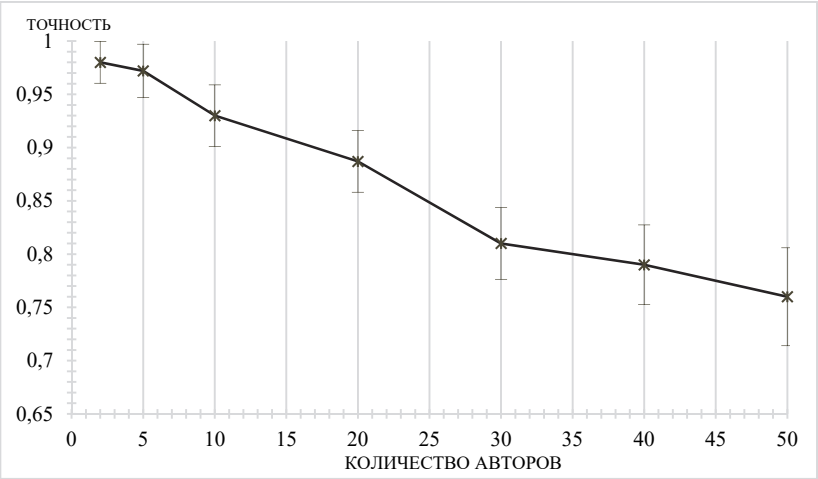

Рис. 5. График зависимости точности кросс-валидации HNN от количества авторов (для языка Java)

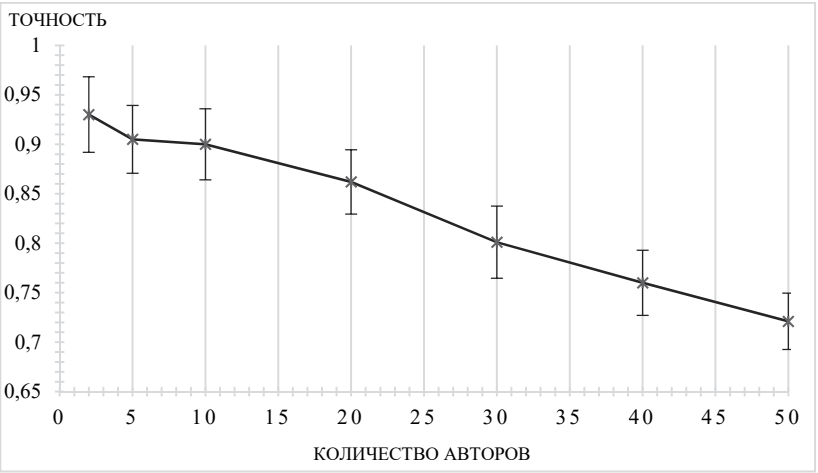

Рис. 6. График зависимости точности кросс-валидации SVM от количества авторов (для языка Java)

Графики демонстрируют постепенное падение точности при увеличении количества авторов, участвующих в эксперименте. Данная проблема может быть решена путем увеличения размера корпуса для каждого автора.

Можно сделать вывод, что обе методики продемонстрировали высокую точность для всех языков программирования. Особенности языка, на котором пишет программист влияют на результативность, но в целом обе методики являются универсальными. Лучшей точностью обладает глубокая HNN. Однако следует отметить, что среднее время обучения такой НС составляет ~900 с., что в десятки раз превышает время обучения SVM-классификатора.

Разработанные методики были также протестированы на корпусе исходных кодов программ по дисциплинам «Основы программирования», «Языки программирования», «Методы и технологии программирования» студентов Томского государственного университета систем управления и радиоэлектроники. В корпус вошли работы 75 студентов 2-3 курса факультета безопасности. Написанные ими программы реализуют решения однотипных задач на языке программирования C++, что исключает повышение разделяющей способности из-за специфичности функционального назначения программ. Выборка была ограничена 15 исходными кодами объемом не более 500 строк. Полученные в ходе эксперимента оценки точности приведены в таблице 8.

Таблица 8. Оценка точности на корпусе студентов ТУСУРа

Классификатор	5 авторов	10 авторов	20 авторов
HNN	93%	91%	86,1%
SVM	88,9%	84,2%	81%

Представленные результаты свидетельствуют о том, что предложенные походы являются эффективными для идентификации авторства кода не только опытных профессиональных программистов, но и начинающих — квалификация не оказывает влияния на точность классификации. Это говорит о перспективности применения методик в образовательном процессе для оценки работ по программированию.

В таблице 9 приведен обзор результатов разработанных методик и других современных работ, посвященных идентификации автора исходного кода.

Таблица 9. Сравнение подходов

Автор	Метод	Корпус	Точность	Язык
Yang X., Li Q., Guo Y. [1]	МРЧ, НС	40 авторов	91%	Java
Alsulami B., Dauber E. [2]	АСД, LSTM/BiLSTM	25 авторов	92/96%	Python
		10 авторов	80/85%	C++
Wisse W., Veenman C.J. [3]	АСД, SCAP, *n*-граммы, SVM	34 автора	80%	JS
Zia T., Ilyas M. [4]	АСД, «запахи кода», SVM	10 авторов	75%	Java
Caliskan-Islam A., Harang R. [5]	Нечеткие АСД, случайные деревья	50 авторов	90%	Python
Caliskan-Islam A., Dauber E. [6]	Нечеткие АСД, калибровочные кривые	106 авторов	73%	C++
Wang T., Ji S. [7]	Статические и динамические метрики	23 автора	94%	Python
Abuhamad M., AbuHmed T. [8]	RNN, random forest	50 авторов	95,2%	C++
Mohsen A.M., El-Makky N.M., [9]	SVM, *n*-граммы, автоэнкодер	50 авторов	95%	-
Куртукова А.В., Романов А.С.	SVM, фильтр быстрой корреляции, информативные признаки	5 авторов	96%	Python
		10 авторов	87,4%	JS
		20 авторов	85%	C
Куртукова А.В., Романов А.С.	Глубокая рекуррентно-сверточная НС (HNN)	5 авторов	97,2%	Java
		10 авторов	89%	PHP
		20 авторов	90%	C++

Сравнительный анализ демонстрирует, что предложенные методики позволили получить наилучшие результаты. Кроме того, они оказались высокоэффективными для языков программирования, не рассмотренных в научных трудах зарубежных исследователей — C, C#, Ruby, PHP.

Заключение. В данной статье были предложены две новых методики для идентификации автора исходного кода: на основе SVM, фильтра быстрой корреляции и информативных признаков и на основе гибридной нейронной сети HNN.

Предложенные методики показали лучшие результаты в сравнении с современными подходами, опробованными зарубежными исследователями.

Анализ результатов экспериментов продемонстрировал, что подход на основе HNN способен достигать точности 97%, что является наилучшим результатом на сегодняшний день. Традиционный метод классификации, реализованный на основе SVM, позволил добиться точности 96%. В среднем точность классификации HNN превосходит SVM на 5%.

Кроме того, следует учитывать тот факт, что на вход HNN подавались необработанные исходные коды, в то время как SVM обучалась на признаковом пространстве, сформированном экспертами вручную и преобразованном с помощью фильтра быстрой корреляции, однако часть из них может сознательно контролироваться программистом. Модель HNN оказалась способна находить новые закономерности и зависимости в данных, являющиеся неявными для исследователя, а также косвенно учитывать интеллектуальное содержание и особенности реализации программного кода.

Таким образом, HNN является устойчивой к намеренным изменениям исходного кода, а признаки, используемые для обучения SVM, напротив, вариативны для одного программиста, что делает возможным принятие контрмер для анонимизации исходных кодов.

Особенности языка программирования не оказали влияния на точность классификации — обе методики универсальны и могут применяться для идентификации авторов-программистов, независимо от предпочитаемого ими языка. Кроме того, методика на основе HNN продемонстрировала высокую точность идентификации на различных наборах исходных кодов и независимость от квалификации программиста.

В дальнейшем планируется продолжить исследования. В частности, извлечение решающих правил из обученной НС и их применение совместно с SVM, а также использование ансамбля классификаторов позволят повысить точность классификации и снизить временные затраты на обучение моделей.

Литература

1. *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. pp. e0187204.
2. *Alsulami B. et al.* 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.
3. *Wisse W., Veenman C.J.* Scripting DNA: Identifying the JavaScript Programmer // Digital Investigation. 2015. vol. 15. pp. 61–71.
4. *Gull M., Zia T., Ilyas M.* Source Code Author Attribution Using Author's Programming Style and Code Smells // International Journal of Intelligent Systems and Applications. 2017. vol. 9. no. 5. pp. 27–33.
5. *Caliskan-Islam A. et al.* De-anonymizing programmers via code stylometry // Proceedings of the 24th USENIX Security Symposium. 2015. pp. 255–270.

6. *Dauber E. et al.* Poster: Git Blame Who?: Stylistic Authorship Attribution of Small, Incomplete Source Code Fragments // 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). 2018. pp. 356–357.
7. *Wang N., Ji S., Wang T.* 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.
8. *Abuhamad M., AbuHmed T., Mohaisen A., Nyang D.* Large-Scale and Language-Oblivious Code Authorship Identification // Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018. pp. 101–114.
9. *Mohsen A.M., El-Makky N.M., Ghanem N.* Author Identification using Deep Learning // 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA). 2016. pp. 898–903.
10. *Созинова И.С., Романов А.С., Мещеряков Р.В.* Определение поискового спама с использованием метода опорных векторов // Труды СПИИРАН. 2014. Вып. 5(36). С. 78–91.
11. *Wu T.F., Lin C.J., Weng R.C.* Probability estimates for multi-class classification by pairwise coupling // Journal of Machine Learning Research. 2004. vol. 5. pp. 975–1005.
12. *Романов А.С., Шелупанов А.А., Мещеряков Р.В.* Разработка и исследование математических моделей, методик и программных средств информационных процессов при идентификации автора текста // Томск: В-Спектр. 2011. 188 с.
13. *LeCun Y., Bengio Y., Hinton G.* Deep learning // Nature. 2015. vol. 521. no. 7553. pp. 436–444.
14. *Schmidhuber J.* Deep Learning in Neural Networks: An Overview // Neural Networks. 2015. vol. 61. pp. 85–117.
15. *Zhang X., Zhao J., LeCun Y.* Character-level Convolutional Networks for Text Classification // Advances in neural information processing systems. 2015. pp. 649–657.
16. *Najafabadi M.M. et al.* Deep learning applications and challenges in big data analytics // Journal of Big Data. 2015. vol. 2. no. 1. pp. 21.
17. *Chen X.W., Lin X.* Big Data Deep Learning: Challenges and Perspectives // IEEE Access. 2014. vol. 2. pp. 514–525.
18. *Gulli A., Pal S.* Deep learning with Keras // Packt Publishing Ltd. 2017. 490 p.
19. *Fei H., Tan F.* Bidirectional Grid Long Short-Term Memory (BiGridLSTM): A Method to Address Context-Sensitivity and Vanishing Gradient // Algorithms. 2018. vol. 11. no. 11. pp. 172.
20. *Szegedy C. et al.* Going Deeper with Convolutions // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. pp. 1–9.
21. *Zeiler M.D.* Adadelta: an adaptive learning rate // arXiv preprint arXiv:1212.5701. 2012.
22. *Nwankpa C., Ijomah W., Gachagan A., Marshall S.* Activation Functions: Comparison of trends in Practice and Research for Deep Learning // arXiv preprint arXiv:1811.03378. 2018.
23. Github. URL: https://github.com/ (дата обращения: 20.02.2019).

Куртукова Анна Владимировна — студентка, кафедра безопасности информационных систем, Томский государственный университет систем управления и радиоэлектроники (ТУСУР). Область научных интересов: информационная безопасность, интеллектуальный анализ данных, искусственный интеллект, обработка текста. Число научных публикаций — 8. av.kurtukova@gmail.com; ул. Красноармейская, 146, 634034, Томск, Российская Федерация; р.т.: +7(905)9916713.

Романов Александр Сергеевич — канд. техн. наук, доцент, кафедры безопасности информационных систем, Томский государственный университет систем управления и радиоэлектроники (ТУСУР). Область научных интересов: информационная безопасность, интеллектуальный анализ данных, искусственный интеллект, обработка текста. Число научных публикаций — 50. alexx.romanov@gmail.com; ул. Красноармейская, 146, 634034, Томск, Российская Федерация; р.т.: +7 (382) 241-34-26.

DOI 10.15622/sp.2019.18.3.741-765

A.V. KURTUKOVA, A.S. ROMANOV

IDENTIFICATION AUTHOR OF SOURCE CODE BY MACHINE LEARNING METHODS

Kurtukova A.V., Romanov A.S. **Identification Author of Source Code by Machine Learning Methods.**

Abstract. The paper is devoted to the analysis of the problem of determining the source code author, which is of interest to researchers in the field of information security, computer forensics, assessment of the quality of the educational process, protection of intellectual property.

The paper presents a detailed analysis of modern solutions to the problem. The authors suggest two new identification techniques based on machine learning algorithms: support vector machine, fast correlation filter and informative features; the technique based on hybrid convolutional recurrent neural network.

The experimental database includes samples of source codes written in Java, C ++, Python, PHP, JavaScript, C, C # and Ruby. The data was obtained using a web service for hosting IT-projects – Github. The total number of source codes exceeds 150 thousand samples. The average length of each of them is 850 characters. The case size is 542 authors.

The experiments were conducted with source codes written in the most popular programming languages. Accuracy of the developed techniques for different numbers of authors was assessed using 10-fold cross-validation. An additional series of experiments was conducted with the number of authors from 2 to 50 for the most popular Java programming language. The graphs of the relationship between identification accuracy and case size are plotted. The analysis of result showed that the method based on hybrid neural network gives 97% accuracy, and it's at the present time the best-known result. The technique based on the support vector machine made it possible to achieve 96% accuracy. The difference between the results of the hybrid neural network and the support vector machine was approximately 5%.

Keywords: Source Code Writer, Deep Learning, Neural Network, SVM, HNN.

Kurtukova Anna Vladimirovna — Student, Information System Security of Security Department, Tomsk State University of Control Systems and Radioelectronics (TUSUR). Research interests: information security, machine learning, text mining. The number of publications — 8. av.kurtukova@gmail.com; 146, Krasnoarmejskaja, 634034, Tomsk, Russian Federation; office phone: +7(905)9916713.

Romanov Alexander Sergeevich — Ph.D., Associate Professor, Information System Security Department, Tomsk State University of Control Systems and Radioelectronics (TUSUR). Research interests: : information security, data mining, artificial intelligence, word processing. The number of publications — 50. alexx.romanov@gmail.com; 146, Krasnoarmejskaja, 634034, Tomsk, Russian Federation; office phone: +7 (382) 241-34-26.

References

1. 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. pp. e0187204.
2. Alsulami B. et al. 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.
3. Wisse W., Veenman C.J. Scripting DNA: Identifying the JavaScript Programmer. *Digital Investigation*. 2015. vol. 15. pp. 61–71.

4. Gull M., Zia T., Ilyas M. Source Code Author Attribution Using Author's Programming Style and Code Smells. *International Journal of Intelligent Systems and Applications.* 2017. vol. 9. no. 5. pp. 27–33.
5. Caliskan-Islam A. et al. De-anonymizing programmers via code stylometry. Proceedings of the 24th USENIX Security Symposium. 2015. pp. 255–270.
6. Dauber E. et al. Poster: Git Blame Who?: Stylistic Authorship Attribution of Small, Incomplete Source Code Fragments. 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). 2018. pp. 356–357.
7. Wang N., Ji S., Wang T. 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.
8. Abuhamad M., AbuHmed T., Mohaisen A., Nyang D. Large-Scale and Language-Oblivious Code Authorship Identification. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018. pp. 101–114.
9. Mohsen A.M., El-Makky N.M., Ghanem N. Author Identification using Deep Learning. 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA). 2016. pp. 898–903.
10. Sozinova I.S., Romanov A.S., Meshcheryakov R.V. [Search Spam Identification Using Support Vector Machine]. *Trudy SPIIRAN – SPIIRAS Proceedings.* 2014. vol. 5(36). pp. 78–91. (In Russ.).
11. Wu T.F., Lin C.J., Weng R.C. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research.* 2004. vol. 5. pp. 975–1005.
12. Romanov A.S., Shelupanov A.A, Meshcheryakov R.V. *Razrabotka i issledovanie matematicheskih modelej, metodik i programmnyh sredstv informacionnyh processov pri identifikacii avtora teksta* [Development and research of mathematical models, methods and software tools of information processes in the identification of the author of the text]. Tomsk: V-Spektr. 2011. 188 p. (In Russ.).
13. LeCun Y., Bengio Y., Hinton G. Deep learning. *Nature.* 2015. vol. 521. no. 7553. pp. 436–444.
14. Schmidhuber J. Deep Learning in Neural Networks: An Overview. *Neural Networks.* 2015. vol. 61. pp. 85–117.
15. Zhang X., Zhao J., LeCun Y. Character-level Convolutional Networks for Text Classification. *Advances in neural information processing systems*. 2015. pp. 649–657.
16. Najafabadi M.M. et al. Deep learning applications and challenges in big data analytics. *Journal of Big Data.* 2015. vol. 2. no. 1. pp. 21.
17. Chen X.W., Lin X. Big Data Deep Learning: Challenges and Perspectives. *IEEE Access*. 2014. vol. 2. pp. 514–525.
18. Gulli A., Pal S. Deep learning with Keras. Packt Publishing Ltd. 2017. 490 p.
19. Fei H., Tan F. Bidirectional Grid Long Short-Term Memory (BiGridLSTM): A Method to Address Context-Sensitivity and Vanishing Gradient. *Algorithms.* 2018. vol. 11. no. 11. pp. 172.
20. Szegedy C. et al. Going Deeper with Convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. pp. 1–9.
21. Zeiler M.D. Adadelta: an adaptive learning rate. arXiv preprint arXiv:1212.5701. 2012.
22. Nwankpa C., Ijomah W., Gachagan A., Marshall S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. arXiv preprint arXiv:1811.03378. 2018.
23. Github. Available at: https://github.com/ (accesed: 20.02.2019).