УДК 004.4(075.8) https://doi.org/10.15407/pp2019.04.092

О.Г. Мороз, Г.Б. Мороз

ПРИМЕНЕНИЕ МАШИННОГО ОБУЧЕНИЯ В ПРОГРАММНОЙ ИНЖЕНЕРИИ: ОБЗОР

В статье дается краткое описание некоторых методов машинного обучения, таких как искусственные нейронные сети, машины опорных векторов, деревья решений, индуктивное логическое программирование, CBR и других, приводятся примеры применения этих методов для решения проблем прогнозирования и оценки качества в программной инженерии, даются общие рекомендации по применению алгоритмов машинного обучения к задачам программной инженерии. Обзор будет полезен исследователям и практикам в качестве отправной точки, поскольку он обеспечивает важные будущие направления исследований. Это в конечном итоге приведет к более эффективному принятию решений в программной инженерии, тем самым обеспечивая лучшие, более надежные и экономически эффективные программные продукты.

Ключевые слова: программная инженерия, программное обеспечение, машинное обучение, нейронные сети, машины опорных векторов, деревья решений.

Введение

Сегодня достижение успеха в промышленной, военной, коммерческой, социальной и многих других сферах деятельности человека немыслимо без применения различных компьютерных систем, функционирование которых невозможно без соответствующего программного обеспечения (ПО), ставшего основной технологией для развития современного общества и являющегося неотъемлемой, но небезопасной частью повседневной жизни. Человечество знает множество неприятных историй, произошедших из-за проблем с ПО, к примеру, неудачная посадка спускаемого модуля на Марс в 1999 г. обошлась США потерей $165 млн. или нецелевой запуск ракет «Патриот» в системе обороны США в 1991 г., что привело к гибели мирного населения. Поэтому в настоящее время проблема качества – это главная проблема для мировой индустрии ПО. Аспектами повышения качества и надежности программных продуктов на всех этапах их жизненного цикла, начиная с подготовительных работ и заканчивая снятием их из обращения, занимается отдельная дисциплина – программная инженерия (software engineering, **SE**).

Основные проблемы разработки и поддержки больших программных систем в изменяющейся среде были изложены в классической статье Ф. Брукса [1] еще в далеком 1987 году. В ней же отмечалось, что многие из классических проблем разработки программных продуктов обусловлены их существенной сложностью, которая нелинейно увеличивается с ростом размера ПО. Спустя более 30 лет разнообразие, размеры и сложность ПО многократно увеличились, что привело к появлению множества новых чрезвычайно сложных проблем, возникающих на всех этапах его жизненного цикла [2]. Уже в конце XX столетия возникла острая необходимость в поиске альтернативы существующим технологиям разработки и анализа ПО. Обнадеживающие результаты в преодолении некоторых из возникающих трудностей дало применение в программной инженерии методов искусственного интеллекта [3–8]. В своей основной речи на ежегодной конференции Американской ассоциации искусственного интеллекта в 1992 году О. Селфридж призывал к активному использованию для решения проблем программной инженерии методов машинного обучения (Machine Learning, **ML**) – одной из важных подобластей искусственного интеллекта [9].

Последние два десятилетия показали, что многие задачи по разработке и сопровождению ПО могут быть сформулированы как проблемы обучения и алгоритмы ML являются жизнеспособной альтернативой существующим подходам к реше-

нию многим проблем SE. Чтобы лучше использовать методы ML в качестве инструментов для решения реальных проблем SE, необходимо знать, как эти проблемы, так и условия, при которых имеющиеся в нашем распоряжении методы ML могут быть наиболее эффективно применены.

Остальная часть статьи организована следующим образом: первый раздел посвящен краткому обзору наиболее важных, с точки зрения SE, методов ML; во втором разделе рассматриваются вопросы применения ML для решения проблем прогнозирования и оценки качества ПО; в третьем разделе даются некоторые рекомендации по применению алгоритмов ML к задачам SE, далее приведены основные выводы.

1. Подходы к машинному обучению

В качестве подобласти искусственного интеллекта ML имеет дело с тем, как создавать компьютерные программы, которые улучшают свою производительность при выполнении определенной задачи с помощью опыта [8]. Поскольку многие задачи разработки или сопровождения SE зависят от некоторой функции (или функций, отображений или моделей) для прогнозирования, оценки, классификации, диагностики, обнаружения, приобретения, понимания, генерации или преобразования определенного качественного или количественного аспекта программного артефакта или программного процесса, применение ML в SE сводится к тому, как в процессе обучения найти такую целевую функцию (отображение или модель), которую можно использовать для выполнения задач SE.

В зависимости от используемых методов обучения целевая функция может быть представлена в различных формализмах языка гипотез (например, деревья решений, битовые строки или правила). Когда целевая функция не определена явно, но учащийся может сгенерировать ее значения для заданных входных запросов (например, в случае обучения на основе экземпляров), то говорят, что функция определена неявно.

При изучении целевой функции на заданном наборе обучающих данных ее обобщение может быть либо *усердным* (на этапе обучения), либо *ленивым* (на этапе классификации). Усердное обучение может создать одну целевую функцию из всех данных обучения, в то время как ленивое обучение может принимать разные (неявные) функции для каждого запроса. Оценка целевой функции зависит от многих факторов: точности прогнозирования, интерпретируемости, статистической значимости, содержания информации и компромисса между ее сложностью и степенью соответствия данным [10].

Алгоритмы обучения в зависимости от чувствительности к изменениям данных обучения делятся на стабильные и нестабильные [11]. Для нестабильных алгоритмов небольшие изменения в обучающих данных приведут к тому, что алгоритмы будут генерировать существенно различную выходную функцию. С другой стороны, стабильные алгоритмы невосприимчивы к небольшим изменениям в данных [11].

Существует четыре основных типа машинного обучения [8, 12, 13]:

- ***Контролируемое обучение*** **(**supervised learning**):** все входные и выходные данные помечены, и алгоритмы учатся предсказывать выходные данные из входных данных, т. е. целевая функция изучается на основе обучающих примеров ее входов и выходов.

- ***Неконтролируемое обучение*** (unsupervised learning): все данные не помечены, и алгоритмы учатся присваивать структуру из входных данных. Пытается изучить шаблоны на входе, для которых нет доступных выходных значений. Примером является кластеризация.

- ***Частично контролируемое обучение*** (semi-supervised learning): некоторые данные помечены, но большинство из них немаркированы, (данные обучения включают в себя несколько желаемых результатов.) Может использоваться смесь контролируемых и неконтролируемых методов. Хорошим примером является фотоархив, в котором помечены только некоторые

изображения (например, собака, кошка, человек), а большинство изображений не имеет маркировки.

- ***Обучение с подкреплением*** (reinforcement learning): это наиболее общая форма обучения. В нем рассматривается вопрос о том, как выучить последовательность действий, называемых стратегией управления, из косвенной и отложенной информации о вознаграждении.

Контролируемое обучение является наиболее зрелым и изученным типом обучения. Подавляющее большинство используемых в SE методов ML принадлежат этому типу обучения. Далее приводится краткое описание основных методов обучения, имеющих отношение к данной работе, в следующих группах:

- деревья решений (decision trees, **DT**);
- нейронные сети (neural networks, **NN**);
- обучение на основе экземпляров (instance-based learning, **IBL**);
- индуктивное логическое программирование (inductive logic programming, **ILP**);
- машины опорных векторов (support vector machines, **SVM**);
- генетические алгоритмы (Genetic Algorithms, **GA**);
- генетическое программирование (genetic programming, **GP**);
- байесовское обучение (Bayesian learning, **BL**);

Организация различных методов обучения в группы в значительной степени находится под влиянием [8, 11].

Деревья решений

Деревья решений, предложенные в [14, 15], состоят из листьев и ветвей, где листья представляют классификации, а ветви – соединения признаков, которые приводят к классификациям. Таким образом, DT являются инструментом моделирования принятия решений, который графически отображает процесс классификации заданных входных данных для заданных меток выходных классов. Внутренние узлы DT обозначают различные атрибуты, а ветви между узлами представляют возможные значения, которые эти атрибуты могут иметь в наблюдаемых выборках.

В алгоритмах DT целевая функция определяется как дерево решений. Поиск в DT часто управляется мерой прироста информации, основанной на энтропии, которая показывает, сколько информации дает тест по атрибуту. Алгоритмы обучения DT часто имеют тенденцию к небольшим деревьям. Это усердный, контролируемый и нестабильный метод обучения, подверженный шумным данным, что является причиной переобучения. Он не может использовать априорные знания в процессе обучения. Тем не менее, он хорошо масштабируется с большими данными несколькими различными способами [11].

Деревья решений являются одним из самых популярных алгоритмов классификации, применяемых в интеллектуальном анализе данных и машинном обучении. Наиболее востребованными алгоритмами DT являются такие как ID3 (*iterative dichotomiser* 3), C4.5, C5, J48 и CART (*classification and regression trees*) [16–19].

Нейронные сети

Каждая из миллиардов клеток нейронов человеческого мозга подобна крошечному компьютеру с чрезвычайно ограниченными возможностями. Искусственные NNs формируются из сотен или тысяч симулированных нейронов, соединенных вместе подобно нейронам в мозге [20]. Модель нейрона показана на рисунке.

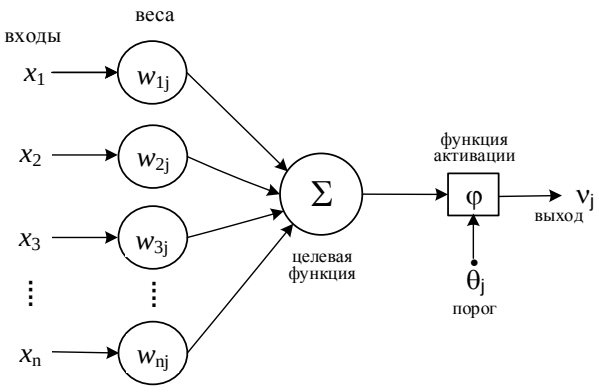

Рисунок. Нелинейная модель нейрона

Нейронные сети, являясь мощным инструментом анализа данных, способна фиксировать и представлять сложные отношения ввода/вывода. Мотивацией для разработки технологии NN является создание искусственной системы, которая могла бы выполнять «интеллектуальные» задачи, подобные тем, которые выполняются человеческим мозгом. NNs подобно человеческому мозгу приобретают знания через обучение и хранят их в пределах межнейронных связей, известных как синаптические веса [21].

Истинная сила и преимущество NN заключается в их способности представлять, как линейные, так и нелинейные отношения и изучать эти отношения непосредственно из моделируемых данных [22]. Традиционные линейные модели просто неадекватны, когда речь идет о данных моделирования, содержащих нелинейные характеристики.

Учитывая фиксированную структуру NN, обучение ее целевой функции равнозначно нахождению вектора весов соединений, минимизирующего сумму квадратов ошибок, применяемых ко всем наборам данных. Поскольку вектор весов по существу определяет целевую функцию, то это делает ее очень трудной для чтения и интерпретации человеком.

Нейронные сети – это усердный, контролируемый и нестабильный подход к обучению, который не может использовать предварительные знания. Популярным алгоритмом для сетей прямой связи является Backpropagation, который принимает поиск с градиентным спуском и санкционирует индуктивное смещение гладкой интерполяции между точками данных [8]. Среди других архитектур NN, применяемых в SE, выделим следующие: многослойный персептрон (multilayer perceptron, MLP) [23], сеть радиально базисных функций (radial basis function network, RBFN) [24], вероятностная нейронная сеть (probabilistic neural network, PNN) [25], сеть Элмана (Elman network, EN) [26], «самоорганизующаяся» каскадно-корреляционная нейронная сеть (Cascade correlation neural network, CCNN) [27], карта самоорганизующихся объектов (self-organising feature map, SOM) [28].

Обучение на основе экземпляров

В машинном обучении IBL представляет собой семейство алгоритмов обучения, которые вместо выполнения явного обобщения сравнивают новые проблемные экземпляры с экземплярами, наблюдаемыми в процессе обучения, хранящиеся в памяти. По этой причине IBL иногда называют обучением на основе памяти [29]. IBL – это типичный метод ленивого обучения в том смысле, что обобщение за пределы обучающих данных откладывается до классификации невидимого случая. Кроме того, целевая функция явно неопределена; вместо этого учащийся возвращает значение целевой функции при классификации данного невидимого случая. Значение целевой функции генерируется на основе подмножества обучающих данных, считающихся локальными для невидимого примера, а не на всех обучающих данных. Это равносильно аппроксимации другой целевой функции для отдельного невидимого примера. Это значительный отход от методов усердного обучения, когда одна целевая функция получается в результате того, что учащийся обобщает все данные обучения. Процесс поиска основан на статистических рассуждениях и состоит в определении данных обучения, которые близки к данному невидимому случаю, и создании значения целевой функции на основе его соседей. Одним из преимуществ IBL по сравнению с другими методами ML является его способность адаптировать свою модель к ранее невидимым данным. Учащиеся на основе экземпляров могут просто сохранить новый экземпляр или выбросить старый.

Наиболее известными алгоритмами IBL являются алгоритмы k-ближайших соседей (K-nearest neighbors) [30] и рассуждения по прецедентах (Case-Based Reasoning, CBR) [31, 32].

Индуктивное логическое программирование

Индуктивное логическое программирование впервые упоминается в статье [33], где «индуктивное» употребляется в философском (предложение теории для объяснения наблюдаемых фактов), а не в

математическом (доказательство свойства членов множества) смысле. Сегодня ILP – раздел машинного обучения, который использует логическое программирование как форму представления примеров, фоновых знаний и гипотез. Получив описания уже известных фоновых знаний и набор примеров, представленных как логическая база фактов, система ILP может породить логическую программу в форме гипотез, объясняющую все положительные примеры и ни одного отрицательного. Поскольку целевая функция в ILP определяется набором правил (пропозициональных или первого порядка), она легко поддается восприятию и интерпретации человеком. ILP позволяет использовать фоновые знания в процессе обучения и является усердным и контролируемым обучением. Наиболее известными алгоритмами ILP являются FOIL [34, 35] и PROGOL [36].

Машины опорных векторов

Предложенный В.Н. Вапником метод SVM [37] представляет собой набор схожих алгоритмов обучения, использующихся для задач классификации и регрессионного анализа. Принадлежит семейству линейных классификаторов и может также рассматриваться как специальный случай регуляризации по А.Н. Тихонову. Являясь не вероятностным двоичным линейным классификатором, SVM, вместо того, чтобы изучать нелинейную целевую функцию непосредственно из данных во входном пространстве, использует функцию ядра (определенную в форме внутреннего произведения обучающих данных) для преобразования обучающих данных из входного пространства сначала в пространство объектов большого размера F, а затем изучает оптимальный линейный разделитель (гиперплоскость) в F. Функцию выбора решения, определенную на основе линейного разделителя, можно использовать для классификации невидимых случаев. Функция ядра играет ключевую роль в SVM. Она основывается только на подмножестве обучающих данных, называемых опорными векторами.

Модели SVM тесно связаны с искусственными NN [38, 39]. Поэтому SVM можно использовать в качестве альтернативного метода обучения для полиномиальной, радиальной базисной функции и многослойных сетей персептрона с использованием функции ядра [40].

Генетические алгоритмы и генетическое программирование

GA был разработан в [41] как альтернативный метод решения общих задач оптимизации с большими поисковыми пространствами. Он имеет то преимущество, что ему не нужны предварительные знания, опыт или логика, связанные с конкретным решением проблемы. Основные идеи GA основаны на дарвиновской теории эволюции, которая в сущности говорит о том, что генетические операции между хромосомами особей в конечном итоге приводят к выживанию сильнейших. Таким образом, в течение длительного периода времени улучшается популяция особей в целом. Целевая функция представляется в виде битовых строк. Процесс поиска начинается с совокупности исходных гипотез. Посредством операций кроссинговера и мутации особи текущей популяции образуют новую популяцию с лучшими средними характеристиками. На каждом шаге итерации гипотезы в текущей популяции оцениваются с учетом заданной степени пригодности, причем наиболее подходящие гипотезы выбираются в качестве членов следующего поколения. Процесс поиска заканчивается, когда некоторая гипотеза имеет значение пригодности выше некоторого порога. Таким образом, процесс обучения по существу воплощен в поиске по пучку траектории с порождением и проверкой вариантов [8].

Предложенное в [42] GP в машинном обучении используется для автоматического создания или изменения программ с помощью GA. Выбор способа кодирования программы в GA – один из основных вопросов в GP. Программа должна быть закодирована в таком виде, чтобы легко было автоматически вносить случайные изменения (оператор мутации) и объединять два алгоритма в один (оператор скрещивания). Различают два способа кодирования: прямое и косвенное. В первом

случае GA работает с программой в явном виде, а во втором – GA работает не с самим кодом программы, а с правилами его построения, то есть GA работает с программой, генерирующая нужную нам программу.

Байесовское обучение

BL предлагает вероятностный подход к выводу, который основан на предположении, что интересующие величины определяются распределением вероятностей, и что оптимальные решения или классификации могут быть достигнуты путем рассуждения об этих вероятностях наряду с наблюдаемыми данными [8]. Методы BL на основе результатов учащегося могут быть разделены на две группы: те, которые дают наиболее вероятную гипотезу с учетом данных обучения, и те, которые производят наиболее вероятную классификацию нового экземпляра с учетом данных обучения. Таким образом, целевая функция явно представлена в первой группе, но неявно определена во второй группе. Одним из основных преимуществ является то, что BL учитывает предшествующее знание (в форме байесовских сетей доверия, априорных вероятностей для гипотез-кандидатов или распределения вероятностей по наблюдаемым данным для возможной гипотезы). Классификация невидимого случая получается посредством комбинированных предсказаний нескольких гипотез. Это также хорошо масштабируется с большими данными. BL – усердный и контролируемый метод обучения, который не требующий поиска в процессе обучения. Хотя BL не имеет проблем с зашумленными данными, у него возникают проблемы с небольшими наборами данных. BL принимает смещение, основанное на принципе минимальной длины описания, который предпочитает гипотезу h, которая минимизирует длину описания h плюс длину описания данных, заданных h [8].

Среди популярных алгоритмов BL можно выделить следующие: максимальный апостериорный (maximum a posteriori (MAP)), оптимальный байесовский классификатор (Bayes optimal classzjier), наивный байесовский классификатор (naive Bayes classijier), GIBBS [8, 11].

2. Применение машинного обучения в программной инженерии

В разработке ПО есть три категории объектов: *процессы* (наборы действий, связанных с ПО, такие как создание спецификации, детальное проектирование или тестирование), *продукты* (артефакты, результаты, документы, являющиеся результатом действия процесса, такие как документ спецификации, проектный документ или сегмент кода) и *ресурсы* (объекты, необходимые для деятельности процесса, такие как персонал, программные средства или оборудование) [2, 43]. Здесь есть внутренние и внешние атрибуты для сущностей вышеназванных категорий. Внутренние атрибуты описывают сам объект, тогда как внешние атрибуты характеризуют поведение объекта (как объект относится к его среде).

В работе [44] выделяются следующие области SE, к решению задач которых успешно применяются методы ML:

1) прогнозирование или оценка измерений для внутренних или внешних атрибутов процессов, продуктов или ресурсов;

2) обнаружение внутренних или внешних свойств процессов, продуктов или ресурсов;

3) преобразование продуктов для достижения некоторых желаемых или улучшенных внешних атрибутов;

4) синтезирование различных программных продуктов;

5) повторное использование продуктов или процессов;

6) улучшение процессов (например, восстановление спецификации из ПО);

7) управление специальными продуктами (такими как знания в области дизайна и разработки).

Поскольку на сегодня этот список стал длиннее, а размер статьи ограничен, то в этом разделе мы кратко рассмотрим результаты, попадающие только в первую из эти областей, где методы ML использу-

ются для прогнозирования или оценки измерений как внутренних, так и внешних атрибутов процессов, продуктов или ресурсов. К ним, в частности, относятся: качество ПО, размер ПО, стоимость разработки ПО, работа над проектом или ПО, работа по техническому обслуживанию, ресурс ПО, стоимость исправления, надежность ПО, дефекты ПО, возможность повторного использования, время выпуска ПО, производительность, время выполнения и тестируемость программных модулей и прочее.

Список приложений, включенных в раздел, хотя и далеко неполный, все же должен убедить читателя о необходимости и важности применения методов машинного обучения в программной инженерии.

Прогнозирование качества программного обеспечения

В работе [45] GP используется для генерации моделей качества ПО, принимающих в качестве входных данных метрики ПО, собранные ранее в процессе разработки, и прогнозируют для каждого модуля количество ошибок, которые будут обнаружены позднее в процессе разработки или во время эксплуатации. Эти прогнозы станут основой для ранжирования модулей, что позволит менеджеру выбрать столько модулей в верхней части списка, сколько ресурсов позволяют повысить надежность.

В работе [46] проведено эмпирическое сравнение нескольких методов моделирования (логистическую регрессию, многоуровневые нейронные сети, голографические сети и др.) для прогнозирования качества программных компонентов в начале жизненного цикла ПО. Другая работа по прогнозированию качества ПО на основе NN, как сообщается в [47], связана с конкретным языком, где сначала определяются метрики проектирования для языка SDL (язык спецификации и описания), а затем используются при построении моделей прогнозирования для идентификации подверженных сбоям компонентов. В работе [48] модели, основанные на NN, используются для прогнозирования ошибок и показателей качества ПО. В работе [49] использован алгоритм CART для оценки моделей качества ПО в течение нескольких выпусков. Авторы использовали очень большую унаследованную телекоммуникационную систему для разработки двух деревьев классификации качества ПО. Оба модуля дали лучшую точность, которая была бы полезной для разработчиков в различных методах разработки ПО.

В статье [50] представлено использование эволюционных DT в качестве подхода к прогнозированию ошибок. Было показано, что α-метрика как атрибут вместе с другими показателями сложности ПО может быть успешно использована для создания деревьев решений для прогнозирования опасных модулей (модулей, имеющих много необнаруженных ошибок). Модернизация таких модулей или выделение для них дополнительных усилий по тестированию или обслуживанию может в значительной степени повысить качество и надежность, сделав программное обеспечение гораздо более безопасным в использовании. В работах [51–55] для прогнозирования качества ПО используется метод обучения CBR. Основное внимание в [51] уделяется сравнению производительности различных классификаторов CBR. В работе [52] CBR применяется для моделирования качества ПО семейства полномасштабных промышленных программных систем, и точность прогнозируется лучше, чем соответствующая модель множественной линейной регрессии при прогнозировании числа ошибок проектирования. Два практических правила классификации (большинство голосов и кластеризация данных) предложены в [53] для оценки качества ПО высоконадежных систем. В работе [54] обсуждается процедура выбора атрибута, который может помочь идентифицировать соответствующие метрики качества ПО, которые будут использоваться в прогнозировании качества на основе CBR. В работе [55] подход CBR используется для калибровки моделей классификации качества ПО.

В работе [56] подход, основанный на DT, используется для генерации моделей компонентов высокого риска. Предложенный метод опирается на исторические

данные (показатели из предыдущих выпусков или проектов) для определения компонентов, склонных к сбоям. Другой подход, основанный на DT, используется при построении моделей для прогнозирования компонентов Ada высокого риска [57]. Деревья регрессии используются в [58] для классификации программных модулей, подверженных ошибкам. Подход позволяет иметь предпочтительный баланс между типами ошибочной классификации типов I и II. Алгоритм SPRINT DT используется в [59] для построения деревьев классификации как моделей оценки качества, предсказывающих класс программных модулей (подверженных ошибкам или не подверженных ошибкам). Набор методов вычислительного интеллекта, одним из которых является DT, предложен в [60] для анализа качества ПО. Гибридный подход, основанный на GP и DT, предложен в [61] для прогнозирования качества ПО и позволяет оптимизировать как показатели производительности, так и точности. Другой сравнительный результат исследования представлен в [62] об использовании методов ILP при прогнозировании сбоев ПО для программ на C ++. Как естественные, так и искусственные данные используются при оценке производительности двух методов ILP (FOIL и FLIPPER), для одного из них предлагаются некоторые расширения.

Оценка размера программного обеспечения

NN и GP используются в [63] для валидации основанного на компонентах метода для оценки размера ПО. В дополнение к полученным результатам, подтверждающие компонентный подход к определению размера ПО, в исследовании отмечается, что NN хорошо работает с данными, распознавая некоторые нелинейные отношения, которые не удается обнаружить методом множественной линейной регрессии. Уравнения, разработанные GP, дают аналогичные или лучшие значения, чем полученные уравнениями регрессии, и являются понятными, обеспечивая уверенность в результатах.

Прогноз стоимости программного обеспечения

Общий подход, называемый *оптимизированное сокращение набора* (optimized set reduction) и основанный на DT, описан в [64] для анализа данных разработки ПО и продемонстрирован как эффективный метод оценки стоимости ПО. Сравнительное исследование нескольких методов моделирования стоимости ПО, основаннное на реальных данных, проведено в [65] и включает метод CBR для прогнозирования стоимости ПО. Результат работы [66] указывает на то, что улучшенные прогностические характеристики моделей стоимости ПО могут быть получены с использованием байесовского анализа, предлагающего структуру, в которой могут быть использованы как предварительные экспертные знания, так и данные выборки для получения прогнозов. В работе [67] предложен подход на основе GP для поиска возможных функций стоимости ПО.

Прогнозирование усилий по разработке ПО (проекта)

Методы IBL используются в [68] для прогнозирования усилий проекта ПО для новых проектов. Полученные эмпирические результаты (из девяти различных промышленных наборов данных, насчитывающих в общей сложности 275 проектов) показывают, что CBR предлагает жизнеспособное дополнение к существующим методам прогнозирования и оценки. Другое применение CBR в оценке усилий ПО описано в [69]. Работа [70] фокусируется на эвристике поиска, чтобы помочь определить оптимальный набор функций в системе CBR для прогнозирования усилий по проекту ПО. Сравнение нескольких методов оценки CBR сделано в [71], и результаты показывают, что оценки, полученные с помощью аналогов, выбранных человеком, являются более точными, чем оценки, полученные с помощью аналогов, выбранных инструментами, и более точными, чем оценки с помощью простой регрессионной модели. В работе [72] DT и NN используются для прогнозирования усилий по раз-

работке ПО. Результаты были конкурентоспособны с традиционными методами, такими как COCOMO и функциональные точки. Основное преимущество систем оценки на основе DT и NN заключается в том, что они являются адаптируемыми и непараметрическими. В работах [73, 74] для прогнозирования усилий по разработке ПО также использовались NN и результаты обнадеживают с точки зрения точности. Дополнительные исследования по прогнозированию усилий на основе ML включают в себя: генетически обученный предиктор NN (GA + NN) [75] и подход, основанный на GP [76]. В работах [77–79] сообщалось о нескольких сравнительных исследованиях оценки программных усилий с использовнием NN&CBR в [78], CBR&NN&DT в [79] и NN&CBR&GP в [77].

Прогнозирование усилий по обслуживанию

В работе [80] модели генерируются в терминах методов NN и DT и методов регрессии для прогнозирования усилий по обслуживанию ПО. Исследование измеряет и сравнивает точность прогноза для каждой модели и делает вывод, что модели на основе DT и множественной регрессии имеют лучшие результаты точности. Рекомендуется использовать прогнозные модели в качестве инструментов для поддержки экспертных оценок и анализа влияния переменных технического обслуживания на процесс и продукт технического обслуживания.

Анализ программных ресурсов

В работе [81] DT используется при анализе данных о программных ресурсах для определения классов программных модулей, испытывающих большие усилия при разработке или отказы (понятие «высокий» определено в отношении самого верхнего квартиля относительно прошлых данных). Шестнадцать программных систем используются в исследовании. Деревья решений правильно идентифицируют 79,3 процента программных модулей, у которых были большие усилия по разработке или сбои.

Оценка стоимости исправления ошибок

Эмпирическое исследование сделано в [82], где DT и ILP используются для генерации моделей для оценки стоимости коррекции при обслуживании ПО. Сгенерированные модели оказываются полезными для оптимизации распределения ресурсов при корректирующих действиях по техническому обслуживанию и принятию решений относительно того, когда следует реструктурировать или реинжинирировать компонент, чтобы сделать его более обслуживаемым. Сравнение приводит к наблюдению, что результаты на основе ILP работают лучше, чем результаты на основе DT.

Прогноз повторного использования

Прогнозирующие модели построены с помощью DT в [83], чтобы проверить влияние некоторых внутренних свойств объектно-ориентированных приложений на возможность повторного использования. Усилия направлены на установление корреляции между возможностью повторного использования компонентов и тремя атрибутами ПО (наследование, связывание и сложность). Результаты эксперимента показывают, что некоторые программные метрики могут использоваться для прогнозирования с высокой степенью точности потенциальных классов многократного использования.

Время выпуска программного обеспечения

Как определить график выпуска ПО – это проблема, влияющая как на разработчика программного продукта, так и на пользователя и рынок. Метод, основанный на NN, предложен в [84] для оценки оптимального времени выпуска ПО. Метод принимает критерий минимизации затрат и переводит его в задачу прогнозирования временных рядов. Затем NN используется

для оценки времени обнаружения неисправности в будущем.

Прогнозирование тестируемости

В работе [85] описывается пример, в котором NN используется для прогнозирования тестируемости программных модулей на основе статических измерений исходного кода. Цель исследования – предсказать величину от нуля до единицы, распределение которой сильно смещено к нулю, что затруднительно для стандартных статистических методов. Результаты отражают характерную особенность прогнозирующих моделей на основе NN, обсуждаемые по настоящее время: их способность моделировать нелинейные отношения.

Производительность

Подход, основанный на BL, описан в [86] для оценки производительности программных проектов. Определена демонстрационная байесовская сеть доверия для захвата причинно-следственных связей между компонентами в модели COCOMO81 вместе с таблицами вероятностей для узлов.

Время выполнения

Временное поведение ПО в реальном времени имеет решающее значение для общей правильности системы. Таким образом, проверка того, нарушает ли система реального времени определенные ограничения по времени для определенных входных данных, становится критической проблемой. Подход, основанный на GA, описан в [87] для получения входных данных с самым длинным или самым коротким временем выполнения, которые можно использовать для проверки того, приведут ли они к временной ошибке или нарушению временных ограничений системы.

Прогнозирование надежности программного обеспечения

В работе [88] сообщается об использовании NN для прогнозирования роста надежности ПО. Эмпирическое сравнение проводится между NN-моделями и пятью известными моделями роста надежности ПО с использованием фактических наборов данных из ряда различных программных проектов. Результаты показывают, что основанные на NN модели хорошо адаптируются к различным наборам данных и имеют лучшую точность прогнозирования. Однако эффективность моделей прогнозирования на основе нейронной сети зависит от поведения набора данных, который в основном носит флуктуирующий характер. Поэтому часто NN страдает от переобучения результатов при работе с реальными неизвестными наборами данных. В работе [89] предложена модель прогнозирования надежности ПО на основе SVM и показано, что данные об ошибках, собранные на ранних этапах жизненного цикла разработки ПО, являются более подходящими для использования и влияют на точность прогнозирования. В работе [90] исследовали состояние методов раннего прогнозирования надежности ПО, используя SVM. Определено, что модель раннего прогнозирования, основанная на SVM, более точна в своем прогнозе с лучшей способностью обобщения. В работе [91] проводилось эмпирическое исследование нескольких методов ML, таких как ANN (BPN, RBFN и сеть Элмана), SVM, CCNN, DT и др., для прогнозирования надежности ПО. В частности, изучался вопрос: являются ли рабочие характеристики SVM и DT лучше, чем CCNN и NN. Отмечалось, что хотя DT действительно демонстрируют свои сильные стороны в различных реальных приложениях, однако на практике они неэффективны для прогнозирования надежности ПО.

Предсказание дефектов

Для прогнозирования дефектов ПО в работе [92] используется BL. Хотя система, о которой сообщается, является всего лишь прототипом, она показывает потенциал байесовских сетей доверия (BBN) в объединении нескольких точек зрения на прогнозирование дефектов в единую унифицированную модель. Переменные в прототипе системы BBN выбираются для представления процессов спецификации, проектирования и реализации, а также тестирования жизненного цикла (сложность

проблемы, усилия на проект, размер проекта, введенные дефекты, тестирующее усилие, обнаруженные дефекты, плотность дефектов при тестировании, остаточное количество дефектов и остаточная плотность дефектов). Надлежащие причинно-следственные связи между этими процессами жизненного цикла ПО затем фиксируются и отражаются в виде дуг, соединяющих переменные. Затем используется инструмент в отношении модели BBN следующим образом. Для заданных фактов об усилиях на проект и размере проекта в качестве входных данных инструмент будет использовать байесовский вывод для получения распределений вероятностей для введенных дефектов, обнаруженных дефектов и плотности дефектов.

3. Общие рекомендации по применению алгоритмов ML к задачам SE

Применяя машинное обучение к решению любой реальной проблемы, обычно нужно следовать определенному курсу действий. Проблемы программной инженерии не являются исключением. Предлагаемые рекомендации по применению ML к задачам SE имеют такие шаги.

Постановка проблемы. Первый шаг – сформулировать данную проблему так, чтобы она соответствовала структуре конкретного метода обучения, выбранного для этой задачи. Разные методы обучения имеют различную индуктивную предвзятость, используют разные стратегии поиска, основанные на различных руководящих факторах, предъявляют различные требования в отношении теории предметной области (наличие или отсутствие) и обучающих данных (оценка и свойства) и основаны на разных рассуждениях обоснования. Все эти вопросы должны быть приняты во внимание на этапе формулирования проблемы. Этот шаг имеет решающее значение для применимости метода обучения. Такие стратегии, как «разделяй и властвуй», могут потребоваться для разложения исходной проблемы на набор подзадач, более поддающихся выбранному методу обучения. Иногда лучшая формулировка проблемы не всегда может быть наиболее интуитивной для исследователя машинного обучения [94].

Проблемное представление. Следующим шагом является выбор подходящего представления как для данных обучения, так и для знаний, которые необходимо изучить. Разные методы обучения имеют разные формализмы представления. Таким образом, представление атрибутов и функций в учебной задаче часто зависит от конкретной проблемы и от формализма.

Сбор информации. Третий шаг – сбор данных, необходимых для учебного процесса. Качество и количество необходимых данных зависят от выбранного метода обучения. Данные, возможно, должны быть предварительно обработаны, прежде чем они могут быть использованы в процессе обучения.

Подготовка теории предметной области. Некоторые методы обучения (например, EBL) полагаются на доступность теории предметной области для данной проблемы. Приобретение и подготовка теории предметной области (или базовых знаний) и обеспечение качества этой теории (правильность, полноту) становится важной проблемой, которая повлияет на результат процесса обучения.

Выполнение учебного процесса. Как только данные и теория предметной области (при необходимости) готовы, процесс обучения может быть выполнен. Данные будут разделены на тренировочный набор и тестовый набор. Если используется какой-либо инструмент обучения или среда, данные обучения и тестовые данные, возможно, должны быть организованы в соответствии с требованиями инструмента. Знания, полученные из обучающего набора, проверяются на тестовом наборе. Из-за различий между обучающим набором и тестовым набором сам процесс обучения является итеративным.

Анализ и оценка полученных знаний является неотъемлемой частью учебного процесса. Интерес и эффективность полученных знаний будут тщательно изучены на этом этапе, часто с помощью экспертов-людей, что, как мы надеемся, приведет к улучшению знаний. Если полученные знания считаются несущественными,

неинтересными, неуместными или отклоняющимися, это может указывать на необходимость пересмотра на ранних стадиях, таких как постановка проблемы и представление. Во многих методах обучения существуют известные практические проблемы, такие как переоснащение, локальные минимумы или проклятие размерности, которые обусловлены либо неадекватностью данных, шумом или не относящимися к ним атрибутами в данных, характером стратегии поиска или неверной теорией предметной области.

Хранение базы знаний. Этот шаг влечет за собой использование полученных знаний [93]. Знания могут быть встроены в систему разработки программного обеспечения или в программный продукт или использованы без встраивания их в компьютерную систему. Как отмечалось в [93], сила методов машинного обучения заключается не в конкретном индукционном методе, а в правильной постановке задач и создании представления, позволяющего сделать процесс обучения более понятным.

Выводы

Область разработки ПО оказалась благодатной почвой, где многие задачи по разработке и сопровождению ПО могут быть сформулированы как проблемы обучения и подходят с точки зрения алгоритмов обучения. Последние два десятилетия стали свидетелями многих приложений ML в разработке и обслуживании ПО. Большинство публикаций посвящены вопросу о том, как создавать модели для прогнозирования или оценки определенных свойств процесса разработки ПО или артефактов. При этом формируется некоторое представление о том, какие типы техник ML чаще других используются при решении задач прогнозирования и оценки качества в программной инженерии. Тремя основными из них являются CBR, NN и DT.

Литература

1. Brooks F. No silver bullet: essence and accidents of software engineering. IEEE *Computer*. 1987. Vol. 20, N 4. P. 10–19.
2. Андон Ф.И. и др. Основы инженерии качества программных систем. 2-е изд., перераб. и доп. К.: Академпериодика. 2007. 672 с.
3. Lowry M. Software engineering in the twenty first century. AI *Magazine.* 1992. Vol. 14, N 3. P. 71–87.
4. Mostow J. Special issue on artificial intelligence and software engineering. IEEE *Trans.* SE. 1985. Vol. 11, N 11. P. 1253–1408.
5. Partridge D. Artificial Intelligence and Software Engineering. AMACOM. 1998. 277 p.
6. Rich C., Waters R. Readings in Artificial Intelligence and Software Engineering. Morgan Kaufmann. 1986. 589 p.
7. Tsai J.J.P., Weigert T., Knowledge-Based Software Development for Real-Time Distributed Systems. *World Scientific Inc*. 1993. 236 p.
8. Mitchell T. Machine Learning. *McGraw-Hill*. 1997. 414 p.
9. Selfridge O. The gardens of learning: a vision for AI. AI *Magazine*. 1993. Vol.14, N 2. P. 36–48.
10. Quinlan J.R. Some elements of machine learning. 9th *International Workshop on Inductive Logic Programming. Lecture Notes in Artificial Intelligence*. Springer-Verlag. 1999. Vol. 1634. P.15–18.
11. Dietterich T.G. Machine learning research: four current directions. AI *Magazine*. 1997. Vol. 18, N 4. P. 97–136.
12. Seeger M. Learning with labeled and unlabeled data. Technical Report. University of Edinburgh. 2001.
13. Zhu X., Ghahramani Z., Lafferty J. D. Semi-supervised learning using gaussian fields and harmonic functions. *International Conference on Machine Learning* (ICML). 2003. P. 912–919.
14. Quinlan J.R. Decision trees as probabilistic classifiers. *4th International Workshop on Machine Learning.* Irvine, CA. 1987. P. 31–37.
15. Gehrke J., Ramakrishnan R., Loh W.R. BOAT-optimistic decision tree construction. ACM SIGMOD *International Conference Management of Data*. 1999. P. 169–180.
16. Quinlan J.R. C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo. CA. 1993. 312 p.
17. Breiman L., Friedman J., Olshen R., Stone C. Classification and Regression Trees. Technical report, Wadsworth International, Monterey, CA. 1984. 358 p.

18. Kohavi R. The power of decision tables. *The eighth european conference on machine learning* (ECML), 1995. P. 174–189.
19. Han J., Kamber M. Data mining: concepts and techniques. Morgan Kaufmann. 2006. 740 p.
20. Lyu M.R. Handbook of Software Reliability Engineering. New York: McGraw-Hill. 1996. 873 p.
21. Park S., Hoyoung N., Sugumaran V. A Semi automated filtering technique for software process tailoring using neural networks. *Expert System and Applications*. 2006. Vol. 30. P. 179–189.
22. Perlovsky L.I. Neural Networks and Intellect: Using Model Based Concepts. New York: Oxford University Press. 2000. 496 p.
23. Rumelhart D.E, Hinto G.E., Williams R.J. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: The MIT Press. 1986. Vol. 1. P. 318–362.
24. Moody J., Darken C.J. Fast learning in networks of locally tuned processing units. *Neural Computing*. 1989. Vol. 1. P. 81–294.
25. Specht D.F. Probabilistic neural networks. *Journal of Neural Networks*. 1990. Vol. 3. P. 110–118.
26. Elman J.L. Finding Structure in Time. *Cognitive science*. 1990. Vol.14, N 2. P. 179–211.
27. Fahlman S.E., Lebiere C. The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*. San Mated, CA: Morgan Kaufmann.1990. P. 524–532.
28. Kohonen T. Self-Organizing Maps. Berlin: Springer-Verlag. 1997. 513 p.
29. Daelemans W., Van den Bosch A. Memory-Based Language Processing. Cambridge University Press. 2005. 198 p.
30. Russell S.P., Norvig P. Artificial Intelligence. A Modern Approach. New Jersey, USA: Prentice-Hall. 2003. 932 p.
31. Hammond K.J. Case-Based Planning. Academic Press: New York. 1989. 297 p.
32. Kolodner J.L. An introduction to Case Based Reasoning. *Artificial Intelligence Review*. 1992. Vol. 6, N 1. P. 3–34.
33. Muggleton S. Inductive Logic Programming. *New Generation Computing*. 1991. Vol. 8. P. 295–318.
34. Quinlan J.R. Learning logical definitions from relations. *Machine Learning.* 1990. Vol. 5. P. 239–266.
35. Muggleton S., Feng C. Efficient induction of logic programs. *The First Conference on Algorithmic Learning Theory*. Japanese Society for Artificial Intelligence, Tokyo. 1990. P. 368–381.
36. Muggleton S. Inverse Entailment and Progol. *New Generation Computing*. 1995. Vol. 13. P. 245–286.
37. Vapnik V. Statistical learning theory. *Adaptive and Learning Systems*. 1998. Vol. 736.
38. Hanley J, McNeil B. J. The meaning and use of the area under a receiver operating characteristic ROC curve. *Radiology*. 1982. Vol. 143. P. 29–36.
39. Yang B, Xiang L. A study on software reliability prediction based on support vector machines. *International conference on industrial engineering and engineering management* (IEEM). 2007. P. 1176–1180.
40. Phillip S. DTReg predictive modeling software available**.** 2003. 395 p. http://www.dtreg.com.
41. Goldberg G.E. Genetic Algorithmic Search, Optimization and Machine Learning. Reading, MA: Addition-Wisely. 1989. 624 p.
42. Koza J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press. 1992. 609 p.
43. Fenton N.E., Pfleeger S.L. Software Metrics. PWS Publishing Company, 2nd ed. 1997. 411 p.
44. Zhang D., Tsai J.J.P. Machine learning and software engineering. *Software Quality Journal*. 2003. Vol. 11, Issue 2. P. 87–119.
45. Evett M., Khoshgoftar T., Chien P. and Allen E. GP-based software quality prediction. *Third Annual Genetic Programming Conference.* 1998. P. 60–65.
46. Lanubile F. and Visaggio G. Evaluating predictive quality models derived from software measures: lessons learned. *Journal of Systems and Software.* 1997. Vol. 38. P. 225–234.
47. Hong E., Wu C. Criticality models using SDL metrics set. *4th Asia-Pacific Software Engineering and International Computer Science Conference.* 1997. P. 23–30.
48. Khoshgoftaar T., Pandya A., Lanning D. Application of neural networks for predicting faults. *Annals of Software Engineering.* 1995. Vol. 1. P. 141–154.
49. Khoshgoftaar T.M., Allen E.B., Jones W.D., Hudepohl J.P. Classification – tree models of software quality over multiple releases. IEEE *Transactions on Reliability.* 2000. Vol. 49, N 1. P. 4–11.
50. Kokol P., Podgorelec V., Pighim M. Using software metrics and evolutionary

decision trees for software quality control. 2001. http://www.escom.co.uk/conference2001/papers/kokol.pdf.
51. El Emam K., Benlarbi S., Goel N., Rai S. Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software.* 2001. Vol. 55, N 3. P. 301–320.
52. Ganesan K., Khoshgoftaar T., Allen E. Cased-based software quality prediction. *International Journal of Software Engineering and Knowledge Engineering.* 2000. Vol. 10, N 2. P. 139–152.
53. Khoshgoftaar T., Seliya N. Analogy-Based Practical Classification Rules for Software Quality Estimation. *Empirical Software Engineering.* 2003. Vol. 8, N 4. P. 325–350.
54. Khoshgoftaar T., Nguyen L., Gao K., Rajeevalochanam J. Application of an attribute selection method to CBR-based software quality classification. 15*th* IEEE *International Conference on Tools with* AI. 2003.
55. Khoshgoftaar T., Cukic B., Seliya N. Predicting fault-prone modules in embedded systems using analogy-based classification models**.** *International Journal of Software Engineering and Knowledge Engineering.* 2002**.** Vol. 12**,** N 2**.** P. 201–221.
56. Porter A., Selby R. Empirically-guided software development using metric-based classification trees. IEEE *Software.* 1990. Vol. 7. P. 46–54.
57. Briand L., Basili V., Hetmanski C. Developing interpretable models with optimized set reduction for identifying high-risk software components. IEEE *Trans.* SE. 1993. Vol. 19, N 11. P. 1028–1043.
58. Khoshgoftaar T., Allen E.B., Deng J. Using regression trees to classify fault-prone software modules. IEEE *Transactions on Reliability.* 2002. Vol. 51, N 4. P. 455–462.
59. Khoshgoftaar T., Seliya N. Software quality classification modeling using the SPRINT decision tree algorithm. 14*th* IEEE *International Conference on Tools with AI.* 2002. P. 365–374.
60. Reformat M., Pedrycz W., Pizzi N.J. Software quality analysis with the use of computational intelligence. *Information and Software Technology.* 2003. Vol. 45, N 7. P. 405–417.
61. Khoshgoftaar T., Liu Y., Seliya N. Genetic programming-based decision trees for software quality classification. 15*th* IEEE *International Conference on Tools with* AI. 2003.
62. Cohen W., Devanbu P. A comparative study of inductive logic programming for software fault prediction. *The fourteenth International Conference on Machine Learning.* 1997.
63. Dolado J. A validation of the component-based method for software size estimation. IEEE *Trans.* SE. 2000. Vol. 26, N 10. P. 1006–1021.
64. Briand L., Basili V., Thomas W. A pattern recognition approach for software engineering data analysis. IEEE *Trans.* SE. 1992. Vol. 18, No 11. P. 931–942.
65. Briand L. et al. An assessment and Comparison of common software cost estimation modeling techniques. *International Conference on Software Engineering.* 1999. P. 313–322.
66. Chulani S., Boehm B., Steece B. Bayesian analysis of empirical software engineering cost models. IEEE *Trans.* SE. 1999. Vol. 25, N 4. P. 573–583.
67. Dolado J.J. On the problem of the software cost function. *Information and Software Technology.* 2001. Vol. 43, N l. P. 61–72.
68. Shepperd M., Schofield C. Estimating software project effort using analogies. IEEE *Trans.* SE. 1997. Vol. 23, N 12. P. 736–743.
69. Vicinanza S., Prietulla M.J., Mukhopadhyay T. Case-based reasoning in software effort estimation. 11*th Intl. Conf. On Information Systems.* 1990. P. 149–158.
70. Kirsopp C., Shepperd M. J., Hart J. Search Heuristics, Case-based Reasoning And Software Project Effort Prediction. *Genetic and Evolutionary Computation Conference* (GECCO). 2002. P. 1367–1374.
71. Walkerden F., Jeffrey R. An empirical study of analogy-based software effort estimation. *Empirical Software Engineering.* 1999. Vol. 4. P. 135–158.
72. Srinivasan K., Fisher D. Machine learning approaches to estimating software development effort. IEEE *Trans.* SE. 1995. Vol. 21, N 2. P. 126–137.
73. Heiat A. Comparison of artificial neural network and regression models for estimating software development effort. *Information and Software Technology.* 2002. Vol. 44, N 15. P. 911–922.
74. Wittig G., Finnie G. Estimating software development effort with connectionist models. *Information and Software Technology.* 1997. Vol. 39. P. 469–476.
75. Shukla K. Neuro-genetic prediction of software development effort. *Information and Software Technology.* 2000. Vol. 42, N 10. P. 701–713.

76. Lefley M., Shepperd M. J. Using genetic programming to improve software effort estimation based on general data sets. *Genetic and Evolutionary Computation Conference* (GECCO). 2003. P. 2477–2487.
77. Burgess C.J., Lefley M. Can genetic programming improve software effort estimation a comparative evaluation. *Information and Software Technology*. 2001. Vol. 43, N 14. P. 863–873.
78. Finnie G., Wittig G., Desharnais J-M. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*. 1997. Vol. 39, N 3. P. 281–289.
79. Mair C., Kadoda G., Lefley M., Phalp K., Schofield C., Shepperd M., Webster S. An investigation of machine learning based prediction systems. *Journal of Systems and Software*. 2000. Vol. 53, N l. P. 23–29.
80. Jorgensen M. Experience with the accuracy of software maintenance task effort prediction models. IEEE *Trans. SE*. 1995. Vol. 21, N 8. P. 674–681.
81. Selby R., Porter A. Learning from examples: generation and evaluation of decision trees for software resource analysis. IEEE *Trans.* SE. 1988. Vol. 14. P. 1743–1757.
82. De Almeida M., Lounis H., Melo W. An investigation on the use of machine learned models for estimating correction costs. *International Conference on Software Engineering*. 1998. P. 473–476.
83. Mao Y., Sahraoui H., Lounis H. Reusability hypothesis verification using machine learning techniques: a case study. 13*th* IEEE *International Conference on Automated Software Engineering*. 1998. P. 84–93.
84. Dohi T., Nishio Y., Osaki S. Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering*. 1999. Vol. 8, N l. P. 167–185.
85. Khoshgoftaar T., Allen E., Xu Z. Predicting testability of program modules using a neural network. IEEE *Symposium on Application-Specific Systems and Software Engineering Technology*. 2000. P. 57–62.
86. Stamelos I., Angelis L., Dimou P., Sakellaris E. On the use of Bayesian belief networks for the prediction of software productivity. *Information and Software Technology*. 2003. Vol. 45, N l. P. 51–60.
87. Wegener J., Sthamer H., Jones B.F., Eyres D.E. Testing real-time systems using genetic algorithms. *Software Quality Journa*. 1997. Vol. 6. P. 127–135.
88. Karunanithi N., Whitely D., Malaiya Y. Prediction of software reliability using connectionist models. IEEE *Trans.* SE. 1992. Vol. 18, N 7. P. 563–574.
89. Yang B., Xiang L. A study on software reliability prediction based on support vector machines. *International conference on industrial engineering and engineering management* (IEEM). 2007. P. 1176–1180.
90. Xingguo L., Yanhua S. An early prediction method of software reliability based on support vector machine. *International conference on wireless communications, networking and mobile computing* (WiCom). 2007. P. 6075–6078.
91. Kumar P., Singh Y. An empirical study of software reliability prediction using machine learning techniques. *International Journal of System Assurance Engineering and Management* (Int J Syst Assur Eng Manag). 2012. Vol. 3, N 3. P. 194−208.
92. Fenton N., Neil M. A critique of software defect prediction models. IEEE *Trans.* SE. 1999. Vol. 25, N 5. P. 675−689.
93. Langley P., Simon H. Applications of machine learning and rule induction. *Communications of ACM*. 1995. Vol. 38, N 11. P. 55−64.

References

1. Brooks F. (1987), "No silver bullet: essence and accidents of software engineering", IEEE *Computer*, Vol. 20 No.4, pp.10-19.
2. Andon P.I., Koval G.I., Korotune T.M., Lavrischeva E.M. and Suslov V.Yu. (2007), The Fundamentals for Software Quality Engineering, 2-nd ed., K.: Akademperiodika, 672 p. (in Russian).
3. Lowry M. (1992), "Software engineering in the twenty first century", AI *Magazine*, Vol.14 No.3, pp.71–87.
4. Mostow J. (1985), "Special issue on artificial intelligence and software engineering", IEEE *Trans.* SE, Vol.11 No.ll, pp. 1253─1408.
5. Partridge D. (1998), Artificial Intelligence and Software Engineering, AMACOM, 277 p.
6. Rich C. and Waters R.(1986), "Readings in Artificial Intelligence and Software Engineering", Morgan Kaufmann, 589 p.
7. Tsai J.J.P**.** and Weigert T. (1993), Knowledge-Based Software Development for

Real-Time Distributed Systems, World Scientific Inc., Singapore, 236 p.
8. Mitchell T. (1997), Machine Learning, McGraw-Hill, 414p.
9. Selfridge O. (1993), "The gardens of learning: a vision for AI", AI *Magazine*, Vol.14, N 2. P.36–48.
10. Quinlan J.R. (1999), "Some elements of machine learning", Proceedings of the *9th International Workshop on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, Springer-Verlag, Vol. 1634. P. 15–18.
11. Dietterich T. G. (1997), "Machine learning research: four current directions", AI *Magazine*. Vol. 18, N 4. P. 97–136.
12. Seeger M. (2001). Learning with labeled and unlabeled data (Technical Report). University of Edinburgh.
13. Zhu X., Ghahramani Z., and Lafferty J.D. (2003), "Semi-supervised learning using gaussian fields and harmonic functions", *In International Conference on Machine Learning* (ICML). P. 912–919.
14. Quinlan J.R. (1987), "Decision trees as probabilistic classifiers", *Proceedings of 4th International Workshop on Machine Learning*, Irvine, CA. P. 31–37.
15. Gehrke J., Ramakrishnan R. and Loh W.R. (1999) "BOAT-optimistic decision tree construction", In *Proceedings ACM SIGMOD International Conference Management of Data*, Philadelphia, PA. P. 169–180.
16. Quinlan J.R. (1993), C4.5: Programs for machine learning**,** Morgan Kaufmann, San Mateo, CA, 312 p.
17. Breiman L., Friedman J., Olshen R. and Stone C. (1984), Classification and Regression Trees. Technical report, Wadsworth International, Monterey, CA, 358 p.
18. Kohavi R. (1995), "The power of decision tables", *In: The eighth european conference on machine learning* (ECML-95), Heraklion, Greece, P. 174–189.
19. Han J. and Kamber M. (2006), Data mining: concepts and techniques, Morgan Kaufmann, India.
20. Lyu M.R. (1996) Handbook of Software Reliability Engineering. New York: McGraw-Hill.
21. Park S., Hoyoung N. and Sugumaran V. (2006) "A Semi automated filtering technique for software process tailoring using neural networks", *Expert System and Applications*, Vol. 30. P. 179–189.
22. Perlovsky L.I. (2000), Neural Networks and Intellect: Using Model Based Concepts. New York: Oxford University Press.
23. Rumelhart D.E, Hinto G.E. and Williams R.J. (1986), "Learning internal representations by error propagation", In D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: The MIT Press, Vol. 1. P. 318–362.
24. Moody J. and Darken C.J. (1989), "Fast learning in networks of locally tuned processing units", *Neural Computing*, Vol. 1, P. 81–294.
25. Specht D.F. (1990), "Probabilistic neural networks", *Journal of Neural Networks*, Vol. 3. P. 110–118.
26. Elman J**.** L. (1990), "Finding Structure in Time", *Cognitive science*, Vol. 14, N 2. P. 179–211.
27. Fahlman S.E. and Lebiere C. (1990), "The cascade-correlation learning architecture", *In Advances in Neural Information Processing Systems*. San Mated, CA: Morgan Kaufmann. P. 524–532.
28. Kohonen T. (1997), *Self-Organizing Maps*. Berlin: Springer-Verlag, 513p.
29. Daelemans W. and Van den Bosch A. (2005), Memory-Based Language Processing. Cambridge University Press.
30. Russell S.P. and Norvig, P. (2003), Artificial Intelligence. A Modern Approach (2nd ed.). New Jersey, USA: Prentice-Hall, 932 p.
31. Hammond K. J. (1989), Case-Based Planning. Academic Press: New York, 297 p.
32. Kolodner J.L. (1992), "An introduction to Case Based Reasoning", *Artificial Intelligence Review*, Vol. 6, N 1, P. 3–34.
33. Muggleton S. (1991), "Inductive Logic Programming",*New Generation Computing*, Vol. 8. P. 295–318
34. Quinlan J.R. (1990), "Learning logical definitions from relations", *Machine Learning*, Vol. 5. P. 239–266.
35. Muggleton S. and Feng C. (1990), "Efficient induction of logic programs", *In Proceedings of the First Conference on Algorithmic Learning Theory*, Japanese Society for Artificial Intelligence, Tokyo, pp. 368–381.
36. Muggleton S. (1995), "Inverse Entailment and Progol", *New Generation Computing*, Vol. 13, pp. 245-286.
37. Vapnik V. (1998), "Statistical learning theory", *Adaptive and Learning Systems*,Vol. 736.
38. Hanley J., McNeil B.J. (1982), "The meaning and use of the area under a receiver operating

characteristic ROC curve"**,** *Radiology*, Vol. 143. P. 29–36.
39. Yang B. and Xiang L. (2007), "A study on software reliability prediction based on support vector machines", *In: Proceedings of international conference on industrial engineering and engineering management* (IEEM'07), pp. 1176–1180.
40. Phillip S. (2003), "DTReg predictive modeling software", 395p. http://www.dtreg.com.
41. Goldberg G.E. (1989), "Genetic Algorithmic Search, Optimization and Machine Learning", Reading, MA: Addition-Wisely, 412p.
42. Koza J.R. (1992), "Genetic Programming: On the Programming of Computers by Means of Natural Selection", MIT Press, 609 p**.**
43. Fenton N.E. and Pfleeger S.L. (1997), Software Metrics, PWS Publishing Company, 2nd ed.
44. Zhang D. and Tsai J.J.P. (2003), "Machine learning and software engineering", *Software Quality Journal,* Vol.11, Issue 2, pp.87–119.
45. Evett M., Khoshgoftar T., Chien P. and E. Allen, (1998) "GP-based software quality prediction", *Proc. Third Annual Genetic Programming Conference,* P. 60–65.
46. Lanubile F. and Visaggio G., (1997) "Evaluating predictive quality models derived from software measures: lessons learned", *Journal of Systems and Software,* Vol. 38, P. 225–234.
47. Hong E. and Wu C., (1997), "Criticality models using SDL metrics set", *Proc. the 4th Asia-Pacific Software Engineering and International Computer Science Conference*, P. 23–30.
48. Khoshgoftaar T., Pandya A. and Lanning D. (1995), "Application of neural networks for predicting faults", *Annals of Software Engineering,* Vol. 1. P. 141–154.
49. Khoshgoftaar T.M., Allen E.B., Jones W.D. and Hudepohl J.P. (2000), "Classification – tree models of software quality over multiple releases", IEEE *Transactions on Reliability*, Vol. 49. N 1. P. 4–11.
50. Kokol P., Podgorelec V. and Pighim M. (2001), "Using software metrics and evolutionary decision trees for software quality control", Available at: http://www.escom.co.uk/conference2001/papers/kokol.pdf.
51. El Emam K., Benlarbi S., Goel N. and Rai S. (2001), "Comparing case-based reasoning classifiers for predicting high risk software components", *Journal of Systems and Software,* Vol. 55, N 3. P. 301–320.
52. Ganesan K., Khoshgoftaar T. and Allen E. (2000), "Cased-based software quality prediction", *International Journal of Software Engineering and Knowledge Engineering*, Vol.10 No.2, pp. 139–152.
53. Khoshgoftaar T. and Seliya N. (2003), "Analogy-Based Practical Classification Rules for Software Quality Estimation", *Empirical Software Engineering.* Vol. 8. N 4. P. 325–350.
54. Khoshgoftaar T., Nguyen L., Gao K. and Rajeevalochanam J. (2003), "Application of an attribute selection method to CBR-based software quality classification", *Proceedings of* 15*th* IEEE *International Conference on Tools with* AI.
55. Khoshgoftaar T., Cukic B. and Seliya N. (2002), "Predicting fault-prone modules in embedded systems using analogy-based classification models", *International Journal of Software Engineering and Knowledge Engineering,* Vol.12 N 2. P. 201–221.
56. Porter A. and Selby R.(1990), "Empirically-guided software development using metric-based classification trees", *IEEE Software,* Vol. 7. P. 46–54.
57. Briand L., Basili V. and Hetmanski C. (1993), "Developing interpretable models with optimized set reduction for identifying high-risk software components", IEEE *Trans.* SE, Vol. 19. N 11. P. 1028–1043.
58. Khoshgoftaar T., Allen E.B. and Deng J. (2002), "Using regression trees to classify fault-prone software modules", *IEEE Transactions on Reliability*, Vol. 51. N 4. P. 455–462.
59. Khoshgoftaar T. and Seliya N. (2002), "Software quality classification modeling using the SPRINT decision tree algorithm", *Proceedings of 14th* IEEE *International Conference on Tools with* AI. P. 365–374.
60. Reformat M., Pedrycz and W. and Pizzi N.J. (2003), "Software quality analysis with the use of computational intelligence", *Information and Software Technology*, Vol.45 No.7, pp.405–417.
61. Khoshgoftaar T., Liu Y. and Seliya N. (2003), "Genetic programming-based decision trees for software quality classification", *Proceedings of* 15*th* IEEE *International Conference on Tools with* AI.
62. Cohen W. and Devanbu P. (1997), "A comparative study of inductive logic programming for software fault prediction", *Proc. the fourteenth International Conference on Machine Learning*.

63. Dolado J. (2000), "A validation of the component-based method for software size estimation", IEEE *Trans.* SE, Vol.26 No. 10, pp. 1006–1021.
64. Briand L., Basili V. and Thomas W. (1992), "A pattern recognition approach for software engineering data analysis", IEEE *Trans.* SE, Vol. 18 No. 11, pp. 931–942.
65. Briand L. et al. (1999), "An assessment and Comparison of common software cost estimation modeling techniques", *Proc. International Conference on Software Engineering*, pp.313–322.
66. Chulani S., Boehm B. and Steece B. (1999), "Bayesian analysis of empirical software engineering cost models", IEEE *Trans.* SE, Vol. 25 No. 4, pp. 573–583.
67. Dolado J.J. (2001), "On the problem of the software cost function", *Information and Software Technology*, Vol.43 No.l, pp.61–72.
68. Shepperd M.and Schofield C. (1997), "Estimating software project effort using analogies", IEEE *Trans.* SE, Vol. 23 No. 12, pp. 736–743.
69. Vicinanza S., Prietulla M.J. and Mukhopadhyay T. (1990), "Case-based reasoning in software effort estimation", *Proc.* 11*th Intl. Conf. On Information Systems*, pp.149–158.
70. Kirsopp C., Shepperd M. J. and Hart J. (2002), "Search Heuristics, Case-based Reasoning And Software Project Effort Prediction", *Proceedings of Genetic and Evolutionary Computation Conference* (GECCO), pp. 1367–1374.
71. Walkerden F. and Jeffrey R. (1999), "An empirical study of analogy-based software effort estimation", *Empirical Software Engineering*, Vol.4, pp.135–158.
72. Srinivasan K. and Fisher D. (1995), "Machine learning approaches to estimating software development effort", IEEE *Trans.* SE, Vol. 21 No. 2, pp. 126–137.
73. Heiat A., (2002), "Comparison of artificial neural network and regression models for estimating software development effort", *Information and Software Technology*, Vol.44 No. 15, pp.911–922.
74. Wittig G. and Finnie G. (1997), "Estimating software development effort with connectionist models", *Information and Software Technology*, Vol.39, pp.469–476.
75. Shukla K. (2000), "Neuro-genetic prediction of software development effort", *Information and Software Technology*, Vol.42 No.10, pp.701–713.
76. Lefley M. and Shepperd M. J. (2003), "Using genetic programming to improve software effort estimation based on general data sets", *Proceedings of Genetic and Evolutionary Computation Conference* (GECCO), pp.2477–2487.
77. Burgess C.J. and Lefley M. (2001), "Can genetic programming improve software effort estimation? a comparative evaluation", *Information and Software Technology*, Vol.43 No.14, pp.863–873.
78. Finnie G., Wittig G.and Desharnais J-M. (1997), "A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models", *Journal of Systems and Software*, Vol.39 No.3, pp.281–289.
79. Mair C., Kadoda G., Lefley M., Phalp K., Schofield C., Shepperd M. and Webster S. (2000), "An investigation of machine learning based prediction systems", *Journal of Systems and Software*, Vol.53 No.l, pp.23–29.
80. Jorgensen M. (1995), "Experience with the accuracy of software maintenance task effort prediction models", IEEE *Trans.* SE, Vol.21 No.8, pp.674–681.
81. Selby R. and Porter A. (1988), "Learning from examples: generation and evaluation of decision trees for software resource analysis," IEEE *Trans.* SE, Vol. 14, pp.1743–1757.
82. De Almeida M., Lounis H. and Melo W. (1998), "*Proc. International Conference on Software Engineering*", 1998, pp.473–476.
83. Mao Y., Sahraoui H. and Lounis H. (1998), "Reusability hypothesis verification using machine learning techniques: a case study", *Proc.* 13th IEEE *International Conference on Automated Software Engineering*, 1998, pp.84–93.
84. Dohi T., Nishio Y. and Osaki S. (1999), "Optimal software release scheduling based on artificial neural networks", *Annals of Software Engineering*, Vol.8 No.l, pp.167–185.
85. Khoshgoftaar T., Allen E. and Xu Z. (2000), "Predicting testability of program modules using a neural network", *Proc.* IEEE *Symposium on Application-Specific Systems and Software Engineering Technology*, pp.57–62.
86. Stamelos I., Angelis L., Dimou P., Sakellaris E. (2003), "On the use of Bayesian belief networks for the prediction of software productivity", *Information and Software Technology*, Vol.45 No.l, pp.51–60.
87. Wegener J., Sthamer H., Jones B.F. and Eyres D.E. (1997), "Testing real-time systems

using genetic algorithms", *Software Quality Journal*, Vol. 6. P.127–135.
88. Karunanithi N., Whitely D. and Malaiya Y. (1992), Prediction of software reliability using connectionist models. IEEE *Trans.* SE. Vol. 18, N 7. P. 563–574.
89. Yang B. and Xiang L. (2007) A study on software reliability prediction based on support vector machines. *International conference on industrial engineering and engineering management* (IEEM). P. 1176–1180.
90. Xingguo L. and Yanhua S. (2007), An early prediction method of software reliability based on support vector machine. *International conference on wireless communications, networking and mobile computing* (WiCom). P. 6075–6078.
91. Kumar P., Singh Y. (2012) An empirical study of software reliability prediction using machine learning techniques. *International Journal of System Assurance Engineering and Management* (Int J Syst Assur Eng Manag).Vol. 3, N 3. P. 194−208.
92. Fenton N., Neil M. (1999) A critique of software defect prediction models. IEEE *Trans.* SE. Vol. 25, N 5. P. 675−689.
93. Langley P., Simon H. (1995), "Applications of machine learning and rule induction", *Communications of ACM*, Vol. 38. N ll. P. 55–64.

Получено 14.10.2019

Об авторах:

Мороз Ольга Григорьевна,
кандидат технических наук,
научный сотрудник.
Количество научных публикаций в украинских изданиях – 27.
Количество научных публикаций в зарубежных изданиях – 2.
Индекс Хирша – 2.
https://orcid.org/0000-0002-0356-8780,

Мороз Григорий Борисович,
кандидат технических наук,
старший научный сотрудник.
Количество научных публикаций в украинских изданиях – 65.
Количество научных публикаций в зарубежных изданиях – 4.
Індекс Хирша – 3.
https://orcid.org/0000-0001-8666-9503.

Место работы авторов:

Международный научно-учебный центр информационных технологий и систем НАН Украины и МОН Украины,
03680, г. Киев,
проспект Академика Глушкова, 40.
Тел.: (044) 502 6355.

Институт программных систем НАН Украины,
03187, г. Киев,
проспект Академика Глушкова, 40, корпус 5.
Тел.: (044) 526 33 09.

E-mail: olhahryhmoroz@gmail.com,
mgb@isofts.kiev.ua