Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://nv.nltu.edu.ua
https://doi.org/10.36930/40310616
Article received 10.12.2021 p.
Article accepted 29.12.2021 p.
UDC 004.422

@ Correspondence author
I. I. Symets
ivan.i.symets@lpnu.ua

В. С. Яковина, І. І. Симець
Національний університет "Львівська політехніка", м. Львів, Україна

ПРОГНОЗУВАННЯ ДЕФЕКТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АНСАМБЛЕМ НЕЙРОННИХ МЕРЕЖ

Прогнозування дефектів програмного забезпечення, зокрема крос-проєктне, є актуальною і важливою науково-прикладною задачею, вирішення якої спрямоване на підвищення якості та надійності програмних продуктів та зменшення вартості їх розроблення та супроводу. Перспективним підходом до розв'язання такої задач може бути використання штучних нейронних мереж, зокрема глибинного навчання та їх ансамблів. Ансамблювання часто може покращити точність прогнозування моделей і розпаралелити результуючу модель, що підвищує швидкість обчислень. У цьому дослідженні побудовано архітектуру глибинних нейронних мереж, яка володіє вищими показниками точності прогнозування дефектів програмного забезпечення порівняно із традиційними моделями машинного навчання. У ролі якості наборів вхідних даних використовували комбінований набір, отриманий з репозиторію PROMISE Software Engineering, який містить дані про тестування програмних модулів п'яти програм (KC1, KC2, PC1, CM1, JM1) та двадцять одну метрику коду. Для реалізації нейронних мереж використано мову програмування Python і відкритої нейромережної бібліотеки Keras. Автоматизоване налаштування гіперпараметрів нейронних мереж реалізовано за допомогою функції GridSearchCV. Розроблено модель прогнозування надійності ПЗ на основі методів глибинного навчання і показано, що підвищення точності прогнозування дефектів ПЗ до 93,97 % можна досягнути у спосіб відповідного вибору множини ознак (метрик програмного коду) з наступним використанням стекового ансамблю нейронних мереж, до якого входять багатошаровий перцептрон (MLP), нейронна мережа на основі радіально-базисних функцій (RBFNN), рекурентна нейронна мережа (RNN) та довга короткотермінова пам'ять (LSTM), а як метамодель використовують логістичну регресію. Реалізація стекового ансамблю нейронних мереж дає змогу в подальшому створити програмний засіб, який зможе допомагати при ідентифікації програмних компонент із найбільшою ймовірністю появи дефектів.

Ключові слова: надійність; класифікація; стековий ансамбль; глибинне навчання; метрики коду.

Вступ / Introduction

Дефект програмного забезпечення – це недолік компонент або модулів системи, що негативно впливає на зовнішній вигляд, роботу, функціональні можливості або продуктивність і може призвести до відмови певної функціональності або неправильної роботи системи. Більшість програмних дефектів виникають на етапі розроблення програмного забезпечення у вихідному коді програми і є спричинені низкою факторів, які виникають на різних етапах життєвого циклу продукту, а саме: помилки у програмному коді, проблеми комунікації, неточності вимог до ПЗ, слабка документація і дизайн, складність системи, терміни виконання проєкту, людський фактор, недостатнє тестування і т.ін. [1]. Поширеними способами виявлення дефектів програмного забезпечення є ручне тестування та перегляд коду. Основним недоліком цих методів є те, що вони досить дорогі з точки зору часу і сил.

Автоматизовані підходи до передбачення програмних дефектів знизять витрати та покращать якість програмних проектів [9]. Для підвищення якості й надійності програмного забезпечення застосовуються методи прогнозування дефектів програмного забезпечення для виявлення потенційних помилок. Традиційні методи виявлення помилок базуються на аналізі метрик програмного продукту і використовуються для класифікації потенційно дефектних модулів або передбачення приблизної кількості помилок у певному модулі системи. Як наслідок – метод прогнозування дефектів програмного забезпечення може допомогти розробникам виявити дефекти на основі наявних програмних показників, використовуючи методи аналізу даних, і цим самим покращити якість програмного забезпечення, що в кінцевому підсумку призводить до зниження витрат на розроблення програмного забезпечення на етапі розроблення та обслуговування.

Останні досягнення в галузі штучних нейронних мереж і машинного навчання, а також зростання потужності сучасних комп'ютерів сприяли появі нових концепцій, таких як глибоке навчання. Основна ідея полягає в тому, що штучна нейронна мережа з декількома

Інформація про авторів:
Яковина Віталій Степанович, д-р техн. наук, професор, кафедра систем штучного інтелекту. **Email:** vitaliy.s.yakovyna@lpnu.ua; https://orcid.org/0000-0003-0133-8591
Симець Іван Ігорович, аспірант, кафедра програмного забезпечення. **Email:** ivan.i.symets@lpnu.ua; https://orcid.org/0000-0003-1873-3168

Цитування за ДСТУ: Яковина В. С., Симець І. І. Прогнозування дефектів програмного забезпечення ансамблем нейронних мереж. Науковий вісник НЛТУ України. 2021, т. 31, № 6. С. 104–111.
Citation APA: Yakovyna, V. S., & Symets, I. I. (2021). Software defect prediction using neural network ensemble. *Scientific Bulletin of UNFU, 31*(6), 104–111. https://doi.org/10.36930/40310616

шарами здатна поступово витягувати функції вищого рівня з вихідних даних для вирішення складних проблем. Моделі глибокого навчання покривають деякі недоліки алгоритмів машинного навчання через те, що дають змогу обчислювальним моделям кількох шарів представляти та вивчати функції та дані з великою кількістю шарів абстракції, дещо копіюючи операційну структуру мозку: як він сприймає та розуміє інформацію та як створює структуру [11]. Останнім часом вони стали популярнішими й активно використовуються у прогнозуванні дефектів програмного забезпечення.

Об'єкт дослідження – прогнозування дефектів програмного забезпечення.

Предмет дослідження – архітектура та параметри ансамблю штучних нейронних мереж для класифікації та прогнозування дефектів програмного забезпечення.

Мета роботи – підвищити точність прогнозування дефектів у програмних модулях і компонентах шляхом використання ансамблю штучних нейронних мереж.

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*: встановити архітектуру та параметри нейронних мереж; побудувати ансамбль мереж з вищими показниками точності прогнозування дефектів програмного забезпечення.

Наукова новизна отриманих результатів дослідження – побудовано модель дефектності програмного забезпечення на основі метрик коду шляхом визначення оптимальних параметрів нейронної мережі та архітектури ансамблю мереж.

Практична значущість результатів дослідження – створено програмний засіб, що містить навчену нейромережеву модель, придатну для прогнозування надійності програмного забезпечення та ідентифікації програмних модулів з найбільшою ймовірністю появи дефектів.

Аналіз останніх досліджень та публікацій. Прогнозування дефектів є популярним напрямом у дослідженнях і активно розвивається, оскільки дає змогу зменшити зусилля та ресурси, необхідні для тестування ПЗ. Прорив у технологіях машинного навчання, особливо розвиток методів глибокого навчання, призвів до вирішення багатьох проблем за допомогою цих методів. Існує велика різноманітність досліджень, які розробляють та застосовують моделі на основі машинного навчання для передбачення дефектів у програмних системах.

У роботі [11] для проблеми прогнозування дефектів програмного забезпечення дослідники запропонували алгоритми глибинного навчання, щоб автоматично вивчати семантичні представлення програм і використовувати це уявлення для ідентифікації схильного до дефектів коду. Використання цих неявних функцій показує кращі результати, ніж попередні підходи, засновані на явних функціях, таких як метрика коду.

У дослідженні [2] автори пропонують підхід для прогнозування дефектів програмного забезпечення на рівні змін із використанням DBN (Deep Belief Network). Мережа навчається на основі традиційними метриками коду і генерує нові виразні функції та використовує їх у класичних класифікаторах машинного навчання. Вони витягують зв'язки з традиційних метрик коду, таких як кількість змінених модулів, каталогів і файлів, додані та видалені рядки, а також деякі функції, пов'язані з досвідом розробника. Пізніше автори запропонували підхід "TLEL" [6] на основі дерева рішень та ансамблевого навчання для класифікації.

У роботі [16] також використовують DBN, але в інший спосіб. Для прогнозування дефектів на основі семантики коду автори розробили aDBN для автоматичного вивчення семантичних ознак з вихідного коду. AST (Abstract Syntax Tree) програм і зміни вихідного коду використовуються для випадків передбачення рівня змін на рівні файлу, як вхідні дані для мережі,. Потім автори використовують класичні класифікатори машинного навчання та вилучені функції, щоб класифікувати файли вихідного коду, чи є вони помилковими чи без помилок.

Модель на основі LSTM (long short-term memory) нейронної мережі використано в роботі [10] для вивчення як семантичних, так і синтаксичних особливостей коду. Запропонований підхід представляє код як послідовність маркерів коду, яка подається в систему LSTM для перетворення коду у вектор ознак і станів ознак, що представляє семантичну інформацію маркера. Пізніше модель Tree-LSTM автори розробили з використанням представлення AST (Abstract Syntax Tree) як вхідних даних [8].

Методику пошуку помилок на основі нейронної мережі запропоновано в роботі [3]. Автори тренують нейронну мережу на прикладах дефектного та правильного коду, а потім використовують отриманий двійковий класифікатор для виявлення помилок. Щоб підготувати позначений набір даних, автори використовують наявне програмне забезпечення для виявлення статичних помилок, щоб ідентифікувати певний тип помилок. Код представлений у вигляді послідовності токенів і перетворюється у вектор реального значення за допомогою одноразового кодування для кожного маркера. Тоді як модель використовується двонаправлена нейронна мережа з LSTM.

Іншу модель, засновану на глибокому навчанні, для передбачення дефектів запропоновано в роботі [15]. Навчання нейронної мережі використовує техніку втрат триплетів і техніку зваженої перехресної ентропії.

У роботі [4] наведено методику прогнозування дефектів програмного забезпечення, засновану на моделі автокодерів із зниженими шумами. Автокодер із накопиченням шумів використовується для вилучення особливостей вищого рівня із традиційних показників. Для класифікації використовується двоетапне ансамблеве навчання на основі нейронних мереж у ролі класифікаторів.

Модель прогнозування дефектів програмного забезпечення побудовано в роботі [17] на основі сіамських паралельних повнозв'язаних нейронних мереж. Ця модель використовує архітектуру парних паралельних сіамських мереж і підхід глибокого навчання. Мережа створює функції високого рівня, які використовуються для класифікації.

Отже, аналіз останніх публікацій показує підвищений інтерес до прогнозування дефектів програмного забезпечення та до побудови таких моделей і методів, які дадуть змогу здійснювати крос-проєктне прогнозування дефектів. Перспективним підходом до вирішення такої задачі може бути використання штучних нейронних мереж, зокрема глибинного навчання та їх ансамблів.

Матеріали та методи дослідження. У досліджені Towards a Software Defect Proneness Model: Feature Selection автори цієї статті виконали дослідження із використанням злитого набору даних з репозиторію PROMISE Software Engineering, який містить дані про тесту-

вання програмних модулів п'яти програм (KC1, KC2, PC1, CM1, JM1) та 21 метрику коду. Здійснено порівняння ефективності роботи різних методів вибору ознак, зокрема досліджено вплив методу вибору ознак на точність класифікації із використанням таких класифікаторів: Random forest, Support vector machine, K-nearest neighbor, Decision tree classifier, AdaBoost classifier, Gradient Boosting. На основі голосування за результатами роботи методів вибору ознак побудовано статичну (детерміністичну) модель надійності ПЗ, яка встановлює взаємозв'язок між ймовірністю появи дефекту в програмному модулі та метриками його коду. Показано, що в цю модель входять такі метрики коду, як: loc, N, I, V, E, v(g), branchCount.

Як зазначено в огляді літературних джерел, традиційні методи машинного навчання на цей час не дають достатньо високої точності прогнозування дефектів програмного забезпечення внаслідок досить великої множини ознак, багато з яких мають сильну взаємну кореляцію, а також внаслідок недослідженості питання переносимості виявлених ознак між різними проєктами зі створення програмного забезпечення, які мають різні характеристики розробки. У такому разі перспективним є використання методів глибинного навчання та ансамблів нейронних мереж для виявлення ознак, підвищення точності прогнозування. Це дослідження спрямоване на застосування методів глибинного навчання для бінарної класифікації програмних модулів на дефектні та бездефектні на основі множини метрик програмного коду, використаних у ролі ознак.

Набір даних для дослідження складається із 15 123 записів, які складаються із 21 метрики коду (табл. 1), цільовою метрикою в цьому дослідженні буде виступати метрика defects, яка може містити значення true (модуль із зазначеними метриками з дефектом) або false (модуль із зазначеними метриками без дефекту).

Табл. 1. Метрики із вибраного набору даних / Description of metrics from the selected dataset

№	Позначення метрики	Визначення метрики
1	loc	LOC (McCabe's)
2	v(g)	Cyclomatic Complexity (McCabe's)
3	ev(g)	Essential Complexity (McCabe's)
4	iv(g)	Design Complexity (McCabe's)
5	n	Operator and Operand total (Halstead)
6	v	Volume (Halstead)
7	l	Program Length (Halstead)
8	d	Difficulty (Halstead)
9	i	Intelligence (Halstead)
10	e	Effort (Halstead)
11	b	(Halstead)
12	t	Time (Halstead)
13	lOCode	LOC (Halstead)
14	lOComment	Line of Comments (Halstead)
15	lOBlank	Line of Blanks (Halstead)
16	lOCodeAndComment	Line of Code and Comment (Halstead)
17	uniq_Op	No. of unique operator
18	uniq_Opnd	No. of unique operand
19	total_Op	Total operator
20	total_Opnd	Total operands
21	branchCount	Count of Branch

Набір даних є незбалансованим і містить 2 665 записів про дефектні модулі і 12 458 записів про модулі без дефектів. Вплив незбалансованих даних при машинному навчанні є неявним, тобто це не спричиняє миттєвої помилки під час створення та запуску моделі, але результати можуть бути оманливими. Якщо ступінь дисбалансу класів для класу більшості є надзвичайним, то можна отримати навчений класифікатор із високою/низькою точністю прогнозування, оскільки модель, швидше за все, передбачить більшість вибірок, що належать до класу більшості.

Тому для подальшого дослідження дані були збалансовані так – було вибрано підмножину із 5 330 записів, в яких значення метрики defects розподілені 50 на 50 % і подальші дослідження було виконано на цій множині даних. 80 % даних із 5 330 було використано як збалансовану навчальну вибірку, а 20 % незбалансованих даних – для тестування отриманих моделей.

Усі дослідження у цій роботі виконуються за допомогою мови програмування Python і головно бібліотеки Keras – відкрита нейромережна бібліотека, написана мовою Python. Вона здатна працювати поверх TensorFlow, Microsoft Cognitive Toolkit, R. Спроектовану для уможливлення швидких експериментів з мережами глибинного навчання, її зосереджено на тому, щоби вона була зручною в користуванні, модульною та розширюваною.

Оцінка класифікації за допомогою вибраних нейронних мереж здійснюється на вибірці даних із 21 метрики і відповідно 7 метриках, які було визначено як важливі для прогнозування дефектів програмного забезпечення для порівняння результатів.

Оптимізація гіперпараметрів є важливою частиною глибокого навчання. Причина в тому, що нейронні мережі, як відомо, важко налаштувати, й існує багато параметрів, які потрібно встановити. Окрім цього, окремі моделі можуть дуже повільно тренуватися.

Продуктивність моделей істотно залежить від значення гіперпараметрів. Зауважте, що неможливо заздалегідь дізнатися найкращі значення для гіперпараметрів, тому в ідеалі нам потрібно спробувати всі можливі значення, щоб дізнатися оптимальні значення. Виконання цього вручну може зайняти багато часу та ресурсів, тому ми використовуємо GridSearchCV для автоматизації налаштування гіперпараметрів.

GridSearchCV – це функція, яка постачається в пакеті model_selection Scikit-learn. Ця функція допомагає перебирати попередньо визначені гіперпараметри та вставляти ваш оцінювач (модель) у ваш навчальний набір. Отже, зрештою, ми можемо вибрати найкращі параметри із перерахованих гіперпараметрів. Наступний фрагмент коду демонструє параметри, які ми будемо передавати у функцію GridSearchCV:

```
   Parameter_Trials={'batch_size':[1, 64,
128, 256], 'epochs':[1, 5, 10, 50, 100],
   'Optimizer_Trial':['SGD','RMSprop',
'Adagrad', 'Adadelta', 'Adam','Adamax'],
'Neurons_Trial': [5, 10, 20, 50],
   'Activation':['softmax', 'softplus',
'softsign', 'relu', 'tanh', 'sigmoid',
'hard_sigmoid', 'linear']}
```

Параметри, які ми передаємо для навчання, означають таке:

- **batch_size** – вказує, скільки рядків буде передано в мережу за один прийом, після чого розпочнеться обчислення SSE, а нейронна мережа почне коригувати свої ваги на основі помилок.
- **Epochs** – кількість епох навчання.

- **Optimizer_Trial** – параметр допомагає знайти оптимальні значення кожної ваги в нейронній мережі.
- **Neurons_Trial** – кількість нейронів у шарі мережі.
- **Activation** – визначає функцію активації для обчислень всередині кожного нейрона.

Нижче наведено фрагмент коду із використанням функції GridSearchCV і на рис. 1 – результати (фрагмент розрахунку) роботи функції GridSearchCV.

```
# Creating the classifier
classifierModel=KerasClassifier(make_classification_rnn_lstm, verbose=0)
# Creating the Grid search space
grid_search=GridSearchCV(estimator=classifierModel, param_grid=Parameter_Trials, scoring='accuracy', cv=5)
# Running Grid Search for different paramenters
grid_result=grid_search.fit(X_train, y_train, verbose=1)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
print("%f (%f) with: %r" % (mean, stdev, param))
```

```
0.514286 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 15, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 15, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 20, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'relu', 'Neurons_Trial': 20, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.285714 (0.393830) with: {'Activation': 'tanh', 'Neurons_Trial': 15, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 15, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 20, 'Optimizer_Trial': 'SGD', 'batch_size': 128, 'epochs': 10}
0.485714 (0.448125) with: {'Activation': 'tanh', 'Neurons_Trial': 20, 'Optimizer_Trial': 'Adam', 'batch_size': 128, 'epochs': 10}
```

Рис. 1. Фрагмент розрахунку параметрів за допомогою функції GridSearchCV / Fragment of parameter calculation using the GridSearchCV function

Для класифікації вибрано і використано такі популярні нейронні мережі у сфері прогнозування дефектів програмного забезпечення:

1. Багатошаровий перцептрон (MLP) є доповненням до нейронної мережі прямого зв'язку. Він складається із трьох типів шарів – вхідного, вихідного та прихованого. Вхідний шар отримує вхідний сигнал для обробляння. Потрібне завдання, таке як передбачення та класифікація, виконується вихідним шаром. Довільна кількість прихованих шарів, які розміщуються між вхідним і вихідним шарами, є справжньою обчислювальною машиною MLP. Подібно до мережі прямого зв'язку в MLP, дані переходять у прямому напрямку від рівня входу до вихідного. Нейрони в MLP тренуються за допомогою алгоритму навчання зворотного поширення. MLP призначені для апроксимації будь-якої безперервної функції і можуть вирішувати задачі, які не є лінійно роздільними. Основними випадками використання MLP є класифікація шаблонів, розпізнавання, передбачення та апроксимація [12].

Використовуючи мову програмування Python і відкриту нейромережну бібліотеку Keras було реалізовано MLP мережу, яка складається із одного вхідного шару, двох прихованих і одного вихідного шарів. За допомогою функції GridSearchCV визначено гіперпараметри для моделі: batch = 64; epochs = 5; optimizer = RMSprop; neurons = 50; activation = relu. Реалізацію MLP нейронної мережі за допомогою мови програмування Python і бібліотеки Keras наведено у такому фрагменті коду:

```
mlnn_model = Sequential()
mlnn_model.add(Dense(units=50, input_dim= 7, kernel_initializer='normal', activation='relu'))
mlnn_model.add(Dense(units=50, kernel_initializer='normal', activation='relu'))
mlnn_model.add(Dense(units=1, kernel_initializer='normal', activation='relu'))
mlnn_model.compile(optimizer='RMSprop', loss='binary_crossentropy', metrics=['accuracy'])
print(mlnn_model.summary())
mlnn_model.fit(X_train, y_train, batch_size=64, epochs = 5)
scores = mlnn_model.evaluate(X_test, y_test)
print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2 f%%" % (scores[1]*100))
```

Результати оцінки точності цієї мережі наведено на рис. 2.

```
Multi-layer Perceptron
All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'
Accuracy: 82.64%
All features:
Accuracy: 69.56%
```

Рис. 2. Точність класифікації MLP нейронної мережі / Accuracy of MLP neural network classification

2. Radial Basis Function (RBF) Networks (Мережі радіальної базисної функції). RBF подібні до MLP із трьома шарами (вхідний, середній або "прихований" і вихід). Також, як і MLP, RBF можуть легко моделювати будь-яку нелінійну функцію. Основна відмінність між двома мережами полягає в тому, що RBF не вводить необроблені вхідні дані, а передає міру відстані від вхідних даних до прихованого шару. Ця відстань вимірюється від деякого центрального значення в діапазоні змінної (іноді середнього) до заданого вхідного значення [5]. За допомогою GridSearchCV визначено гіперпараметри для моделі: batch = 128; epochs = 5; optimizer = SGD; neurons = 50; activation = sigmoid. Реалізацію RBF нейронної мережі за допомогою мови програмування Python і бібліотеки Keras наведено у такому фрагменті коду:

```
rbfnn_model = Sequential()
rbfnn_model.add(Flatten(input_shape=(7, 1)))
rbfnn_model.add(RBFLayer(50, 0.5))
rbfnn_model.add(Dense(1, activation='sigmoid'))
rbfnn_model.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])
rbfnn_model.fit(X_test, y_test, batch_size=128, epochs=10)
rbfnn_model.fit(X_train, y_train, batch_size=64, epochs = 5)
scores = rbfnn_model.evaluate(X_test, y_test)
print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2 f%%" % (scores[1]*100))
```

Результати оцінки точності даної мережі наведено на рис. 3.

```
Radial Basis Function (RBF) Networks
All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'
Accuracy: 87.44%
All features:
Accuracy: 75.01%
```

Рис. 3. Точність класифікації RBF нейронної мережі / Accuracy of RBF neural network classification

3. Simple Recurrent Neural Networks (Рекурентна нейронна мережа). Рекурентна нейронна мережа (RNN) – це особливий тип штучної нейронної мережі, пристосованої для роботи з даними часових рядів або даними, які містять послідовності. Звичайні нейронні мережі прямого зв'язку призначені тільки для точок даних, які не залежать одна від одної. RNN називають рекурентними, оскільки виконує одну і ту саму функцію для кожного введення даних, тоді як вихід поточного входу залежить від одного минулого обчислення. RNN мають концепцію "пам'яті", яка допомагає їм зберігати стани або інформацію попередніх входів для створення наступного виходу послідовності. В інших нейронних мережах всі входи незалежні один від одного. Але в RNN всі вхідні дані пов'язані один з одним [14]. RNN мають низку переваг, таких як:

- здатність обробляти послідовності даних;
- здатність обробляти вхідні дані різної довжини;
- здатність зберігати або "запам'ятовувати" історичну інформацію.

Недоліками є:

- обчислення може бути дуже повільним;
- мережа не враховує майбутні вхідні дані для прийняття рішень;
- проблема зникаючого градієнта, коли градієнти, які використовуються для обчислення оновлення ваги, можуть наблизитися до нуля, заважаючи мережі вивчати нові ваги. Чим глибша мережа, тим більш виражена ця проблема.

Використовуючи мову програмування Python і бібліотеку Keras реалізовано Simple RNN мережу, яка складається із одного вхідного, одного прихованого рекурентного і одного вихідного шарів. За допомогою функції GridSearchCV визначено гіперпараметри для моделі: batch = 256; epochs = 50; optimizer = Adam; neurons = 20; activation = tanh. Фрагмент коду із реалізацією RNN нейронної мережі:

```
rnn_model = Sequential()
rnn_model.add(SimpleRNN(20, input_shape=(7,1), activation = 'tanh' , use_bias=True, kernel_initializer='glorot_uniform'))
rnn_model.add(Dense(units = 1, activation = 'tanh'))
rnn_model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])
rnn_model.fit(X_train, y_train, epochs=50, batch_size=256, verbose=2)
rnn_model.fit(X_train, y_train, batch_size=256, epochs = 50)
scores = rnn_model.evaluate(X_test, y_test)
print("All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2 f%%" % (scores[1]*100))
```

Результати оцінки точності цієї мережі наведено на рис. 4.

```
Simple Recurrent Neural Networks
All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'
Accuracy: 89.97%
All features:
Accuracy: 78.87%
```

Рис. 4. Точність класифікації RNN нейронної мережі / Accuracy of RNN neural network classification

4. LSTM Neural Networks. Довга короткотермінова пам'ять (Long short-term memory; LSTM) – особливий різновид архітектури рекурентних нейронних мереж, здатна до навчання довготермінових залежностей. Вони чудово вирішують цілу низку різноманітних завдань і нині широко використовуються. LSTM розроблено спеціально, щоб уникнути проблеми довготривалої залежності. Запам'ятовування інформації на довгі періоди часу – це їхня звичайна поведінка, а не щось, чому вони намагаються навчитися. LSTM також нагадує ланцюжок, але модулі виглядають інакше. Замість одного шару нейронної мережі вони містять чотири, і ці шари взаємодіють особливо. Тут вирішується проблема зникаючого градієнта RNN. LSTM добре підходить для класифікації, оброблення та прогнозування часових рядів [14].

Використовуючи бібліотеку Keras реалізовано LSTM мережу, яка складається із одного вхідного, трьох прихованих LSTM і одного вихідного шарів. За допомогою функції GridSearchCV визначено гіперпараметри для моделі: batch = 128; epochs = 20; optimizer = SGD; neurons = 20; activation = tanh. Фрагмент коду реалізації нейронної мережі:

```
lstm_model = Sequential()
```

```
lstm_model.add(LSTM(30, input_shape=(7, 1), return_sequences=True))
lstm_model.add(LeakyReLU( alpha=0.3))
lstm_model.add(LSTM(15, return_sequences=True))
lstm_model.add(LeakyReLU( alpha=0.3))
lstm_model.add(LSTM(30, return_sequences=False))
lstm_model.add(Dense(1, activation='tanh'))
lstm_model.compile(optimizer='SGD', loss='binary_crossentropy', metrics=['accuracy'])
lstm_model.fit(X_train, y_train, batch_size=128, epochs = 20)
scores = lstm_model.evaluate(X_test, y_test)
print("Important features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'")
print("Accuracy: %.2 f%%" % (scores[1]*100))
```

Результати оцінки точності цієї мережі наведено на рис. 5.

```
LSTM Neural Networks
All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'
Accuracy: 91.16%
All features:
Accuracy: 80.79%
```

Рис. 5. Точність класифікації LSTM нейронної мережі / Accuracy of LSTM neural network classification

Результати дослідження та їх обговорення / Research results and their discussion

У цьому дослідженні використано і реалізовано чотири нейронні мережі, які добре себе проявили в точності класифікації модулів на дефектні і такі, що без дефектів (табл. 2).

Табл. 2. Точність класифікації з використанням нейронних мереж / Accuracy of classification using neural networks

№ п/п	Нейронна мережа	Точність класифікації (Accuracy) %	
		Усі метрики (21 метрика)	Важливі метрики (7 метрик)
1	MLP	69,56	82,64
2	RBF	75,01	87,44
3	RNN	78,87	89,97
4	LSTM	80,79	91,16

Використання ансамблювання часто дає змогу підвищити точність прогнозу моделей машинного навчання, особливо у разі різнорідних даних, а також розпаралелювати отриману результуючу модель, що підвищує швидкість обчислень.

Ансамблеве навчання – це парадигма машинного навчання, де кілька моделей (часто називаються "слабкими учнями") навчаються вирішувати одну й ту саму проблему та об'єднуються для отримання кращих результатів. Основна гіпотеза полягає в тому, що якщо слабкі моделі правильно комбінувати, ми можемо отримати більш точні та/або надійні моделі.

На основі розглянутих нейронних мереж вирішено створити стековий ансамбель (Stacking Ensemble), який у ролі метамоделі (супервайзера) використовує логістичну регресію LogisticRegression (рис. 6).

Ідея стекування полягає в тому, щоб вивчити кілька різних слабких учнів і об'єднати їх шляхом навчання метамоделі для виведення прогнозів на основі множинних передбачень, які повертаються цими слабкими моделями.

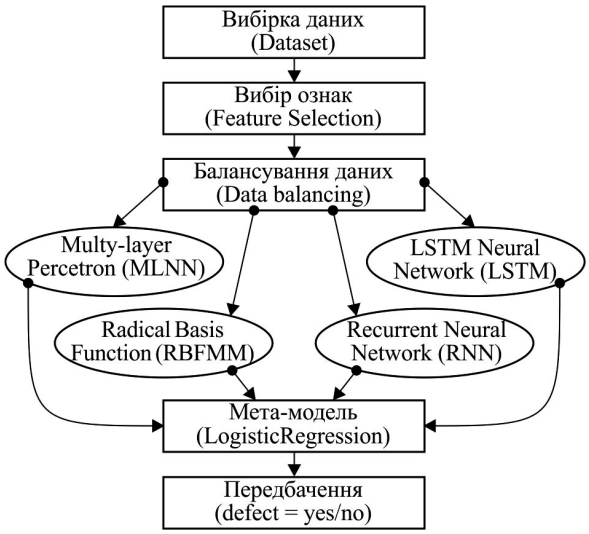

Рис. 6. Схема роботи стекового ансамблю із чотирьох нейронних мереж / Workflow of stack ensemble of four neural networks

За допомогою компонента StackingClassifier бібліотеки Sklearn і мови програмування Python реалізовано стековий ансамбль. Цей ансамбль показав хороші результати прогнозування дефектності модулів системи, результати наведено на рис. 7, і видно що при класифікації з важливими ознаками точність прогнозу становить 93,97 %, ця точність є вищою, ніж точності, які були при класифікації із використанням окремих нейронних мереж.

```
mlnn = KerasClassifier(build_fn=get_mlnn_model, batch_size=64, epochs = 5)
mlnn._estimator_type = "classifier"
mlnn_pipe = Pipeline([('scaler', scaler), ('mlnn', mlnn)])
rbfnn = KerasClassifier(build_fn=get_rbfnn_model, batch_size=256, epochs=10)
rbfnn._estimator_type = "classifier"
rbfnn_pipe = Pipeline([('scaler', scaler), ('rbfnn', rbfnn)])
rnn = KerasClassifier(build_fn=get_rnn_model, epochs=50, batch_size=256)
rnn._estimator_type = "classifier"
rnn_pipe = Pipeline([('scaler', scaler), ('rnn', rnn)])
lstm = KerasClassifier(build_fn=get_lstm_model, epochs=50, batch_size=256)
lstm._estimator_type = "classifier"
lstm_pipe = Pipeline([('scaler', scaler), ('lstm', lstm)])
# Make an ensemble
ensemble = StackingClassifier(estimators=[
('mlnn', mlnn_pipe), ('rnn', rnn_pipe),('rbfnn', rbfnn_pipe),('lstm', lstm_pipe)],
final_estimator=LogisticRegression())
```

```
Neural Networks Ensemble
All features: 'loc', 'v(g)', 'N', 'I', 'branchCount', 'V', 'E'
Accuracy: 93.97%
All features:
Accuracy: 81.84%
```

Рис. 7. Точність класифікації стекового ансамблю із чотирьох нейронних мереж / Accuracy of stack ensemble classification of four neural networks

Обговорення результатів дослідження. У статті [18] розглянуто дві моделі глибокого навчання, Stack Sparse Auto Encoder (SSAE) і Deep Belief Network (DBN) використовуються для класифікації наборів даних NASA, які є незбалансованими та мають недостатню кількість вибірок. Згідно з проведеним експериментом, модель SSAE отримує кращі результати порівняно з моделлю DBN, і вона становить 90 %.

У дослідженні [13] використовують вибірку даних з проєкту JM1 з репозиторію, який ми використовуємо. Для підвищення точності класифікації автори пропонують використання самоорганізуючих карт з ієрархічною кластеризацією та попереднім обробленням даних, при якому можна отримати точність прогнозування 86 %.

У дослідженні Towards a Software Defect Proneness Model: Feature Selection автори цієї статті виконали дослідження із використанням злитого набору даних з репозиторію PROMISE Software Engineering із використанням наступних класифікаторів: Random forest, Support vector machine, K-nearest neighbor, Decision tree classifier, AdaBoost classifier, Gradient Boosting. За допомогою цих класифікаторів отримано точність прогнозування в межах 79,8–86,0 %.

Тому це дослідження підтверджує, що використання методів глибинного навчання покращує точність прогнозування дефектів програмного забезпечення. А використання ансамблювання часто дає змогу підвищити точність прогнозу моделей машинного навчання. Використання розробленого ансамблю дає змогу прогнозувати дефектність модулів у програмному забезпеченні із точністю до 93,97 %.

Висновки / Conclusions

Ця робота присвячена вирішенню важливого науково-прикладного завдання підвищення точності прогнозування дефектів у програмних модулях та компонентах шляхом використання методів глибинного машинного навчання та ансамблю штучних нейронних мереж.

У ролі наборів вхідних даних використовували комбінований набір, отриманий з відкритого репозиторію PROMISE Software Engineering. Набір даних містив 15123 записи, кожен з яких складався із 21 метрики коду та цільової бінарної змінної, яка вказувала на наявність дефектів у цьому програмному модулі чи проєкті.

У цьому дослідженні використано і реалізовано чотири нейронні мережі, а саме багатошаровий перцептрон, нейронна мережа на основі радіально-базисних функцій, рекурентна нейронна мережа та довга короткотермінова пам'ять. Оптимальні параметри архітектури кожної нейронної мережі визначали методом GridSearch за допомогою програмної реалізації мовою Python. Найкращої точності прогнозування досягнуто у разі використання моделей на основі двох останніх мереж: рекурентної нейронної мережі та довгої короткотермінової пам'яті. У цьому разі точність прогнозування дефектів ПЗ з використанням усього набору ознак становила 78,87 та 80,79 % відповідно. Використання розробленої авторами раніше статичної моделі надійності ПЗ, яка передбачає зменшення множини ознак до семи найважливіших, дає змогу збільшити точність прогнозування зазначеними методами до 89,97 (для RNN) та 91,16 % (у разі LSTM).

На основі чотирьох нейронних мереж, використаних у дослідженні, створено стековий ансамбль, який у ролі метамоделі (супервайзера) використовує логістичну регресію. Використання такого ансамблю дало змогу збільшити точність прогнозування дефектів ПЗ до 93,97 % у разі використання семи найважливіших метрик коду в ролі ознак, і до 81,84 % – у разі використання усієї сукупності ознак.

References

1. Akimova, Elena N., Konygin, V., Mezentsev, Ilya P., & Misilov, Vladimir E. (2021). A Survey on Software Defect Prediction Using Deep Learning. *Mathematics, 9*(11). https://doi.org/10.3390/math9111180
2. Albahli, Saleh. (2019). A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction. *Future Internet 11*(12), 246. https://doi.org/10.3390/fi11120246
3. Albahli, Saleh. (2019). A Deep Ensemble Learning Method for Effort-Aware Just-In-Time Defect Prediction. *Future Internet, 11*(12), 246. https://doi.org/10.3390/fi11120246
4. Bin, Liu, & Shihai, Wang. (2018). Software Defect Prediction Using Stacked Autoencoders. *Information and Software Technology, 96*, 94–111. https://doi.org/10.1016/j.infsof.2017.11.008
5. Dash, Ch. Sanjeev Kumar, Ajit Kumar Behera, Satchidananda Dehuri, & Sung-Bae Cho. (2016). Radial Basis Function Neural Networks: A Topical State-of-The-Art Survey. *Open Computer Science, 6*(1). https://doi.org/10.1515/comp-2016-0005
6. David, Lo, Xin, Xia, & Jianling, Sun. (2017). TLEL: A Two-Layer Ensemble Learning Approach for Just-In-Time Defect Prediction. *Information and Software Technology 87*, 206–2020. https://doi.org/10.1016/j.infsof.2017.03.007
7. Hrytsiuk, Yu. I., & Nemova, E. A. (2018). Management Features Process of Developing Software Requirements. *Scientific Bulletin of UNFU, 28*(8), 161–169. https://doi.org/10.15421/40280832
8. Jingfei, Chang, & Zhen, Wei. (2020). PathPair2Vec: An AST Path Pair-Based Code Representation Method for Defect Prediction. *Journal of Computer Languages, 59*. https://doi.org/10.1016/j.cola.2020.100979
9. Kedharnath, Nagella, & Vidya, S. (2021). Software Defect Estimation Using Machine Learning Algorithms. *International Journal of Recent Technology and Engineering, 10*(1), 204–208. https://doi.org/10.35940/ijrte.a5898.0510121
10. Manjula, C., & Lilly, Florence. (2018). Deep Neural Network Based Hybrid Approach for Software Defect Prediction Using Software Metrics. Cluster Computing, January. https://doi.org/10.1007/s10586-018-1696-z
11. Qiao, Lei, Xuesong, Li, Qasim, Umer, & Ping, Guo. (2020). Deep Learning Based Software Defect Prediction. Neurocomputing, 385, 100–110. https://doi.org/10.1016/j.neucom.2019.11.067
12. Rudenko, Oleg, Bezsonov, Oleksandr, & Romanyk, Oleksandr. (2019). Neural Network Time Series Prediction Based on Multilayer Perceptron. *Development Management, 17*(1), 23–34. https://doi.org/10.21511/dm.5(1).2019.03
13. Shakhovska, N., Yakovyna, V., Kryvinska, N. (2020). An improved software defect prediction algorithm using self-organizing

maps combined with hierarchical clustering and data preprocessing. In DEXA 2020, LNCS, 12391, 414–424.
14. Sherstinsky, Alex. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. Physica D: Nonlinear Phenomena, 404. https://doi.org/10.1016/j.physd.2019.132306
15. WANG, Pei, Cong, JIN, & He-he, GE. (2013). Mutual Information-Based Feature Selection Approach for Software Defect Prediction. *Journal of Computer Applications 32*(6), 1738–17340. https://doi.org/10.3724/sp.j.1087.2012.01738
16. Wang, Song, Taiyue, Liu, Jaechang, Nam, & Lin, Tan. (2018). Deep Semantic Feature Learning for Software Defect Prediction. IEEE Transactions on Software Engineering, 1–1. https://doi.org/10.1109/tse.2018.2877612
17. Zhao, Linchang, Zhaowei, Shang, Ling, Zhao, Taiping, Zhang, & Yuan, Yan Tang. (2019). Software Defect Prediction via Cost-Sensitive Siamese Parallel Fully-Connected Neural Networks. Neurocomputing, 352, 64–74. https://doi.org/10.1016/j.neucom.2019.03.076
18. Zhu, K., Zhang, N., Zhang, Q., Ying, S. & Wang, X. (2020). Software defect prediction based on non-linear manifold learning and hybrid deep learning techniques. *Computers, Materials & Continua, 65*(2), 1467–1486.

V. S. Yakovyna, I. I. Symets

Lviv Polytechnic National University, Lviv, Ukraine

SOFTWARE DEFECT PREDICTION USING NEURAL NETWORK ENSEMBLE

Defect prediction is one of the key challenges in software development and programming language research for improving software quality and reliability. The problem in this area is to properly identify the defective source code with high accuracy. This study describes the process of improving the accuracy of defect prediction in software modules and components using an ensemble of artificial neural networks. The combined data set obtained from the open PROMISE Software Engineering repository was used as the input data set. The data set contained 15,123 records, each of which consisted of 21 code metrics and a target binary variable that indicated defects in this software module or project. This study describes the use and implementation of the four most common neural networks, namely Multilayer Perceptron, neural network based on Radial-Basis Functions, Recurrent Neural Network and Long Short-Term Memory for software defect prediction using Python programming language and Keras open-source library. The performance of the models significantly depends on the value of hyperparameters, so the GridSearch function was used to determine the optimal parameters of the architecture of each neural network. The best prediction accuracy was achieved with recurrent neural networks and Long Short-Term Memory neural networks. The use of a previously static software reliability model developed by the authors, which reduces the set of features to the seven most important, allows increasing the accuracy of prediction by these methods to 89.97 % (for RNN) and 91.16 % (in the case of LSTM). The study describes the integration of developed neural networks into a stacked ensemble, which as a meta-model (supervisor) uses logistic regression to improve prediction results. The use of such an ensemble allowed increasing the accuracy of software defect prediction to 93.97 % in the case of using the seven most important code metrics as features, and up to 81.84 % in the case of using the whole set of features.

Keywords: reliability; binary classification; stacking ensemble; deep learning; code metrics.