МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

РИПІЧ Богдан Віталійович

Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж / Method for Software Development Project Management Based on Fuzzy Logic and Neural Networks

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма – Управління проєктами

Кваліфікаційна робота

Виконав студент групи КНУПм-21
Б.В. Рипіч
_____

Науковий керівник:
к.е.н., доцент Г.М. Гладій
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
В.о. завідувача кафедри
_____ Н.М. Васильків

ТЕРНОПІЛЬ – 2024

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «магістр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Управління проєктами

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
РИПІЧ Богдан Віталійович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж / Method for Software Development Project Management Based on Fuzzy Logic and Neural Networks
керівник роботи к.е.н., доцент Г.М. Гладій
затверджені наказом по університету від 12 грудня 2023 року № 753.
2. Строк подання студентом закінченої кваліфікаційної роботи 4 грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити
- опис предметної області;
- аналіз класичних статистичних моделей оцінювання зусиль розробки програмного забезпечення;
- аналіз методів машинного навчання для оцінювання зусиль розробки програмного забезпечення;
- постановка задачі дослідження;
- концепція методу управління проєктами з розробки програмного забезпечення;
- збір та попередня обробка даних;
- оцінювання подібності проєктів за допомогою нечітких правил;
- класифікація на основі нейронної мережі;
- дослідження продуктивності запропонованих рішень;
- порівняльний аналіз запропонованого рішення з відомими.

5. Перелік графічного матеріалу у роботі
- схема роботи запропонованого підходу;
- архітектура алгоритму MM-Fuzzy.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи.	до 01.01. 2024 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.03. 2024 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 20.05.2024 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 28.10. 2024 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 11.11.2024 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів.	до 25.11.2024 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту.	до 30.11.2024 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 04.12.2024 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії.	до 14.12. 2024 р.	

Студент _____ Б.В. Рипіч
підпис

Керівник роботи _____ к.е.н., доцент Г.М. Гладій
підпис

РЕЗЮМЕ

Кваліфікаційна робота на тему «Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж» на здобуття освітнього ступеня «Магістр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Управління проєктами» написана обсягом в 66 сторінок і містить 6 ілюстрацій, 3 таблиці, 1 додаток та 32 використаних джерел.

Метою кваліфікаційної роботи є підвищення ефективності управління проєктами з розробки програмного забезпечення через впровадження підходу до оцінювання зусиль, що базується на поєднанні нечіткої логіки та нейронних мереж.

Методи досліджень: методи кластеризації, методи видобування та відбору ознак, методи нечіткої логіки, методи машинного навчання та нейронних мереж, експериментальні методи, порівняльний аналіз.

Результати дослідження: вдосконалено метод управління проєктами з розробки програмного забезпечення, що забезпечує більш точне оцінювання зусиль завдяки попередній кластеризації проєктів на основі нейронних мереж та використанню нечітких правил для аналізу подібності між проєктами.

Результати роботи можуть бути використані для впровадження розробленого методу у реальних умовах управління програмними проєктами. Запропонований підхід дозволяє значно підвищити точність оцінки зусиль, що є критично важливим для планування ресурсів, бюджету та часу виконання програмних проєктів.

Ключові слова: УПРАВЛІННЯ ПРОЄКТАМИ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, НЕЧІТКА ЛОГІКА, НЕЙРОННА МЕРЕЖА, ОЦІНЮВАННЯ ЗУСИЛЬ, КЛАСТЕРИЗАЦІЯ ПРОЄКТІВ.

ABSTRACT

Qualification work on the topic «Method for Software Development Project Management Based on Fuzzy Logic and Neural Networks» for Master's degree on speciality 122 «Computer Science» educational and professional program «Project Management» is written on 66 pages and it contains 6 figures, 3 tables, 1 annex and 32 sources.

The purpose of this qualification work is to improve the efficiency of software development project management by implementing an effort estimation approach based on the combination of fuzzy logic and neural networks.

Research methods: clustering methods, feature extraction and selection methods, fuzzy logic methods, machine learning and neural network methods, experimental methods, and comparative analysis.

Research results: the method for software development project management has been improved, enabling more accurate effort estimation through preliminary project clustering based on neural networks and the application of fuzzy rules for analyzing project similarity.

The results of this work can be applied to the real-world management of software projects. The proposed approach significantly enhances the accuracy of effort estimation, which is critically important for planning resources, budgets, and timelines for software projects.

Keywords: PROJECT MANAGEMENT, SOFTWARE DEVELOPMENT, FUZZY LOGIC, NEURAL NETWORK, EFFORT ESTIMATION, PROJECT CLUSTERING.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі успішний бізнес-план для розробки програмного забезпечення створюється компаніями, які орієнтуються на випуск якісних продуктів для конкурентних глобальних ринків [19]. Для того, щоб задовольнити високі вимоги, процес розробки програмного забезпечення має бути добре організований і контрольований. Важливо оцінювати програмні продукти як з кількісної, так і з якісної точки зору, враховуючи зусилля, ресурси, витрати та інші аспекти процесу розробки [17].

Одним із ключових, але складних завдань в управлінні проєктами є оцінювання зусиль, необхідних для розробки програмного забезпечення (Software Development Effort Estimation, SDEE) [8, 20, 27]. Це процес прогнозування обсягу зусиль, часу та кількості людей, які потрібні для реалізації проєкту. SDEE є важливою частиною управління проєктами, адже саме на основі цієї оцінки керівництво вирішує, чи схвалити проєкт, чи ні [13].

Завищені оцінки зусиль можуть призвести до нераціонального використання ресурсів і затримок у виконанні, тоді як недооцінювання часто спричиняє перевитрати бюджету, нестачу персоналу та затримки [4, 24].

Багато бізнес-процесів, таких як планування ітерацій, проєктування, ціноутворення, аналіз інвестицій та складання бюджетів, потребують точного прогнозування зусиль [9, 23]. Однак, через унікальні особливості кожного проєкту, ця задача є складною і вимагає особливої уваги та ретельного аналізу [12, 18].

Мета і завдання дослідження. Метою кваліфікаційної роботи є підвищення ефективності управління проєктами з розробки програмного забезпечення через впровадження підходу до оцінювання зусиль, що базується на поєднанні нечіткої логіки та нейронних мереж.

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести огляд та порівняння сучасних методів оцінювання зусиль, виявити їхні переваги та недоліки;

– розробити метод управління проєктами з розробки програмного забезпечення на базі рекурентної нейронної мережі для підвищення точності та швидкості навчання;

– виконати кластеризацію проєктів на основі алгоритму K-means для більш точного розподілу проєктів за розміром та складністю;

– здійснити видобування значущих ознак з набору даних для зниження обчислювальної складності та підвищення точності моделі;

– виконати попередню обробку ознак та розробити нечіткі правила для оцінювання подібності між проєктами, що дозволить краще враховувати специфіку кожного проєкту під час оцінювання зусиль;

– провести експериментальні дослідження, проаналізувати показники продуктивності запропонованої моделі;

– провести порівняльний аналіз продуктивності запропонованого методу з відомими рішеннями.

Об’єктом дослідження є процес оцінювання зусиль на розробку програмного забезпечення, що включає аналіз, прогнозування та розрахунок необхідних ресурсів для виконання програмних проєктів.

Предметом дослідження є методи та алгоритми оцінювання зусиль на розробку програмного забезпечення, зокрема методи кластеризації, видобування та відбору ознак, нейронні мережі для прогнозування зусиль та нечітка логіка для аналізу подібності між проєктами.

Методи дослідження, які використані в роботі: методи кластеризації – для групування проєктів за розміром та складністю; методи видобування та відбору ознак – для вибору найважливіших ознак з великого обсягу даних; методи нечіткої логіки – для оцінювання подібності між проєктами; методи машинного навчання та нейронних мереж – для підвищення точності та швидкості прогнозування зусиль; експериментальні методи – для оцінювання продуктивності запропонованого підходу, зокрема аналіз точності, чутливості, специфічності та часу навчання; порівняльний аналіз – для підтвердження переваг та ефективності запропонованого підходу.

Наукова новизна одержаних результатів. Вдосконалено метод управління проєктами з розробки програмного забезпечення, що забезпечує більш точне прогнозування зусиль завдяки попередній кластеризації проєктів на основі нейронних мереж та використанню нечітких правил для аналізу подібності між проєктами.

Практичне значення отриманих результатів полягає в можливості застосування розробленого методу у реальних умовах управління програмними проєктами. Запропонований підхід дозволяє значно підвищити точність оцінювання зусиль, що є критично важливим для планування ресурсів, бюджету та часу виконання програмних проєктів.

Публікації та апробація КР. Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах:

– міжнародної мультидисциплінарної наукової інтернет-конференції «Світ наукових досліджень» (випуск 35), 20-21 листопада 2024 р.;

– міжнародної наукової Інтернет-конференції «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (випуск 94), 11-12 грудня 2024 р.

Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ УПРАВЛІННЯ ПРОЄКТАМИ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Опис предметної області

Оцінювання зусиль для розробки програмного забезпечення є важливою складовою управління програмними проєктами, оскільки від неї залежить ефективне планування ресурсів, бюджетування та дотримання строків виконання завдань. У сучасних умовах, коли проєкти стають дедалі масштабнішими і складнішими, а вимоги до швидкості розробки зростають, точність оцінювання зусиль набуває особливого значення. Необхідність точного оцінювання зусиль обумовлена тим, що помилки у прогнозуванні можуть призвести до недоцільного використання ресурсів, перевищення бюджету, затримок у строках виконання і, як наслідок, до значних фінансових втрат [17, 24].

За останні роки було розроблено різноманітні методи оцінювання зусиль, починаючи від класичних статистичних моделей і закінчуючи сучасними методами машинного навчання. Серед основних методів, які використовуються для SDEE, є класичні статистичні підходи, такі як лінійна та множинна регресія, що дозволяють побудувати математичні моделі на основі історичних даних проєктів. Однак ці методи мають обмежену точність при застосуванні до проєктів з нелінійними залежностями або великою кількістю параметрів [18]. Крім того, традиційні моделі мають труднощі з обробкою великих обсягів даних і не враховують змінні фактори, що можуть впливати на зусилля упродовж розробки [6].

Сучасні підходи для оцінювання зусиль включають методи машинного навчання, які дозволяють враховувати складні залежності та працювати з великими даними. Зокрема, ансамблеві методи, такі як Random Forest та Gradient Boosting, забезпечують високу точність завдяки комбінуванню результатів кількох моделей [14]. Нейронні мережі, зокрема глибокі нейронні мережі та рекурентні нейронні мережі, також застосовуються для SDEE завдяки їхній здатності моделювати нелінійні зв’язки між параметрами проєкту [5].

Незважаючи на високу ефективність, ML-методи потребують великих обчислювальних ресурсів, а також якісних і великих наборів даних для тренування моделей, що може бути складним для невеликих проєктів або компаній [16].

Актуальність задачі полягає в необхідності розробки таких підходів до SDEE, які поєднували б високу точність із можливістю роботи в умовах обмежених ресурсів та динамічних змін у проєктах. Одним із перспективних напрямків є поєднання методів машинного навчання з оптимізаційними алгоритмами, що дозволяє вибирати оптимальні ознаки та знижувати обчислювальну складність [1]. Інтеграція нейронних мереж із нечіткою логікою також сприяє підвищенню точності за рахунок обробки невизначеності та розмитих даних, які часто виникають у процесі розробки програмного забезпечення [7].

Таким чином, розробка та вдосконалення методів оцінювання зусиль для розробки програмного забезпечення є актуальною задачею, яка потребує врахування як класичних підходів, так і сучасних технологій машинного навчання та оптимізації. Це дозволить створити ефективніші та гнучкіші системи управління проєктами, здатні адаптуватися до змінних умов та забезпечувати своєчасне виконання проєктів у межах запланованих ресурсів [12].

1.2 Аналіз класичних статистичних моделей оцінювання зусиль розробки програмного забезпечення

Класичні статистичні моделі є одними з найстаріших та найпоширеніших методів оцінювання зусиль у розробці програмного забезпечення. Вони базуються на застосуванні регресійного аналізу та емпіричних моделей, які використовують історичні дані для прогнозування зусиль, необхідних для виконання певних завдань у програмних проєктах. Основна мета цих моделей – виявлення залежностей між характеристиками проєкту та обсягом зусиль, що дозволяє побудувати математичну модель для оцінювання зусиль.

До таких методів належать:

1. Лінійна регресія. Це один з найпоширеніших методів, який базується на припущенні лінійного зв’язку між незалежними змінними (вхідні характеристики проєкту) та залежною змінною (зусилля). Модель оцінює коефіцієнти, які дозволяють побудувати лінійну функцію для прогнозування зусиль. Лінійна регресія є простою у використанні та інтерпретації, але не завжди може відобразити складні зв’язки між змінними [17].

2. Множинна регресія. У випадках, коли на зусилля впливають кілька факторів, використовується множинна регресія. Вона дозволяє враховувати кілька незалежних змінних, що робить її більш гнучкою у порівнянні з лінійною регресією. Множинна регресія краще підходить для комплексних проєктів, де на обсяг зусиль впливають різноманітні характеристики, такі як розмір коду, досвід команди, складність проєкту та інші параметри [23].

3. COCOMO (Constructive Cost Model). Це емпірична модель, розроблена Баррі Бемом у 1980-х роках, яка базується на статистичному аналізі великої кількості програмних проєктів. Модель COCOMO використовує низку показників для оцінювання зусиль, таких як розмір програми (у тисячах рядків коду) та коефіцієнти, що враховують фактори складності проєкту [13]. COCOMO існує в кількох варіантах, включаючи Basic COCOMO, Intermediate COCOMO та COCOMO II, кожен з яких призначений для різних рівнів деталізації та складності проєктів. Ця модель є популярною завдяки простоті використання, але має обмеження у випадку нових технологій та специфічних проєктів, де точні дані важко отримати [19].

4. Function Point Analysis (FPA). Цей метод базується на оцінці функціональності, яку повинна забезпечувати програмна система. FPA розглядає проєкт з точки зору кількості функціональних точок (функцій, екранних форм, звітів тощо), що дозволяє оцінити обсяг роботи незалежно від технології реалізації. Метод FPA є корисним у випадках, коли розмір коду не є хорошим показником складності проєкту. Однак FPA вимагає великого досвіду та знань для точного оцінювання, що може бути складним для нових проєктів [8].

Переваги класичних статистичних моделей:

1. Простота та інтуїтивність. Багато статистичних моделей є простими у використанні та зрозумілими, що дозволяє легко пояснювати результати та використовувати їх для ухвалення рішень.

2. Швидка обробка даних. Класичні моделі добре підходять для роботи з невеликими наборами даних, оскільки їхні обчислення відносно прості та не потребують значних ресурсів.

3. Основа для більш складних моделей. Статистичні моделі часто використовуються як базова точка для розробки більш складних методів та алгоритмів машинного навчання.

Недоліки класичних статистичних моделей:

1. Лінійні обмеження. Більшість класичних моделей, таких як лінійна та множинна регресія, передбачають лінійні зв’язки між змінними, що обмежує їхню здатність враховувати складні, нелінійні взаємозв’язки.

2. Залежність від якості даних. Точність прогнозів залежить від якості та кількості історичних даних. Неповні, застарілі або непослідовні дані можуть суттєво знижувати точність оцінок.

3. Недооцінювання динамічних умов. Статистичні моделі часто не враховують зміни у середовищі проєкту, такі як зміна технологій або вимог, що може призводити до неточних прогнозів.

Розглянемо області застосування класичних статистичних моделей.

Класичні статистичні моделі найчастіше застосовуються у випадках, коли проєкти мають доступні історичні дані та коли зв’язки між змінними є відносно простими та стабільними. Вони є корисними для початкового оцінювання зусиль, планування бюджету та оцінювання строків виконання проєктів [22]. Їх можна застосовувати для невеликих та середніх проєктів, де точність оцінки не є критичною або коли необхідно швидко отримати попередню оцінку.

Проте з розвитком програмного забезпечення, збільшенням обсягів даних та складності проєктів, класичні методи почали поступатися місцем більш

сучасним підходам, таким як методи машинного навчання, глибокі нейронні мережі та метаевристичні алгоритми [10, 29, 30, 35].

У статті [29] досліджено впровадження технологій машинного навчання та штучного інтелекту в управління програмними проєктами. Основна увага приділяється автоматизації процесів, таких як планування, оцінювання витрат, оптимізація розподілу ресурсів та управління ризиками. Автор описує можливості сучасних мовних моделей, зокрема GPT-4, для аналізу текстових даних, генерації рекомендацій та підтримки управлінських рішень у реальному часі.

У статті [30] проаналізовано трансформаційний вплив штучного інтелекту на управління проєктами. Описано, як технології ШІ змінюють традиційні підходи до управління, забезпечуючи автоматизацію рутинних завдань, аналіз даних та підвищення ефективності прийняття рішень.

Стаття [35] акцентує увагу на використанні гнучких моделей машинного навчання для підвищення ефективності контролю за проєктами, особливо в умовах невизначеності. Інтеграція машинного навчання в процеси контролю за проєктами значно покращує можливості прогнозування, ідентифікації ризиків та управління ресурсами. Інноваційні підходи дозволяють проєктним командам бути гнучкішими та краще реагувати на зміни, що сприяє успішному виконанню проєктів навіть в умовах високої невизначеності.

Методи штучного інтелекту дозволяють враховувати більш складні взаємозв’язки між параметрами проєкту та адаптуватися до нових умов. Це створює необхідність у розробці нових підходів, які можуть поєднувати простоту класичних моделей з гнучкістю сучасних алгоритмів, що є важливим завданням для подальшого розвитку оцінювання зусиль у розробці програмного забезпечення.

1.3 Аналіз методів машинного навчання для оцінювання зусиль розробки програмного забезпечення

Методи машинного навчання дедалі частіше використовуються для оцінювання зусиль розробки програмного забезпечення, оскільки вони дозволяють автоматизувати та підвищити точність прогнозів завдяки здатності обробляти великі обсяги даних і враховувати складні взаємозв'язки між характеристиками проєкту. Сучасні ML-методи для оцінювання зусиль розробки програмного забезпечення включають методи ансамблевого навчання, нейронні мережі та гібридні підходи, які комбінують кілька алгоритмів для підвищення ефективності.

1. Ансамблеві методи – об'єднують результати кількох моделей для підвищення точності прогнозів. Найпопулярнішими ансамблевими підходами є:

1.1. Random Forest. Створює набір дерев рішень на випадкових підмножинах даних і об'єднує їх результати. Random Forest є стійким до переобучення та добре підходить для роботи з великими наборами характеристик проєкту [23].

1.2. Gradient Boosting. Побудова ансамблю дерев рішень, кожне з яких коригує помилки попереднього. Цей метод має високу точність, проте потребує значних обчислювальних ресурсів для тренування на великих даних [14].

2. Нейронні мережі, зокрема глибокі нейронні мережі та рекурентні нейронні мережі є потужними інструментами для прогнозування зусиль завдяки здатності моделювати складні та нелінійні залежності. До поширених нейронних мереж для SDEE належать:

2.1. Глибокі нейронні мережі. DNN використовуються для аналізу складних залежностей між різними характеристиками проєкту, що забезпечує високу точність. Однак ці моделі мають високу обчислювальну складність і потребують великих обсягів даних [2].

2.2. Рекурентні нейронні мережі. Використовуються для послідовних даних та запам'ятовування попередніх значень, що є корисним для

довгострокового прогнозування зусиль на основі часових послідовностей проєктних характеристик. Проте класичні RNN можуть мати проблему зникання градієнта, що впливає на точність оцінки [5].

2.3. Гібридні нейронні мережі (наприклад, HTRR-RNN). Рекурентні радіальні нейронні мережі з гіперболічною тангенсною активацією (HTRR-RNN) усувають проблему зникання градієнта та покращують продуктивність моделі при прогнозуванні зусиль у глибоких мережах. Такі моделі є ефективними у гнучкості та точності прогнозування [16].

3. Гібридні та метаевристичні методи. Гібридні методи поєднують кілька підходів для досягнення точних та надійних результатів.

Приклади таких методів:

3.1. Гібридні моделі на основі нейронних мереж та нечіткої логіки (Fuzzy Logic). Поєднання нейронних мереж із нечіткою логікою допомагає враховувати невизначеність та нелінійність у даних, що є корисним для складних проєктів, де змінні можуть мати розмиті значення [7].

3.2. Оптимізаційні алгоритми (наприклад, Genetic Algorithm, Grey Wolf Optimization). Використовуються для вибору оптимальних характеристик, що знижує обчислювальну складність та покращує точність моделей. Наприклад, Grey Wolf Optimization та Genetic Algorithm добре справляються з пошуком оптимальних параметрів моделей, що покращує якість оцінки [1].

Проведемо аналіз наукових робіт, в яких використано машинне навчання для оцінювання зусиль розробки програмного забезпечення.

У роботі [21] використовували модель, засновану на ансамблевому навчанні, разом з рекурсивним усуненням ознак для оцінювання зусиль розробки програмного забезпечення (SEE). Підхід прогнозував зусилля, використовуючи критерії, такі як розмір та вартість, на основі ранжування та відбору ознак. Метод ранжування ознак на основі рекурсивного усунення та ансамблева модель на основі ранжованих ознак для SEE були двома етапами цієї схеми. Ранжування та відбір характеристик і фактичних витрат у SEE для ансамблевого навчання відбувалися на першому етапі. На другому етапі розглядалося використання всіх

семи класифікаторів для ансамблевого навчання. Сім класифікаторів класифікували кожен зразок набору даних, а процес голосування визначав кінцеву мітку класу. За результатами симуляцій ця структура перевершувала попередні методи. Однак цей підхід мав недолік тривалого часу обчислень.

У роботі [11] розробили неалгоритмічний підхід, який отримав назву гібридної моделі хвильової нейронної мережі (WNN) разом з метаевристичним алгоритмом для оцінювання зусиль. Два метаевристичні алгоритми, що використовувалися у цій структурі, були алгоритмами світлячків та кажанів. Morlet та Gaussian функції були двома варіантами хвильових функцій, які використовувалися як функції активації в WNN. Традиційна WNN, яка не оптимізувалася жодним метаевристичним підходом, поступалася WNN, інтегрованій з метаевристичними алгоритмами в SEE. Статистичні результати були підтверджені непараметричним статистичним тестом за допомогою IBM SPSS. Проте виникла проблема неефективного оброблення відсутніх даних.

У роботі [3] реалізували нечітку нейронну мережу, яка формувала нечіткі правила, що сприяли створенню експертної системи на основі інтерпретованих правил. Завдяки елементам складності проєкту це сприяло прогнозуванню годин розробки програмного забезпечення. Підвищення ефективності у процесі нечіткої класифікації даних було здійснено за допомогою методу інкрементальної щільності, який використовувався на першому рівні методу. Крім того, функція активації Leaky-ReLU підвищила чутливість нейронів у нейронній мережі. Експериментальні результати моделі були ефективними у прогнозуванні зусиль, необхідних для розробки програмного забезпечення. Основним недоліком цього методу була деградація продуктивності у великих проєктах.

У роботі [22] розробили гібридну методологію, яка поєднувала метод прийняття рішень за багатьма критеріями для обробки невизначеності та методи машинного навчання для роботи з неточністю, для точного прогнозування зусиль. Для ранжування ознак ефективно використовувався метод нечіткої ієрархічного аналізу (FAHP). Для оцінювання зусиль ранжування інтегрувалося

з виваженою підтримуючою векторною машиною з ядром найменших квадратів. Ця модель емпірично демонструвалася на сховищах даних для SEE. У порівнянні з оптимізацією на основі колонії бджіл та базовою моделлю з радіальною базовою функцією (RBF), точне оцінювання зусиль досягалося завдяки комбінації ваг, створених FAHP, і ядра RBF. Проте модель була вразливою до нерелевантної інформації.

У роботі [15] розробили та порівняли три спеціальні моделі нечіткої логіки (FLM) для прогнозування SEE: Mamdani, Sugeno з постійним виходом та Sugeno з лінійним виходом. Використання регресійного аналізу (RA) підвищило ефективність FLM. Кожен навчальний набір даних створював модель множинної лінійної регресії. Той самий набір даних використовувався для створення FLM. Для навчання та тестування моделей використовувалися дані з набору проєктів від International Software Benchmarking Standards Group (ISBSG). Згідно з результатами, використання RA значно покращило продуктивність моделі. Однак було виявлено проблему перенавчання та недонавчання в цьому методі.

У роботі [10] представили підхід до оцінювання зусиль для розробки програмного забезпечення за допомогою моделі Omni Ensemble Learning. Спочатку використовувався набір даних для оцінювання зусиль розробки програмного забезпечення та виконувалася його попередня обробка для видалення нерелевантних атрибутів. Потім, оптимальні ознаки вибиралися за допомогою генетичного алгоритму. Далі статична ансамблева модель використовувалася для вибору найкращого класифікатора для оцінювання зусиль. Після цього на етапі динамічного ансамблевого відбору вибирався підмножина класифікаторів. Розроблений метод був статистично проаналізований за допомогою тесту Вілкоксона. Модель оцінювала зусилля з мінімальною помилкою та витратою часу. Однак ансамблева модель не була масштабованою щодо підкласифікаторів, що обмежувало точність оцінки зусиль.

У роботі [16] запропонували параметричне оцінювання зусиль для програмного забезпечення, використовуючи регресійну модель. Для аналізу

програмного забезпечення ознаки вибиралися за допомогою моделі регресії з оператором найменшого скорочення та відбору (LASSO). Потім вибрані фактори оцінювалися за допомогою регресійних формул, розроблених у моделі множинної лінійної регресії. Далі коригувальні коефіцієнти для оцінювання зусиль оптимізувалися шляхом налаштування оцінених факторів. Запропонована модель досягла кращих результатів порівняно з існуючими роботами під час статистичного аналізу продуктивності з точки зору показника прогнозування та середньоквадратичної помилки. Однак схожість між ознаками проєктів не була врахована до оцінки зусиль, що ускладнювало процес та обмежувало надійність моделі.

У роботі [1] дослідили кластерний аналіз для покращення оцінювання зусиль розробки програмного забезпечення. Спочатку набір даних кластеризувався за промисловим сектором, типом організації, мовою, розміром та платформою розробки. Паралельно дані кластеризувалися за допомогою алгоритму K-means. Після кластеризації видалялися викиди. Далі зусилля розробки програмного забезпечення оцінювалися за допомогою алгоритму машинного навчання. Продуктивність оцінювання зусиль була покращена порівняно з іншими методами з мінімальною середньоквадратичною помилкою та абсолютною помилкою. Проте випадковий вибір кластерів у використаному алгоритмі K-means знижував якість кластеризації та обмежував ефективність оцінювання зусиль.

У роботі [7] розробили покращений підхід до оцінювання зусиль для програмних проєктів, використовуючи алгоритм оптимізації сірих вовків (GWO). Початкова популяція алгоритму GWO була ініціалізована і коефіцієнти проєктів використовувалися для оцінювання. Паралельно зусилля прогнозувалися з урахуванням функції придатності, що мінімізувала середньоквадратичну помилку. Ефективність алгоритму GWO оцінювалася шляхом застосування чотирьох інших методів оптимізації, включаючи оптимізацію прерійних собачок, оптимізацію зебри, оптимізацію білих акул та оптимізацію на основі полум’я метеликів. Експериментальні результати

показали, що запропонований підхід досягнув покращеної продуктивності щодо дисперсії та середньоквадратичної помилки. Однак набір даних не був попередньо оброблений, і важливі ознаки для оцінювання зусиль не були обрані в цьому дослідженні, що обмежує точність оцінки.

У роботі [26] запропонували оцінювання зусиль для програмних проєктів, використовуючи еволюційні методи з багатьма цілями. Базовий набір даних було підготовлено шляхом масштабування та видалення пропущених значень. Потім використовувався метод аргументації на основі випадків, де відхилення між ознаками оцінювалися за допомогою метрики евклідової відстані. Далі вибиралися найближчі проєкти за ознаками, і зусилля оцінювалися за допомогою генетичного алгоритму. Запропонований підхід оцінював зусилля з нижчою зворотною збалансованою відносною помилкою та середньою абсолютною помилкою. Однак проблема повільної та передчасної збіжності генетичного алгоритму не була вирішена в цій роботі, що спричинило неефективне оцінювання.

Переваги методів машинного навчання для оцінювання зусиль розробки програмного забезпечення:

1. Гнучкість та адаптивність. Більшість методів ML можуть адаптуватися до різноманітних наборів даних та змінних умов у проєкті, що робить їх універсальними інструментами для SDEE.

2. Висока точність. Використання глибоких нейронних мереж та ансамблевих методів дозволяє досягти високої точності у прогнозуванні зусиль завдяки моделюванню складних взаємозв’язків.

3. Масштабованість. ML-методи можуть працювати з великими обсягами даних, що особливо корисно для великих проєктів та великих компаній.

Недоліки методів машинного навчання для оцінювання зусиль розробки програмного забезпечення:

1. Висока обчислювальна складність. Глибокі мережі та складні ансамблеві моделі потребують великих обчислювальних ресурсів, що може бути недоступним для невеликих команд чи проєктів.

2. Необхідність великих наборів даних. Для тренування більшості моделей ML потрібні великі обсяги історичних даних, що може бути проблематичним для нових проєктів без достатньої кількості прикладів.

3. Труднощі інтерпретації. Деякі моделі, особливо глибокі нейронні мережі, є складними для інтерпретації, що може ускладнювати прийняття рішень на основі результатів.

Методи машинного навчання значно розширюють можливості оцінювання зусиль для розробки програмного забезпечення завдяки їхній точності та здатності працювати з великими даними. Кожен підхід має свої переваги та недоліки, і вибір методу залежить від специфіки проєкту, доступності даних та обчислювальних ресурсів. Поєднання нейронних мереж із оптимізаційними та нечіткими підходами, а також використання ансамблевих моделей, дозволяє досягти оптимального балансу між точністю та ресурсомісткістю, забезпечуючи ефективне управління проєктами та оптимізацію ресурсів у розробці програмного забезпечення.

1.4 Постановка задачі дослідження

Проблема оцінювання зусиль для розробки програмного забезпечення залишається відкритою через наявність складних взаємозв’язків між численними параметрами проєкту, що часто призводить до неточних прогнозів [6]. Сучасні методи оцінювання стикаються з труднощами при обробці непослідовних, неповних, розріджених та недостатньо задокументованих наборів даних, що є однією з основних причин неточності результатів [28]. Існуючі моделі не можуть ефективно обробляти нелінійні зв’язки між характеристиками проєктів та обсягом зусиль, що знижує їх ефективність і надійність [5].

Глибокі нейронні мережі дозволили покращити точність оцінки, проте їх використання обмежується через недоліки, такі як тривалий час навчання, перенавчання та недонавчання [25].

Недоліки існуючих підходів до SDEE можна сформулювати наступним чином:

– відсутність достатньої уваги до диференціації функціональності програмного забезпечення, що призводить до неточних оцінок зусиль для програмних проєктів;

– відсутність групування проєктів за специфікаціями, що ускладнює обробку неточних та неякісних даних;

– невідповідний вибір ознак, що призводить до навчання моделей на непідходящих даних і, відповідно, до неточних прогнозів;

– використання необроблених даних з історичних проєктів без попередньої обробки, що обмежує точність класифікації.

З огляду на вищезазначені недоліки, основна мета цього дослідження полягає в розробці підходу для оцінювання зусиль розробки програмного забезпечення, який дозволить підвищити точність прогнозування, зменшити час навчання та забезпечити адаптацію моделі до характеристик проєктів. Для досягнення цієї мети необхідно дослідити методи класифікації та обробки даних, що враховують специфіку програмних проєктів, нелінійні зв’язки між параметрами, а також забезпечують попередню обробку даних для підвищення якості класифікації [32].

Висновки до розділу 1

1. Точне оцінювання зусиль є ключовою для ефективного управління програмними проєктами, оскільки вона дозволяє раціонально розподілити ресурси, спланувати бюджет та встановити реалістичні терміни. Сучасні тенденції до збільшення складності проєктів і швидкості розробки підвищують потребу в точних прогнозах, оскільки неточні оцінки можуть призводити до перевитрати ресурсів, затримок і фінансових втрат. Таким чином, розробка точних методів для оцінювання зусиль є важливою задачею для галузі.

2. Класичні статистичні методи, такі як лінійна та множинна регресія, забезпечують основу для оцінювання зусиль на основі історичних даних і є зрозумілими у використанні. Проте ці підходи мають обмежену ефективність для великих або складних проєктів, де наявні нелінійні залежності між параметрами. У сучасних умовах вони поступаються місцем більш гнучким і точним методам машинного навчання, здатним краще враховувати різні фактори.

3. Методи машинного навчання, такі як ансамблеві моделі та нейронні мережі, стали важливим інструментом для точнішого оцінювання зусиль завдяки їх здатності моделювати складні зв’язки між даними. Вони дозволяють покращити точність прогнозів, але вимагають значних обчислювальних ресурсів та великих наборів даних для навчання. Подальший розвиток цих методів, у тому числі їх інтеграція з оптимізаційними алгоритмами, дозволяє створювати більш точні та адаптивні моделі для оцінювання зусиль у динамічних умовах програмних проєктів.

2 УПРАВЛІННЯ ПРОЄКТАМИ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Концепція методу управління проєктами з розробки програмного забезпечення

Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж є інноваційним підходом, який поєднує переваги методів нечіткої логіки для роботи з невизначеністю та потужність нейронних мереж у прогнозуванні складних залежностей. Цей метод розроблено для забезпечення більш точного оцінювання зусиль, часу та ресурсів, необхідних для реалізації програмного проєкту, що дозволяє ефективніше керувати процесами розробки, оптимізувати витрати та зменшувати ризики, пов’язані з невизначеністю та змінами у проєкті [34].

Основні елементи концепції [34]:

1. Нечітка логіка для роботи з невизначеністю.

1.1. Нечіткі множини та правила. Нечітка логіка використовується для формалізації невизначених та розмитих понять, таких як "великий", "складний", "середній", що часто виникають у процесі управління програмними проєктами. Це дозволяє створити нечіткі множини для різних параметрів проєкту (наприклад, досвід команди, складність задач, розмір бази даних) та формувати правила на основі цих множин.

1.2. Мамдані-метод. Найбільш поширеним способом використання нечіткої логіки є Мамдані-метод, який дозволяє створювати правила у вигляді "Якщо-то" (If-Then). Наприклад, "Якщо проєкт має високу складність та великий розмір, то необхідний високий рівень зусиль". Нечітка система на основі цих правил дозволяє автоматизувати процес прийняття рішень при розподілі ресурсів та плануванні строків.

1.3. Дефазифікація. Після обчислення нечітких значень відбувається процес дефазифікації, який перетворює результати у чіткі значення, які

використовуються для подальшого управління проєктом. Це дозволяє приймати конкретні рішення щодо планування ресурсів, складу команди та розподілу часу.

2. Нейронні мережі для прогнозування та адаптації.

2.1. Рекурентні нейронні мережі (RNN). Рекурентні нейронні мережі, зокрема гіперболічні тангенсні рекурентні радіальні нейронні мережі (HTRR-RNN), використовуються для моделювання складних залежностей між параметрами проєкту та прогнозування зусиль. RNN добре підходять для обробки послідовних даних та врахування попередніх результатів, що дозволяє їм краще моделювати динаміку проєктів у часі.

2.2. Навчання на основі історичних даних. Нейронна мережа навчається на основі історичних даних проєктів, що включають такі параметри, як розмір програмного продукту, досвід команди, тривалість попередніх проєктів, складність завдань тощо. Це дозволяє їй виявляти приховані закономірності та робити точніші прогнози зусиль для нових проєктів.

2.3. Адаптація до змін у проєкті. Використання нейронних мереж дозволяє адаптуватися до змінних умов проєкту, таких як зміна вимог або збільшення складності. Мережа може оновлювати свої параметри на основі нових даних, що дозволяє керівникам проєктів оперативно реагувати на зміни та коригувати прогноз зусиль.

3. Інтеграція нечіткої логіки та нейронних мереж.

3.1. Поєднання переваг обох підходів. Метод поєднує здатність нечіткої логіки обробляти невизначеність із можливістю нейронних мереж моделювати складні нелінійні залежності. Наприклад, на етапі кластеризації проєктів нечітка логіка може використовуватися для визначення подібності між проєктами на основі різних критеріїв, а нейронні мережі — для точного прогнозування зусиль на основі виділених ознак.

3.2. Нечітко-нейронний підхід. Це підхід, у якому нечітка логіка використовується на початкових етапах для формування правил та розподілу даних, а нейронна мережа застосовується для прогнозування та корекції

результатів. Така інтеграція дозволяє зменшити обчислювальну складність процесу, підвищити точність та забезпечити більш гнучке управління проєктами.

Переваги та практичні аспекти методу:

1. Підвищення точності оцінювання зусиль. Використання нечіткої логіки для роботи з невизначеними даними та адаптивної нейронної мережі дозволяє досягти більш точного оцінювання зусиль. Це забезпечує ефективне планування ресурсів та уникнення проблем, пов’язаних з недооцінюванням або переоцінюванням зусиль.

2. Оптимізація використання ресурсів. Метод дозволяє керівникам проєктів ефективніше розподіляти ресурси, враховуючи специфіку кожного проєкту та його потреби. Це допомагає уникнути перевантаження команд або нераціонального використання ресурсів.

3. Адаптивність до динамічних змін. Завдяки можливості нейронних мереж адаптуватися до нових даних та змін у проєкті, метод дозволяє швидко реагувати на зміни вимог або умов виконання проєкту. Це особливо важливо для великих та комплексних проєктів, де зміни часто є невід’ємною частиною процесу розробки.

4. Прийняття обґрунтованих рішень. Завдяки використанню нечітких правил та нейронних мереж, метод дозволяє приймати більш обґрунтовані рішення щодо строків виконання, розподілу ресурсів та складу команди. Це сприяє підвищенню продуктивності та успішному завершенню проєктів.

Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж є потужним інструментом для оцінювання зусиль у сучасних програмних проєктах. Він забезпечує високу точність та гнучкість у змінних умовах, що дозволяє зменшити ризики, пов’язані з невизначеністю, та підвищити ефективність реалізації програмних проєктів. Це робить його особливо цінним для великих ІТ-компаній, які працюють у конкурентному середовищі та прагнуть оптимізувати свої процеси управління проєктами.

Отже, для вирішення цієї задачі пропонується підхід покращення оцінювання зусиль на основі правил-аналогій з використанням нейронних мереж в управлінні програмним проєктом. Спочатку, з урахуванням розміру проєкту, відбувається поділ вхідних проєктів за допомогою TD-KMeans. З розподілених проєктів виділяються найважливіші та найінформативніші ознаки. Необхідна та релевантна кількість ознак обирається з виділених ознак за допомогою алгоритму LF-RHO. Далі видалення повторюваних даних та числове перетворення є двома етапами, що включені до процесу попередньої обробки обраних ознак.

За допомогою алгоритму MM-Fuzzy оцінюється подібність ознак та генеруються певні правила. Кількість зусиль, необхідних для розробки програмного забезпечення, ефективно визначається за допомогою класифікаційного алгоритму HTRR-RNN, який ґрунтується на правилах.

На рисунку 2.1 представлено схему роботи запропонованого підходу.

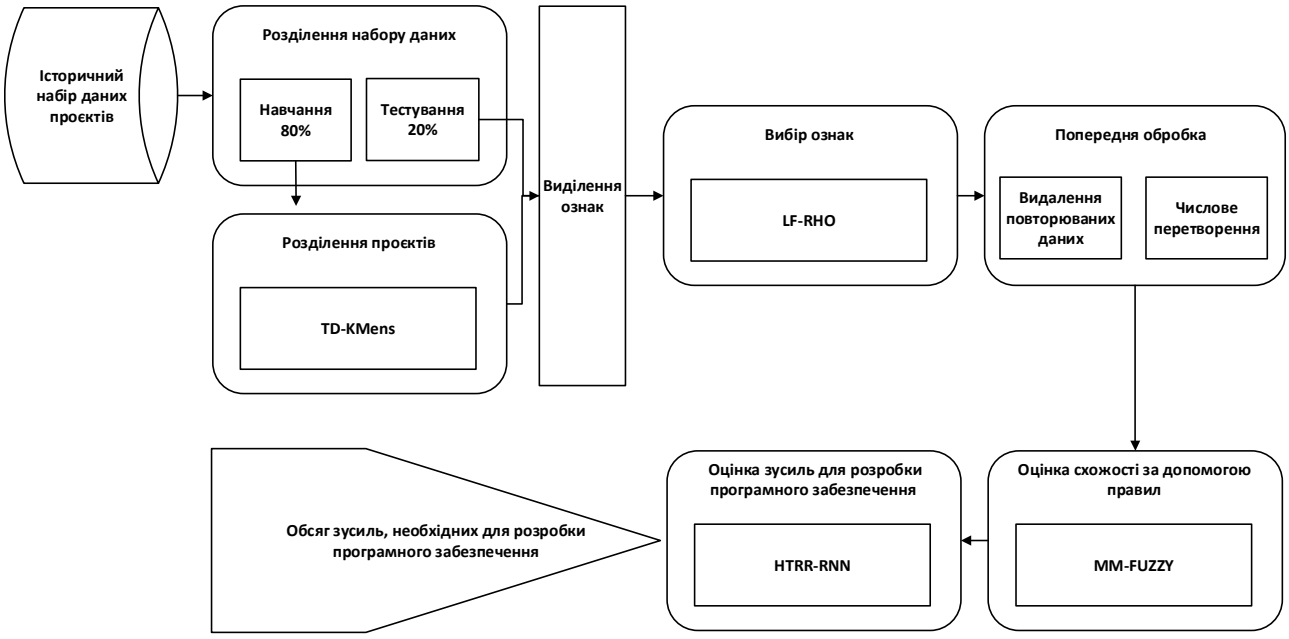

Рисунок 2.1 – Схема роботи запропонованого підходу

2.2 Збір та попередня обробка даних

2.2.1 Джерело вхідних даних

Для цього дослідження використовуються історичні набори даних проєктів, такі як China, COCOMO81 та MAXWELL, які є у відкритому доступі в інтернеті. У цьому дослідженні 20% даних використовується на етапі тестування, а 80% – на етапі навчання.

2.2.2 Розподіл проєктів

Проєкти спочатку розподіляються на основі їхнього розміру. Алгоритм K-means з таксичними відстанями (TD-K Means) використовується як метод групування, за допомогою якого проєкти подібного розміру об'єднуються в один кластер. Зазвичай, у традиційному алгоритмі K-means відстань між точками даних і центроїдами обчислюється за допомогою евклідової відстані (ED). Проте, продуктивність кластеризації може погіршуватися, оскільки ED не підходить для великого обсягу даних. Для заміни ED тут використовується таксична відстань, яка є більш вигідною при роботі з великим обсягом точок даних. В результаті TD-KMeans створює ідеальні кластери з вищою точністю кластеризації.

Кроки алгоритму TD-K Means наведено нижче:

Крок 1. Ініціалізуються центроїди разом з деякими точками даних. У цьому випадку проєкти позначаються як точки даних. Математичний вираз формулюється таким чином:

$$D_i = \{D_1, D_2, D_3, \dots\dots\dots, D_n\}, \tag{2.1}$$

$$C_i = \{C_1, C_2, C_3, \dots\dots\dots, C_n\}, \tag{2.2}$$

де D_i – кількість точок даних;

C_i – набір центроїдів.

Крок 2. Початковий центроїд кластеру обирається випадковим чином. Кластери призначаються точкам даних, відстань яких до центроїда є найменшою. Обчислення відстані виконується за допомогою таксичної відстані, яка позначається як:

$$TD = |D_2 - D_1| + |D_2 - C_1| \, . \tag{2.3}$$

Крок 3. Обчислюється фактичний центроїд кластеру шляхом усереднення всіх точок даних, призначених цьому кластеру. Взаємозв'язок функцій формулюється таким чином:

$$C_1 = \frac{\sum_{i-1}^{n} D_i}{n} \, . \tag{2.4}$$

де C_1 – початковий центроїд;

n – загальна кількість точок даних, призначених кластеру.

Крок 4. Повторюйте кроки 2 і 3 до досягнення збіжності. Групування проєктів ефективно виконується за допомогою алгоритму TD-KMeans, який зображується як:

$$Cs_i = \{Cs_1, Cs_2, Cs_3, \dots \dots \dots, Cs_n\} \, , \tag{2.5}$$

де $\{Cs_1, Cs_2, Cs_3, \dots \dots \dots, Cs_n\}$ – кількість сформованих кластерів.

2.2.3 Видобування ознак

З кластеризованих проєктів витягуються найважливіші та інформативні ознаки, такі як здатність аналітиків і програмістів, досвід у застосуванні, досвід роботи з віртуальними машинами, розмір бази даних, складність процесу, час обробки та інші. Основна мета методу видобування ознак – зменшити кількість ознак для зниження обчислювальної складності. Математична формула для витягнутих ознак F_{Ex} виглядає так:

$$F_{Ex} = \{F_1, F_2, F_3, \dots\dots\dots, F_n\}\ . \quad (2.6)$$

2.2.4 Вибір ознак

З видобутих ознак вибирається найдоцільніша та необхідна кількість ознак. У даній роботі важливі ознаки обираються за допомогою оптимізаційного підходу, що називається оптимізацією з використанням Levy Flight-Rock Hyraxes Swarm Optimization (LF-RHS). Ці вибрані ознаки мають значний вплив на точність оцінки зусиль для розробки програмного забезпечення. Крім того, цей підхід до вибору ознак підвищує точність класифікації, використовуючи мінімум часу та ресурсів.

Псевдокод алгоритму TD-KMeans:

Вхід: Кількість проєктів

Вихід: Розподіл проєктів

Початок

Ініціалізувати кількість проєктів $D_i = \{D_1, D_2, D_3, \dots\dots\dots, D_n\}$

Ініціалізувати центроїди $C_i = \{C_1, C_2, C_3, \dots\dots\dots, C_n\}$

Обчислити відстань між проєктами та центроїдом за допомогою

$$TD = |D_2 - D_1| + |D_2 - C_1|$$

Обчислити середнє значення всіх проєктів, призначених до кластера

$$C_1 = \frac{\sum_{i-1}^{n} D_i}{n}$$

Ітерація продовжується, доки всі проєкти не будуть згруповані в кластери

Кінець

2.2.4.1 Вибір ознак за допомогою LF-RHS

Метаевристичний метод Rock Hyraxes Swarm Optimization (RHSO) використовується для оновлення позиції лідера. Така модифікація значно підвищує продуктивність та точність, а також зменшує проблеми повільної збіжності.

Розглянемо кроки алгоритму LF-RHS.

На початковому етапі відбувається ініціалізація популяції:

$$F = \{F_1, F_2, F_3, \dots \dots \dots, F_n\}. \quad (2.7)$$

Лідер популяції обирається на основі значення придатності (FV). Лідер обирається з урахуванням індивіда з найкращим значенням FV. Для спостереження за рештою групи лідер обирає найкраще місце. Оновлення позиції виконується лідером на основі наступного виразу:

$$L = lf^* y(L_{Pos}, k)\ , \quad (2.8)$$

$$lf = \begin{cases} 1 & y(L_{Pos}, k) < 1 \\ y(L_{Pos}, k) & y(L_{Pos}, k) \geq 1 \end{cases}, \quad (2.9)$$

де lf – розподіл levy flight;

y – попередня позиція лідера;

L_{Pos} – позиція старого лідера;

k – кожне зменшення.

Після оновлення позиції лідера всі учасники або пошукові агенти оновлюють свої позиції залежно від своїх місць. Цей взаємозв'язок представлений так:

$$y(i,j) = (y(i,j) - (Crc^* y(i,j) + L))\ , \quad (2.10)$$

де Crc – круговий рух, він намагається імітувати кругову систему. Це виражається так:

$$v_1 = rand_2 * Cos(ang)\ , \quad (2.11)$$

$$v_2 = rand_2 * Sin(ang)\ , \quad (2.12)$$

$$Crc = \sqrt{(v_1^2 + v_2^2)}, \tag{2.13}$$

де $rand$ – радіус, який знаходиться в межах [0,1];

ang – кут переміщення, який є випадковим числом у діапазоні [0,360].

Кожне покоління оновлюється ang, і це залежить від нижніх і верхніх меж змінних, де Lb і Ub – це нижні та верхні межі генерації випадкових чисел:

$$Dalta = Rd[Lb, Ub] , \tag{2.14}$$

$$ang = ang + Dalta . \tag{2.15}$$

Кут ang може зберігатися в межах вказаного діапазону, приймаючи значення 360 для значень, що перевищують 360, або 0 для значень, що менші за 0.

У режимі дослідження RHSO починає оптимізацію з генерації випадкових рішень і оцінки їхньої ефективності. Найкращі варіанти обираються на основі їхньої «придатності». Коли система наближається до глобального оптимуму, вона переходить до локального режиму, зосереджуючись на найперспективніших напрямках.

Для задач оптимізації найкраще рішення демонструється «лідером». Пошукові агенти знову запускають новий цикл досліджень, після чого переходять до нового етапу уточнення. Позиція лідера оновлюється, а інші агенти слідують за ним. Лідер обирається після оцінки придатності всіх агентів.

Процес триває ітераціями, під час яких система поступово покращує результати. Врешті-решт, найкращий результат, досягнутий системою, вважається рішенням, поки не буде виконано умову завершення.

Арифметичне представлення вибраних оптимальних характеристик подано таким чином:

$$S(F) = \{\lambda_1, \lambda_2, \lambda_3, \dots\dots\dots, \lambda_n\} \ . \quad (2.16)$$

2.2.5 Попередня обробка

Етап попередньої обробки завершується після вибору оптимальних ознак. В цьому етапі відбувається перетворення обраних ознак у передбачуваний та аналітичний формат, ігноруючи непотрібні секції даних. Це підвищує загальну якість ознак, і машина може легко розуміти ці ознаки. Процедура попередньої обробки в цій моделі зосереджується на двох етапах: видалення повторюваних даних та числове перетворення. Математична функція попередньої обробки представлена так:

$$P_R = \alpha_f(S(F)) \ , \quad (2.17)$$

де P_R – вихід функції попередньої обробки;

$S(F)$ – вибрані ознаки;

α_f – функція попередньої обробки.

Вираз для функції попередньої обробки виглядає так:

$$\alpha_f = \{\alpha_{RD}, \alpha_{NC}\}, \quad (2.18)$$

де α_{RD} – функція видалення повторюваних даних;

α_{NC} – функція числового перетворення.

2.2.5.1 Видалення повторюваних даних

На цьому етапі видаляються ознаки, які повторюються більше одного разу. Це дозволяє значно знизити рівень споживання ресурсів та скоротити час обробки. Математично це представлено так:

$$R_{data} = \alpha_{RD}\{P_R\}, \quad (2.19)$$

де R_{data} – результат етапу видалення повторюваних даних;

$\alpha_{RD}\{P_R\}$ – функція видалення даних з попередньо оброблених даних.

2.2.5.2 Числове перетворення

Числове перетворення (або Numeralization) використовується для перетворення символьних значень або рядків у числовий формат. Формула для функції числового перетворення виглядає так:

$$N_u = \alpha_{NC}\{P_R\}, \tag{2.20}$$

де N_u – результат етапу числового перетворення;

$\alpha_{NC}\{P_R\}$ –функція числового перетворення для попередньо оброблених даних

Таким чином, попередньо оброблені ознаки математично позначаються як PF_e.

2.3 Оцінювання подібності проєктів за допомогою нечітких правил

Після попередньої обробки для кожної ознаки створюються певні правила. Для побудови цих правил використовується MM-Fuzzy. Щоб оцінити подібність між проєктами, згенеровані правила порівнюються між собою. Далі проєкти групуються на основі їх подібностей.

Вхідні, оброблювані та вихідні етапи системи нечіткої логіки виконуються стандартним чином. Дані спочатку вводяться як вхідні та потім мапуються для обробки. На етапі обробки виконується операція на основі правил. Ця система обробки ґрунтується на логічних правилах, що визначаються типом операції. Врешті-решт, дефазифікатор виконує процес дефазифікації та виводить результат.

Кроки системи нечіткої логіки описані наступним чином:

Спочатку, використовуючи процес фузифікації, попередньо оброблені ознаки PF_e перетворюються на нечіткі множини:

$$PF_e \xrightarrow{fuzzification} \Delta PF_e, \tag{2.21}$$

де ΔPF_e – перетворена нечітка множина.

Набір правил створюється для відображення вхідних даних на вихідні, коли нечітка множина застосовується до нечіткої інтерференції. Метод Мамдані включається до існуючого алгоритму нечіткої логіки для покращення етапу інтерференції. Цей вираз виглядає так:

$$M_A(x)\&M_B(x) = MIN\{M_A(x), M_B(x)\}, \tag{2.22}$$

де $M_A(x)$ – це функція належності множини A;

$M_B(x)$ – функція належності множини B.

Зазвичай правила формулюються у вигляді «Якщо-Тоді» за допомогою лінгвістичних змінних, таких як «низький», «високий» та «середній». Наприклад, правило F_r може бути сформульоване так:

$$F_r = \begin{cases} \text{ЯКЩО досвід програміста і можливості програміста} = \text{Низький ТОДІ зусилля} = \text{Високий} \\ \text{ЯКЩО досвід програміста і можливості програміста} = \text{Високий ТОДІ зусилля} = \text{Низький} \end{cases} \tag{2.23}$$

Відповідно до функції належності, виконується дефазифікація. Щоб перетворити нечіткі значення на вихідні результати, використовується процес дефазифікації.

На рисунку 2.2 представлена блок-схема MM-Fuzzy.

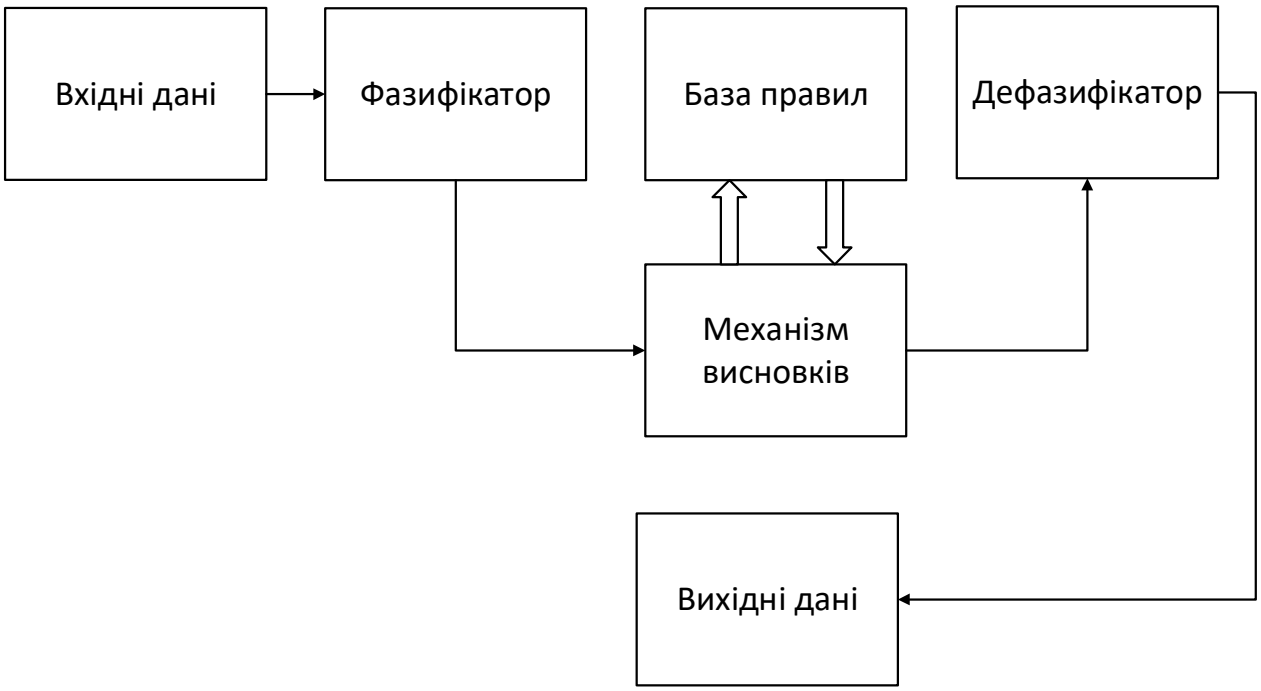

Рисунок 2.2 – Архітектура алгоритму MM-Fuzzy

2.4 Класифікація на основі нейронної мережі

Для ефективного оцінювання зусиль для розробки програмного забезпечення агреговані ознаки подаються як вхідні дані до класифікатора, який називається алгоритм-HTRR-RNN. Для класифікації в традиційних рекурентних нейронних мережах (RNN) використовуються функції активації, такі як сигмоїдна або softmax. Проте, у процесі зворотного поширення ці функції активації мають недолік зникання градієнта, що призводить до повільного навчання та зниженої продуктивності класифікації. Алгоритм RNN інтегрує гіперболічну тангенсну рекурентну радіальну (HTRR) функцію для вирішення цих недоліків. Функція HTRR особливо підходить для глибоких мереж, оскільки вона ефективно обробляє зазначені проблеми. Крім того, вона знижує рівень помилок і дозволяє розпізнавати найкращий результат за мінімальну кількість ітерацій. За допомогою HTRR-RNN отримується кращий результат класифікації у стислий термін.

Вихід попереднього етапу подається як вхід на наступний крок у RNN. Найважливішою особливістю RNN є прихований стан, який запам'ятовує певну інформацію про послідовність. Кожен прихований шар (HL) у традиційних нейронних мережах має власні ваги та зсуви. Наприклад, у HL 1 ваги та зсуви позначаються як (W_1, B_1), для другого HL ваги та зсуви позначаються як (W_2, B_2), і (W_3, B_3) для третього HL. Ці шари не запам'ятовують попередні результати, оскільки кожен із них незалежний від інших. Однак, надаючи однакові ваги та зсуви всім шарам, RNN перетворює незалежні активації в залежні активації, тим самим зменшуючи складність параметрів і запам'ятовуючи кожен попередній вихід, подаючи кожен вихід як вхід до наступного HL.

Процедури HTRR-RNN описані наступним чином.

Крок 1. Алгоритм HTRR-RNN виконується на вхідних даних $X_{IN} = \{X_1, X_2, X_3, \dots, X_t\}$, що включають послідовність прихованих векторів

$$H_{Lyr} = A_{act}[wg_{XH}X_t + wg_{HH}H_{t-1} + \beta_{bais}], \tag{2.24}$$

де wg – матриці ваг (наприклад, вага між вхідним шаром і прихованим шаром wg_{XH},

wg_{HH} – вага прихованого шару;

β_{bais} – вектор зміщень;

A_{act} – гіперболічна тангенсна рекурентна радіальна функція/

Вираз для гіперболічної тангенсної рекурентної радіальної функції виглядає так:

$$A_{act} = \frac{(e^X - e^{-X})(X - wg)^2}{R^2(e^X + e^{-X})}, \tag{2.25}$$

де R – випадкова змінна.

Крок 2. Точність процесу оцінювання зусиль розробки програмного забезпечення (SEE), виконаного моделлю, передбачає вихідний шар (OL). Сигмоїдна функція активації σ_S активує вихідний шар, який визначається так:

$$Y_{out} = \sigma_S[wg_{XY}H_t + \beta_{bais}] , \tag{2.26}$$

$$X = wg_{XY}H_t + \beta_{bais} \tag{2.27}$$

Крок 3. Сигмоїдна функція активації обчислюється за допомогою наступного співвідношення:

$$\sigma_S(X) = \frac{1}{1+\mathcal{E}^{-\alpha}}. \tag{2.28}$$

Крок 4. Потім, обчислюючи різницю між фактичним значенням X_a та прогнозованим значенням $\hat{X}_p$, обчислюється значення втрат (loss):

$$Loss = (X_a - \hat{X}_p)^2. \tag{2.29}$$

Якщо значення втрат або помилки дорівнює нулю ($Loss = 0$), це означає, що процес оцінювання зусиль був виконаний точно за допомогою моделі. У разі, якщо значення втрат $Loss \neq 0$, виконується зворотне поширення шляхом оновлення значень ваг.

Зрештою, кількість зусиль, необхідних для розробки програмного забезпечення, ефективно передбачається класифікатором HTRR-RNN.

Висновки до розділу 2

1. Запропоновано метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж, який поєднує в собі здатність нечіткої логіки обробляти невизначеність з потужністю нейронних

мереж у прогнозуванні складних залежностей. Це забезпечує більш точне прогнозування зусиль, що необхідні для реалізації програмних проєктів, оптимізацію використання ресурсів та зменшення ризиків, пов’язаних зі змінами у проєкті. Завдяки інтеграції цих методів, запропонований підхід дозволяє адаптуватися до динамічних умов, що особливо важливо для великих і складних проєктів, де важливо швидко реагувати на зміну вимог та умов.

2. Описано процес збору та підготовки даних для оцінювання зусиль у програмних проєктах, що включає використання історичних наборів даних, таких як China, COCOMO81 та MAXWELL, розподіл проєктів за допомогою методу TD-K Means, видобування та вибір інформативних ознак. Попередня обробка, яка включає видалення повторюваних даних та числове перетворення, покращує якість ознак та спрощує їх подальшу обробку моделлю.

3. Оцінювання подібності проєктів з використанням нечітких правил забезпечує більш точний розподіл проєктів на основі їхніх характеристик. Нечітка логіка, реалізована за допомогою алгоритму MM-Fuzzy, дозволяє враховувати невизначеність і розмитість даних, що особливо важливо для складних програмних проєктів. Використання правил у вигляді «Якщо-Тоді» та процесу дефазифікації дозволяє ефективно визначати рівень подібності між проєктами та створювати групи з подібними характеристиками.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МЕТОДУ УПРАВЛІННЯ ПРОЄКТАМИ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ ТА НЕЙРОННИХ МЕРЕЖ

3.1 Дослідження продуктивності запропонованих рішень

3.1.1 Аналіз продуктивності TD-KMeans

У цьому розділі проілюстровано експериментальний аналіз запропонованого методу. Крім того, для демонстрації ефективності запропонованого підходу виконано аналіз продуктивності та порівняльний аналіз. Дані для запропонованої системи зібрані з трьох різних наборів даних, а саме China, COCOMO81 та MAXWELL. Для реалізації використовується Python.

Для підтвердження ефективності запропонованої моделі TD-KMeans виконано аналіз продуктивності щодо часу поділу проєктів, а також порівняно результати з різними існуючими техніками, такими як CLARA (Clustering Large Applications), K-Means, Fuzzy C-Means (FCM) та Partition Around Medoids (PAM).

Графічне представлення часу поділу проєктів за допомогою TD-KMeans та існуючих моделей, таких як K-Means, PAM, CLARA та FCM, наведено на рисунку 3.1.

Модель TD-KMeans досягла кращих результатів щодо часу поділу проєктів, а складність зменшилася. В запропонованій системі ефективна таксична відстань інтегрована з алгоритмом кластеризації K-means для підвищення ефективності кластеризації. Замість загальної евклідової відстані в алгоритмі K-means використовується таксична відстань. Модифікована таксична відстань краще підходить для роботи з великим обсягом даних, що підвищує продуктивність кластеризації та результати. Для завершення загального процесу поділу проєктів існуючій техніці потрібно в середньому 82019 мс, тоді як запропонованій схемі потрібно всього 1154 мс для завершення. Покращена продуктивність кластеризації, досягнута запропонованим алгоритмом, свідчить про те, що проєкти ефективно розподіляються за розміром для подальшого оцінювання зусиль розробки програмного забезпечення.

Рисунок 3.1 – Порівняльний аналіз TD-KMeans щодо часу поділу проєктів

Оскільки існуючі методи потребують більше часу, ніж запропоновані, модель TD-KMeans досягає обнадійливого результату з меншим часом виконання.

3.1.2 Аналіз продуктивності алгоритму LF-RHS

Для демонстрації його ефективності щодо «фітнес проти ітерацій» виконано аналіз продуктивності запропонованого алгоритму вибору ознак під назвою LF-RHS та порівняно з різними існуючими рішеннями, такими як Rock Hyraxes Swarm (RHS), Sequential Minimal Optimization (SMO), Monarch Butterfly Optimization (MBO) та Crow Search Algorithm (CSA).

Час вибору ознак LF-RHS разом з іншими існуючими методами порівнюється на рисунку 3.2.

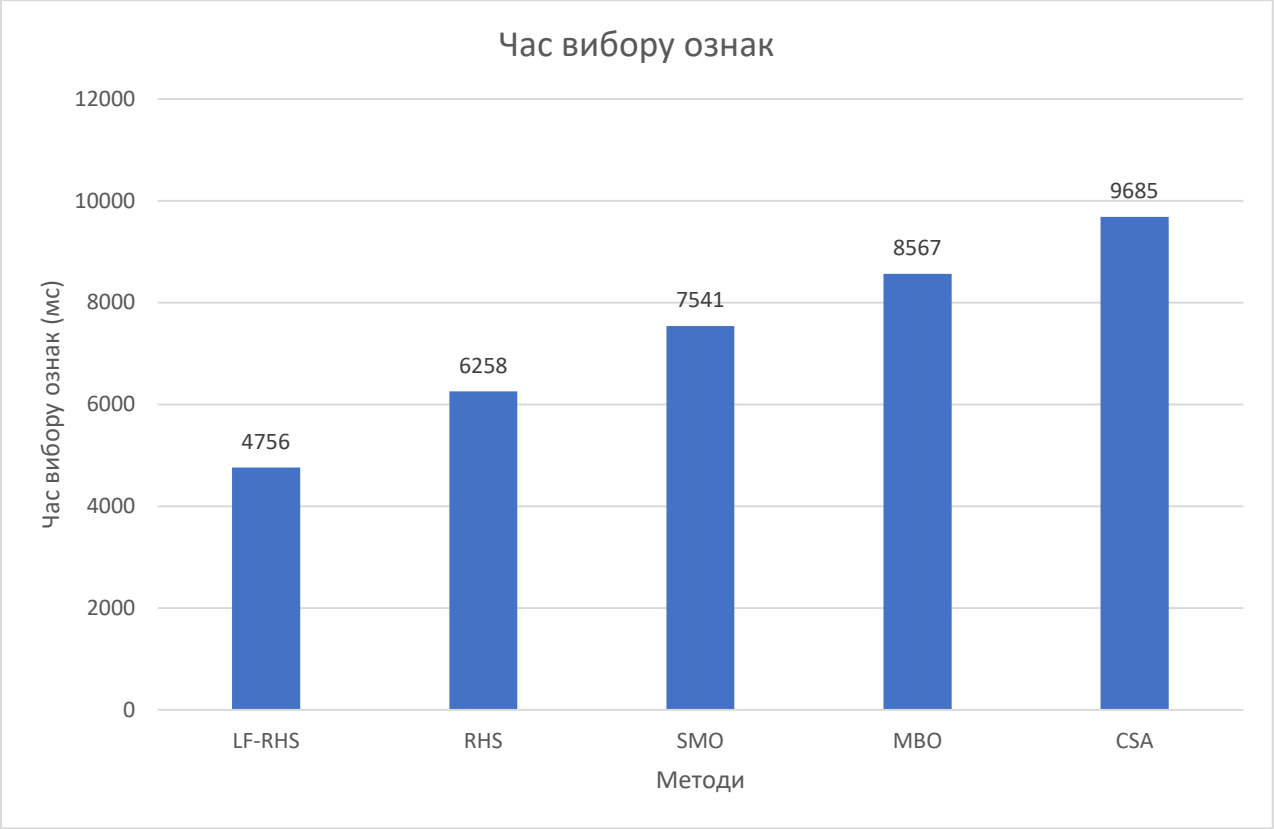

Рисунок 3.2 – Порівняльний аналіз LF-RHS щодо часу вибору ознак

Час, витрачений алгоритмом оптимізації для вибору оптимальних ознак, називається часом вибору ознак. У запропонованому LF-RHS використовується домінуючий підхід Levy Flight-Rock для оновлення позиції лідера. Випадковий процес оновлення позиції лідера обмежує результати оптимізації та продуктивність. Тому метод LF використовується для покращення швидкості збіжності оптимізації. У той час як час вибору ознак існуючими техніками, такими як RHS, SMO, MBO та CSA, становить 6258 мс, 7541 мс, 8567 мс і 9685 мс відповідно, запропонована техніка LF-RHS потребує лише 4756 мс для вибору ідеальних ознак. З графічного аналізу видно, що LF-RHS вибирає оптимальні ознаки за обмежений час. Оскільки проєкти розподіляються перед отриманням ознак, це сприяє швидкому вибору ознак за мінімальний час за допомогою запропонованого підходу. Таким чином, запропонований метод перевершує інші існуючі методи та залишається більш надійним.

Графічний аналіз на рисунку 3.3 показує, що алгоритм LF-RHS досяг кращих результатів значення придатності для відповідної ітерації порівняно з існуючими алгоритмами CSA, MBO, SMO та RHS під час вибору ознак.

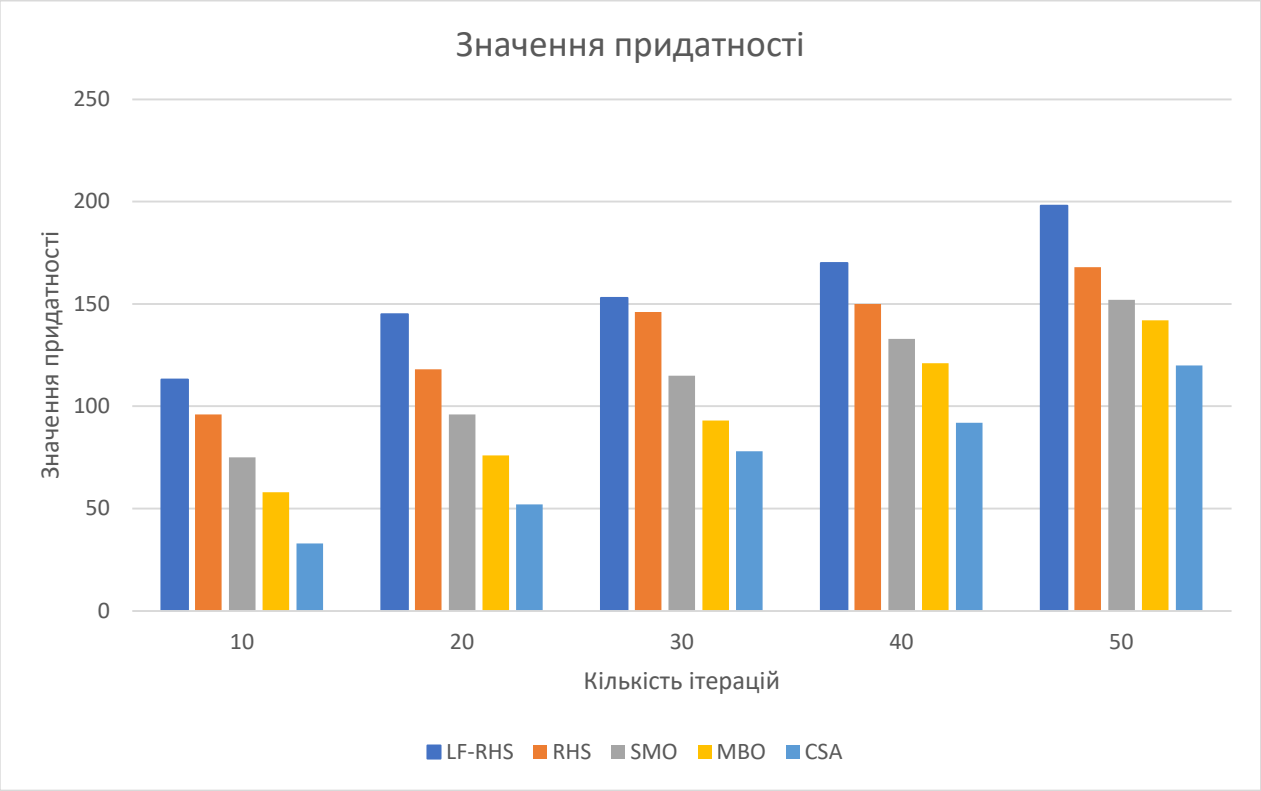

Рисунок 3.3 – Порівняльний аналіз LF-RHS відносно значення придатності та ітерацій

Запропонований LF-RHS досяг хорошої продуктивності під час процесу оптимізації завдяки схемі Levy Flight-Rock. За невеликої кількості ітерацій кращі значення придатності можуть зменшити складність часу, і також досягається висока точність без подальших ітерацій. Запропонована система досягає 113 значень придатності на 10-й ітерації та 198 значень фітнесу на 50-й ітерації, тоді як існуючі методи в середньому досягають 66 значень придатності на 10-й ітерації та 146 значень придатності на 50-й ітерації. Оскільки проблема виникає через випадкове оновлення позиції та повільну швидкість збіжності оптимізації рою, продуктивність придатності покращується за допомогою запропонованого алгоритму. З результатів порівняння видно, що найкращі ознаки досягаються LF-RHS за обмежену кількість ітерацій у порівнянні з існуючими алгоритмами.

3.1.3 Аналіз продуктивності моделі HTRR-RNN

Для підтвердження ефективності моделі HTRR-RNN виконано аналіз продуктивності за різними метриками, такими як точність, чутливість, специфічність, точність (precision), повнота (recall), F-міра та час навчання, у порівнянні з існуючими методами, такими як Deep Belief Network (DBN), RNN, Deep Neural Network (DNN) та Convolutional Neural Network (CNN).

Щодо точності, чутливості та специфічності, запропонований HTRR-RNN порівнюється з існуючими методами, такими як DBN, RNN, DNN та CNN, що зображено на рисунку 3.4.

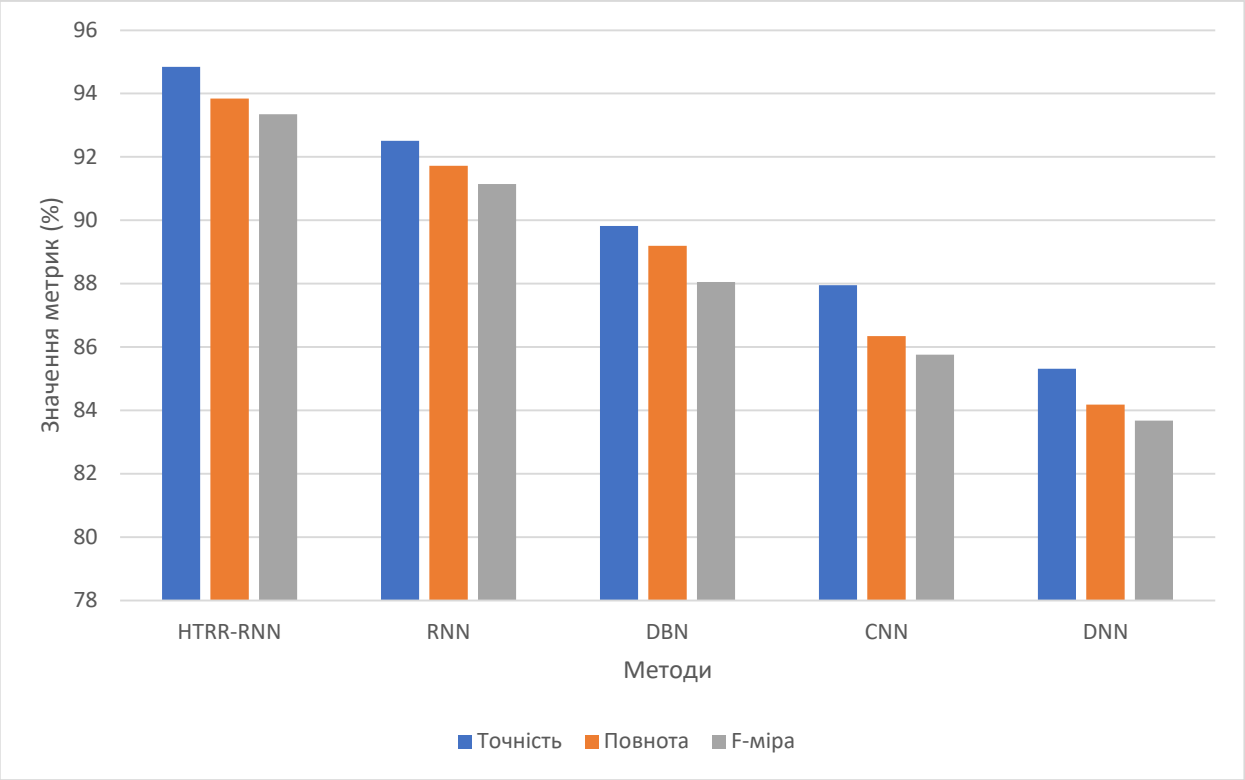

Рисунок 3.4 – Графічне представлення HTRR-RNN щодо точності, чутливості та специфічності

Запропонований HTRR-RNN впроваджує гіперболічну тангенсну рекурентну радіальну функцію (HTRR) замість базової функції активації. Запропонована функція HTRR усуває проблему зникання градієнта, характерну для загальних функцій активації, за допомогою HTRR-функції та покращує процес навчання, а також точність класифікатора. Запропонований HTRR-RNN досягає точності 94,84 %, чутливості 93,84 % та специфічності 93,35 % за

результатами порівняльного аналізу з існуючими методами, такими як RNN, DBN, CNN та DNN, які в середньому досягають точності 88,73 %, чутливості 87,81 % та специфічності 88,57 %. Запропонований HTRR-RNN демонструє кращі показники у порівнянні з існуючими методами. Кількість зусиль, необхідних для розробки програмного забезпечення, обчислюється за допомогою запропонованого методу.

Точність (precision), повнота (recall) та F-міра запропонованого HTRR-RNN порівнюються з існуючими роботами, такими як RNN, DBN, CNN та DNN,.

Вищий рівень точності, повноти та F-міри визначає якість моделі. Запропонований HTRR-RNN показав гарні результати у прогнозуванні зусиль у середовищі розробки програмного забезпечення завдяки ефективній функції HTRR. Згідно з цим, запропонована техніка досягає 94,74 % точності, 93,68 % повноти та 94,84 % F-міри, тоді як існуючі методи мають точність у діапазоні 85,87 % - 92,48 %, повноту в діапазоні 84,32 % - 91,75 % та F-міру в діапазоні 85,62 % - 91,86 %, що є нижчими у порівнянні з запропонованою роботою. Оскільки подібність між ознаками проєктів аналізується за допомогою нечітких правил перед оцінюванням зусиль, запропонована мережа досягає кращої продуктивності порівняно з існуючими нейронними мережами. Отже, різні складності зменшуються, а процес оцінювання помилок програмного забезпечення покращується за допомогою HTRR-RNN.

Щодо часу навчання, оцінка продуктивності HTRR-RNN у порівнянні з іншими існуючими методами, такими як RNN, CNN, DBN та DNN, наведена в таблиці 3.1.

Запропонований класифікатор HTRR-RNN ефективно покращує процес навчання, що призводить до низької часової складності. За результатами аналізу існуючих технік, час навчання є меншим для запропонованої моделі. Існуючі методи, такі як RNN, DBN, CNN та DNN, потребують 10954 мс, 12658 мс, 14623 мс та 16357 мс відповідно для навчання даних, тоді як запропонований HTRR-RNN потребує лише 8754 мс для навчання даних.

Таблиця 3.1 – Аналіз продуктивності запропонованого HTRR-RNN на основі часу навчання

Метод	Час навчання (мс)
HTRR-RNN	8754
RNN	10954
DBN	12658
CNN	14623
DNN	16357

Загальний час виконання моделі значно залежить від тривалості навчання, що, своєю чергою, впливає на продуктивність моделі. У цьому методі складності зменшуються за рахунок завершення процесу навчання з меншими витратами часу та обчислювальними ресурсами.

Продуктивність моделі HTRR-RNN аналізується шляхом порівняння її з новими нейронними мережами, такими як RNN, Gated Recurrent Unit (GRU), Bidirectional Long Short Term Memory (BiLSTM) та Long Short Term Memory (LSTM). З таблиці 3.2 видно, що запропонована модель досягає точності 94,84 % і повноти 94,74 %. Тим часом складні нові мережі, такі як RNN, досягли точності 92,63 %, GRU досягла повноти 91,27 %, LSTM досягла точності 87,45 %, а BiLSTM досягла повноти 90,12 %. Ці нові мережі досягли нижчої продуктивності, ніж запропонована модель. Це пояснюється тим, що подібність між проєктами аналізується перед оцінюванням, а здатність до навчання запропонованої мережі покращується за допомогою HTRR-функції активації. Таким чином, запропонована модель досягає вищої продуктивності в оцінці зусиль (SEE) порівняно з іншими сучасними нейронними мережами.

Таблиця 3.2 – Аналіз HTRR-RNN у порівнянні зі складними нейронними мережами

Підхід	Точність (%)	Точність (Precision, %)	Повнота (Recall, %)
HTRR-RNN	94.85	94.85	94.74
RNN	92.63	91.87	92.49
GRU	91.56	90.46	91.27
BiLSTM	90.32	89.65	90.12
LSTM	88.61	87.45	88.35

3.2 Порівняльний аналіз запропонованого рішення з відомими

Продуктивність оцінювання зусиль за допомогою запропонованої роботи підтверджується порівнянням її з деякими наявними роботами за такими метриками, як середня абсолютна помилка (MAE) та середньоквадратична помилка (RMSE).

У таблиці 3.3 наведено порівняння запропонованої роботи з деякими наявними методами. Помітно, що запропонована робота досягає MAE на рівні 0.18 та RMSE на рівні 0.38. Існуючі роботи, такі як [16] та [26] досягли MAE на рівні 0.43 та 0.36 відповідно. Далі, значення RMSE 0.26 та 0.62 досягаються у [10] та [16] відповідно. Існуючі моделі мали вищі значення помилок у порівнянні з запропонованою моделлю. Оскільки у запропонованому підході оцінюється подібність між проєктами, зусилля на розробку програмного забезпечення оцінюються з меншою кількістю помилок, ніж у пов’язаних роботах. Таким чином, запропонована робота ефективно справляється з обмеженою продуктивністю наявних методів.

Таблиця 3.3 – Порівняльний аналіз

Джерело	Використаний метод	MAE	RMSE
Запропонована модель	HTRR-RNN	0.18	0.38
(Jadhav та ін., 2023) [10]	Omni Ensemble Learning	0.26	0.47
(Nhung та ін., 2022) [16]	Optimization of Correction Factors based Regression model	0.43	0.62
(Hameed та ін., 2023) [7]	Grey Wolf Optimization	3.51	4.53
(van Hai та ін., 2022) [26]	Function Point Analysis	0.36	0.54

Запропонована модель демонструє більш низькі значення MAE та RMSE порівняно з іншими роботами, що свідчить про її вищу точність в оцінці зусиль для розробки програмного забезпечення.

Висновки до розділу 3

1. Запропонована модель TD-KMeans суттєво покращує продуктивність кластеризації, забезпечуючи значно менший час поділу проєктів у порівнянні з традиційними методами, такими як K-Means, CLARA та FCM. Аналогічно, алгоритм LF-RHS демонструє більш швидкий і точний вибір ознак завдяки інтеграції схеми Levy Flight, що дозволяє досягти оптимальних результатів за значно меншу кількість ітерацій.

2. Запропонована модель HTRR-RNN показала значне покращення в метриках точності, чутливості, специфічності, а також точності та F-міри у порівнянні з існуючими нейронними мережами, такими як RNN, CNN, DNN та DBN. Висока ефективність моделі досягнута завдяки використанню функції

активації HTRR, яка усуває проблему зникання градієнта та підвищує якість класифікації.

3. Усі запропоновані рішення демонструють кращі результати у порівнянні з відомими підходами за ключовими метриками, такими як MAE та RMSE. Зокрема, модель HTRR-RNN досягає мінімальних значень помилок, що підтверджує її перевагу в оцінюванні зусиль на розробку програмного забезпечення. Це свідчить про здатність запропонованих рішень забезпечувати ефективне управління проєктами за рахунок зниження обчислювальних витрат та покращення точності оцінювання.

ВИСНОВКИ

У цій роботі запропоновано підхід до оцінювання зусиль для розробки програмного забезпечення, який перевершує існуючі методи за точністю, чутливістю та специфічністю. Запропонована методологія ефективно враховує диференціацію функціональності програмного забезпечення та забезпечує швидкий і точний розподіл проєктів та вибір ознак навіть у динамічних умовах.

Основні наукові результати:

1. Точне оцінювання зусиль є ключовою для ефективного управління програмними проєктами, оскільки вона дозволяє раціонально розподілити ресурси, спланувати бюджет та встановити реалістичні терміни. Сучасні тенденції до збільшення складності проєктів і швидкості розробки підвищують потребу в точних прогнозах, оскільки неточні оцінки можуть призводити до перевитрати ресурсів, затримок і фінансових втрат. Таким чином, розробка точних методів для оцінювання зусиль є важливою задачею для галузі.

2. Класичні статистичні методи, такі як лінійна та множинна регресія, забезпечують основу для оцінювання зусиль на основі історичних даних і є зрозумілими у використанні. Проте ці підходи мають обмежену ефективність для великих або складних проєктів, де наявні нелінійні залежності між параметрами. У сучасних умовах вони поступаються місцем більш гнучким і точним методам машинного навчання, здатним краще враховувати різні фактори.

3. Методи машинного навчання, такі як ансамблеві моделі та нейронні мережі, стали важливим інструментом для точнішого оцінювання зусиль завдяки їх здатності моделювати складні зв’язки між даними. Вони дозволяють покращити точність прогнозів, але вимагають значних обчислювальних ресурсів та великих наборів даних для навчання. Подальший розвиток цих методів, у тому числі їх інтеграція з оптимізаційними алгоритмами, дозволяє створювати більш точні та адаптивні моделі для оцінювання зусиль у динамічних умовах програмних проєктів.

4. Запропоновано метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж, який поєднує в собі здатність нечіткої логіки обробляти невизначеність з потужністю нейронних мереж у прогнозуванні складних залежностей. Це забезпечує більш точне прогнозування зусиль, що необхідні для реалізації програмних проєктів, оптимізацію використання ресурсів та зменшення ризиків, пов'язаних зі змінами у проєкті. Завдяки інтеграції цих методів, запропонований підхід дозволяє адаптуватися до динамічних умов, що особливо важливо для великих і складних проєктів, де важливо швидко реагувати на зміну вимог та умов.

5. Описано процес збору та підготовки даних для оцінювання зусиль у програмних проєктах, що включає використання історичних наборів даних, таких як China, COCOMO81 та MAXWELL, розподіл проєктів за допомогою методу TD-K Means, видобування та вибір інформативних ознак. Попередня обробка, яка включає видалення повторюваних даних та числове перетворення, покращує якість ознак та спрощує їх подальшу обробку моделлю.

6. Оцінювання подібності проєктів з використанням нечітких правил забезпечує більш точний розподіл проєктів на основі їхніх характеристик. Нечітка логіка, реалізована за допомогою алгоритму MM-Fuzzy, дозволяє враховувати невизначеність і розмитість даних, що особливо важливо для складних програмних проєктів. Використання правил у вигляді «Якщо-Тоді» та процесу дефазифікації дозволяє ефективно визначати рівень подібності між проєктами та створювати групи з подібними характеристиками.

8. Запропонована модель TD-KMeans суттєво покращує продуктивність кластеризації, забезпечуючи значно менший час поділу проєктів у порівнянні з традиційними методами, такими як K-Means, CLARA та FCM. Аналогічно, алгоритм LF-RHS демонструє більш швидкий і точний вибір ознак завдяки інтеграції схеми Levy Flight, що дозволяє досягти оптимальних результатів за значно меншу кількість ітерацій.

9. Запропонована модель HTRR-RNN показала значне покращення в метриках точності, чутливості, специфічності, а також точності та F-міри у

порівнянні з існуючими нейронними мережами, такими як RNN, CNN, DNN та DBN. Висока ефективність моделі досягнута завдяки використанню функції активації HTRR, яка усуває проблему зникання градієнта та підвищує якість класифікації.

10. Усі запропоновані рішення демонструють кращі результати у порівнянні з відомими підходами за ключовими метриками, такими як MAE та RMSE. Зокрема, модель HTRR-RNN досягає мінімальних значень помилок, що підтверджує її перевагу в оцінюванні зусиль на розробку програмного забезпечення. Це свідчить про здатність запропонованих рішень забезпечувати ефективне управління проєктами за рахунок зниження обчислювальних витрат та покращення точності оцінювання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alsheikh N. M., Munassar N. M. Improving software effort estimation models using grey wolf optimization algorithm. *IEEE Access*. 2023. Vol. 11. P. 143549–143579.

2. Benala T. R., Mall R. DABE: Differential evolution in analogy-based software development effort estimation. *Swarm and Evolutionary Computation*. 2018. Vol. 38. P. 158–172.

3. de Campos Souza P. V., Guimaraes A. J., Araujo V. S., Rezende T. S., Araujo V. J. S. Incremental regularized data density-based clustering neural networks to aid in the construction of effort forecasting systems in software development. *Applied Intelligence*. 2019. Vol. 49(13). P.1–14.

4. De Carvalho H. D. P., Fagundes R., Santos W. Extreme learning machine applied to software development effort estimation. *IEEE Access*. 2021. Vol. 9. P. 92676–92687.

5. Ezghari S., Zahi A. Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method. *Applied Soft Computing*. 2018. Vol. 67. P. 540–557.

6. Fadhil A. A., Alsarraj R. G., Altaie A. M. Software cost estimation based on dolphin algorithm. *IEEE Access*. 2020. Vol. 8. P. 75279–75287.

7. Hameed S., Elsheikh Y., Azzeh M. An optimized case-based software project effort estimation using genetic algorithm. *Information and Software Technology*. 2023. Vol. 153. P. 1–22.

8. Hastings T. E., Sajeev A. S. M. A vector-based approach to software size measurement and effort estimation. *IEEE Transactions on Software Engineering*. 2001. Vol. 27(4). P. 337–350.

9. Huang S. J., Chiu N. H., Chen L. W. Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research*. 2008. Vol. 188(3). P. 898–909.

10. Jadhav A., Shandilya S. K., Izonin I., Gregus M. Effective software effort

estimation leveraging machine learning for digital transformation. *IEEE Access*. 2023. Vol. 11. P. 83523–83536.

11. Kaushik A., Singal N. A hybrid model of wavelet neural network and metaheuristic algorithm for software development effort estimation. *International Journal of Information Technology*. 2022. Vol. 14(3). P.1689–1698.

12. Khan M. S., Jabeen F., Ghouzali S., Rehman Z., Naz S., Abdul W. Metaheuristic algorithms in optimizing deep neural network model for software effort estimation. *IEEE Access*. 2021. Vol. 9. P. 60309–60327.

13. Kocaguneli E., Menzies T., Keung J., Cok D., Madachy R. Active learning and effort estimation: Finding the essential content of software effort estimation data. *IEEE Transactions on Software Engineering*. 2013. Vol. 39(8). P. 1040–1053.

14. Mittas N., Angelis L. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on Software Engineering*. 2013. Vol. 39(4). P. 537–551.

15. Nassif A. B., Azzeh M., Idri A., Abran A. Software development effort estimation using regression fuzzy models. *Computational Intelligence and Neuroscience*. 2019. P. 1–18.

16. Nhung H. L. T. K., Van Hai V., Silhavy R., Prokopova Z., Silhavy P. Parametric software effort estimation based on optimizing correction factors and multiple linear regression. *IEEE Access*. 2022. Vol. 10. P. 2963–2986.

17. Pendharkar P. C., Subramanian G. H., Rodger J. A. A probabilistic model for predicting software development effort. *IEEE Transactions on Software Engineering*. 2005. Vol. 31(7). P. 615–624.

18. Phannachitta P. On an optimal analogy-based software effort estimation. *Information and Software Technology*. 2020. Vol. 125. P. 1–11.

19. Pillai K., Nair V. S. A model for software development effort and cost estimation. *IEEE Transactions on Software Engineering*. 1997. Vol. 23(8). P. 485–497.

20. Rankovic N., Rankovic D., Ivanovic M., Lazic L. A new approach to software effort estimation using different artificial neural network architectures and Taguchi orthogonal arrays. *IEEE Access*. 2021. Vol. 9. P. 26926–26936.

21. Rao K. E., Rao G. A. Ensemble learning with recursive feature elimination integrated software effort estimation: A novel approach. *Evolutionary Intelligence*. 2021. Vol. 14(1). P.151–162.

22. Sehra S. K., Brar Y. S., Kaur N., Sehra S. S. Software effort estimation using FAHP and weighted kernel LSSVM machine. *Soft Computing*. 2019. Vol. 23(4). P. 10881–10900.

23. Shah M. A., Jawawi D. N. A., Isa M. A., Younas M., Abdelmaboud A., Sholichin F. Ensembling artificial bee colony with analogy-based estimation to improve software development effort prediction. *IEEE Access*. 2020. Vol. 8. P. 58402–58415.

24. Shepperd M., Schofield C. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*. 1997. Vol. 23(11). P. 736–743.

25. Singal P., Kumari A. C., Sharma P. Estimation of software development effort: A differential evolution approach. *Procedia Computer Science*. 2020. Vol. 167. P. 2643–2652.

26. van Hai V., Nhung H. L. T. K., Prokopova Z., Silhavy R., Silhavy P. Toward improving the efficiency of software development effort estimation via clustering analysis. *IEEE Access*. 2022. Vol. 10. P. 83249–83264.

27. Velarde H., Santiesteban C., Garcia A., Casillas J. Analyzing the effect of variables in the software development effort estimation. *IEEE Latin America Transactions*. 2016. Vol. 14(8). P. 3797–3803.

28. Zare F., Zare H. K., Fallahnezhad M. S. Software effort estimation based on the optimal Bayesian belief network. *Applied Soft Computing*. 2016. Vol. 49. P. 968–980.

29. Бедратюк Г. Метод машинного навчання в управлінні програмними проектами. *Міжнародний науково-технічний журнал «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2024. №4. С.23-30.

30. Василенко В.М., Вакалюк Т.А. Штучний інтелект в управлінні проєктами: аналіз сучасних досліджень та перспектив розвитку. *Інформатика, обчислювальна техніка та автоматизація*. 2024. Том 35 (74) №4. С.60–67.

31. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Турченко І.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Управління проєктами» спеціальності 122 «Комп'ютерні науки» за другим (магістерським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 32 с.

32. Мись А. Р., Рипіч Б. В., Шевчук В. М. Сучасні підходи до управління проєктами: машинне навчання та блокчейн-технології. *Світ наукових досліджень* : матеріали міжнародної мультидисциплінарної наукової інтернет-конференції (випуск 35), м. Тернопіль, Україна, м. Ополе, Польща, 20-21 листопада 2024 р. Тернопіль, 2024. С. 104–106.

33. Островерхов В.М., Біловус Л.І., Возьний К.З., Луцишин О.О., Монастирський Г.Л., Надвиничний С.А., Питель С.В., Шандрук С.К. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

34. Рипіч Б. В. Управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення* : матеріали міжнародної науково-практичної інтернет-конференції (випуск 94), м. Ополе, Польща, 11-12 грудня 2024 р. URL: http://www.konferenciaonline.org.ua/ua/articles/year-4/rozdil-40/pidrozdil-121/pidrozdil2-0/.

35. Сікора В.О., Дорошко О.А., Каляєв Ю.О. Використання методів машинного навчання для підвищення ефективності управління проєктами. *Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення* : матеріали міжнародної наукової інтернет-конференції, (м. Тернопіль, Україна, м. Ополе, Польща, 9-10 листопада 2023 р.). Вип. 82. С. 79-81.

Додаток А

Копії публікацій

УДК 001 (063)

Світ наукових досліджень. Випуск 35: матеріали Міжнародної мультидисциплінарної наукової інтернет-конференції (м. Тернопіль, Україна, м. Ополе, Польща, 20-21 листопада 2024 р.) / за ред. : О. Патряк та ін. ГО "Наукова спільнота", WSZIA w Opolu. Тернопіль: ФО- П Шпак В.Б. 2024. 278 с.

Збірник наукових публікацій укладено за матеріалами доповідей наукової мультидисциплінарної інтернет-конференції «Світ наукових досліджень. Випуск 35», які оприлюднені на інтернет-сторінці www.economy-confer.com.ua

Оргкомітет

Адреса оргкомітету:
46005, Україна, м. Тернопіль, а/с 797
тел. +380977547363 e-mail: economy-confer@ukr.net

ISSN 2786-6823 (print)

Література:

1. Golovko V., Kroshchanka A., Mikhno E., Komar M., Sachenko A. Deep convolutional neural network for detection of solar panels. Lecture Notes on Data Engineering and Communications Technologies. 2020. 48. 371-389.
2. Zotov V. B., Demin S. S., Glazkova I. Y. Features of material flow accounting for the efficient supply chain management. *International Journal of Supply Chain Management*, 2019. 8.
3. Kalinov I., Petrovsky A., Ilin V., Pristanskiy E., Kurenkov M., Ramzhaev V., et al. WareVision: CNN barcode detection-based UAV trajectory optimization for autonomous warehouse stocktaking. *IEEE Robotics and Automation Letters*, 2020, 5. http://dx.doi.org/10.1109/LRA.2020.3010733.
4. Molling G., Klein, A. Z. Value proposition of IoT-based products and services: A framework proposal. *Electronic Markets*, 2022, 32. http://dx.doi.org/10.1007/s12525-022-00548-w.

СУЧАСНІ ПІДХОДИ ДО УПРАВЛІННЯ ПРОЄКТАМИ: МАШИННЕ НАВЧАННЯ ТА БЛОКЧЕЙН-ТЕХНОЛОГІЇ

Мись Андрій Русланович
магістрант, Західноукраїнський національний університет, м. Тернопіль

Рипіч Богдан Віталійович
магістрант, Західноукраїнський національний університет, м. Тернопіль

Шевчук Вадим Михайлович
магістрант, Західноукраїнський національний університет, м. Тернопіль

Інтернет-адреса публікації на сайті:
https://www.economy-confer.com.ua/full-article/5879/

В умовах зростаючої складності програмних проєктів і невизначеності у плануванні дедалі більшої популярності набувають інтелектуальні технології для управління проєктами. Традиційні методи, хоча й забезпечують певний рівень ефективності, часто не здатні враховувати всі взаємозв'язки між параметрами проєкту, що призводить до неточних оцінок і ризиків перевитрат.

Проблема оцінки зусиль для розробки програмного забезпечення залишається відкритою через наявність складних взаємозв’язків між численними параметрами проєкту, що часто призводить до неточних прогнозів [1]. Сучасні методи оцінки стикаються з труднощами при обробці непослідовних, неповних, розріджених та недостатньо задокументованих наборів даних, що є однією

з основних причин неточності результатів [2]. Існуючі моделі не можуть ефективно обробляти нелінійні зв'язки між характеристиками проєктів та обсягом зусиль, що знижує їх ефективність і надійність [3].

Глибокі нейронні мережі дозволили покращити точність оцінки, проте їх використання обмежується через недоліки, такі як тривалий час навчання, перенавчання та недонавчання [4].

Недоліки існуючих підходів оцінки зусиль розробки програмного забезпечення можна сформулювати наступним чином:

- відсутність достатньої уваги до диференціації функціональності програмного забезпечення, що призводить до неточних оцінок зусиль для програмних проєктів;
- відсутність групування проєктів за специфікаціями, що ускладнює обробку неточних та неякісних даних;
- невідповідний вибір ознак, що призводить до навчання моделей на непідходящих даних і, відповідно, до неточних прогнозів;
- використання необроблених даних з історичних проєктів без попередньої обробки, що обмежує точність класифікації.

З огляду на вищезазначені недоліки, актуальність полягає в розробці підходу для оцінки зусиль розробки програмного забезпечення, який дозволить підвищити точність прогнозування, зменшити час навчання та забезпечити адаптацію моделі до характеристик проєктів. Для вирішення цієї задачі необхідно дослідити методи класифікації та обробки даних, що враховують специфіку програмних проєктів, нелінійні зв'язки між параметрами, а також забезпечують попередню обробку даних для підвищення якості класифікації.

Останніми роками розробники програмного забезпечення зосереджують свою увагу на використанні конкретних методологій або моделей розробки відповідно до вимог програмного продукту. Після вивчення наукових праць та обговорення з експертами галузі було обрано чотири моделі, які будуть розглядатися. Це були водоспадна модель, інкрементна модель, еволюційна модель та гнучкі методології [5].

Щоб відповідати останнім нововведенням, таким як граничні обчислення, інтернет речей та машинне навчання, було б надзвичайно корисно, якби існувала система, яка могла б передбачати відповідну модель розробки програмного забезпечення залежно від вимог до програмного продукту. Оцінювання якості програмного забезпечення є важливим, але також складним процесом. З цієї причини моделі прогнозування дефектів програмного забезпечення отримали широке застосування. З іншого боку, визначення відповідної моделі та вибір найбільш ефективної з наявних моделей залежать від факторів продуктивності [6]. Отже, задача прогнозування ризиків у проєктах програмного забезпечення є актуальною.

Перший крок до успіху проєкту – це створення надійного та точного графіка. Один із методів складання графіків – це метод критичного шляху (Critical path method, CPM), який обчислює ранні та пізні дати початку і завершення робіт, аналізуючи шляхи проєктної мережі вперед і назад. Після цього до кожної діяльності додаються необхідні ресурси. Власники завдань також додають резерви часу або страхові запаси для кожної діяльності, щоб подолати невизначеності [7, 8].

Інтеграція технології блокчейн в управління проєктами може покращити можливості планування та контролю проєктів. Блокчейн – це розподілена база даних, яка ділиться між вузлами комп'ютерної мережі, забезпечує надійне збереження інформації та унеможливлює її підробку. Блокчейн може допомогти керівникам проєктів знизити витрати на моніторинг і контроль, підвищуючи загальну ефективність проєкту. Він також може покращити моніторинг виконання завдань за часом, дозволяючи кожному учаснику оновлювати свій прогрес у реальному часі без необхідності залучення координатора проєкту, зменшуючи витрати на вимірювання для контролю термінів виконання.

Література:

1. Fadhil A. A., Alsarraj R. G., Altaie A. M. Software cost estimation based on dolphin algorithm. IEEE Access. 2020. 8. 75279-75287.
2. Singal P., Kumari A. C., Sharma P. Estimation of software development effort: A differential evolution approach. Procedia Computer Science. 2020. 167. 2643-2652.
3. Ezghari S., Zahi A. Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method. Applied Soft Computing. 2018. 67. 540-557.
4. Zare F., Zare H. K., Fallahnezhad M. S. Software effort estimation based on the optimal Bayesian belief network. Applied Soft Computing. 2016. 49. 968-980.
5. Pressman R. S. Software Engineering: a practitioners approach. URL: https://invent.ilmkidunya.com/images/Section/12.pdf.
6. Rizwan M., Nadeem A., Sindhu M.A. Analyses of classifier’s performance measures used in software fault prediction studies. IEEE Access. 2019. 7. 82764-82775.
7. Herroelen W., Leus R. On the merits and pitfalls of critical chain scheduling. Journal of Operations Management. 2001. 19(5). 559-577.
8. Kannan J., Chitra G. Critical chain over critical path in construction projects. International Journal of Engineering and Management Research. 2017. 7 (1). 338-344.

Відкрити тези доповіді »

30.11.2024 20:38

СИСТЕМА КОНТРОЛЮ ПОКАЗНИКІВ СЕРЕДОВИЩА

[1. Інформаційні системи і технології]

Автор: **Ревуцький Володимир Андрійович, студент, Чернівецький національний університет імені Юрія Федьковича, Україна**

Відкрити тези доповіді »

01.12.2024 13:48

УПРАВЛІННЯ ПРОЄКТАМИ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ ТА НЕЙРОННИХ МЕРЕЖ

[1. Інформаційні системи і технології]

Автор: **Рипіч Богдан Віталійович, магістр, Західноукраїнський національний університет, м. Тернопіль**

Відкрити тези доповіді »

- 1
- 2

Приймаються тези доповідей міжнародної науково-практичної інтернет-конференції на тему:

Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 94)

Термін подання матеріалів

11 грудня 2024

До початку конференції залишилось днів 10 07 год. 45 хв. 16 сек.

Подати заявку

Конференції

Конференції 2024

Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 84) (18-19.01.2024)

- 1. Інформаційні системи і технології 28
- 2. Економічні науки 14
- 3. Технічні науки 12

Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 85) (15-16.02.2024)

- 1. Інформаційні системи і технології 16
- 2. Економічні науки 12
- 3. Технічні науки 11

Рипіч Богдан Віталійович, магістр

Західноукраїнський національний університет, м. Тернопіль

УПРАВЛІННЯ ПРОЄКТАМИ З РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ ТА НЕЙРОННИХ МЕРЕЖ

Сучасні методи оцінки зусиль використовують машинне навчання для врахування складних залежностей і роботи з великими даними. Ансамблеві підходи, такі як Random Forest і Gradient Boosting, забезпечують високу точність завдяки комбінуванню кількох моделей [1]. Глибокі та рекурентні нейронні мережі ефективно моделюють нелінійні зв’язки між параметрами проєкту [2]. Водночас ML-методи потребують значних обчислювальних ресурсів і якісних великих даних, що ускладнює їх застосування в невеликих проєктах [3].

Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж є інноваційним підходом, який поєднує переваги методів нечіткої логіки для роботи з невизначеністю та потужність нейронних мереж у прогнозуванні складних залежностей. Цей метод розроблено для забезпечення більш точного прогнозування зусиль, часу та ресурсів, необхідних для реалізації програмного проєкту, що дозволяє ефективніше керувати процесами розробки, оптимізувати витрати та зменшувати ризики, пов’язані з невизначеністю та змінами у проєкті.

Основні елементи концепції:

2. Нечітка логіка для роботи з невизначеністю.

1.1. Нечіткі множини та правила. Нечітка логіка використовується для формалізації невизначених та розмитих понять, таких як "великий", "складний", "середній", що часто виникають у процесі управління програмними проєктами. Це дозволяє створити нечіткі множини для різних параметрів проєкту (наприклад, досвід команди, складність задач, розмір бази даних) та формувати правила на основі цих множин.

1.2. Мамдані-метод. Найбільш поширеним способом використання нечіткої логіки є Мамдані-метод, який дозволяє створювати правила у вигляді "Якщо-то" (If-Then). Наприклад, "Якщо проєкт має високу складність та великий розмір, то необхідний високий рівень зусиль". Нечітка система на основі цих правил дозволяє автоматизувати процес прийняття рішень при розподілі ресурсів та плануванні строків.

1.3. Дефазифікація. Після обчислення нечітких значень відбувається процес дефазифікації, який перетворює результати у чіткі значення, які використовуються для подальшого управління проєктом. Це дозволяє приймати конкретні рішення щодо планування ресурсів, складу команди та розподілу часу.

2. Нейронні мережі для прогнозування та адаптації.

2.1. Рекурентні нейронні мережі (RNN). Рекурентні нейронні мережі, зокрема гіперболічні тангенсні рекурентні радіальні нейронні мережі (HTRR-RNN), використовуються для моделювання складних залежностей між параметрами проєкту та прогнозування зусиль. RNN добре підходять для обробки послідовних даних та врахування попередніх результатів, що дозволяє їм краще моделювати динаміку проєктів у часі.

2.2. Навчання на основі історичних даних. Нейронна мережа навчається на основі історичних даних проєктів, що включають такі параметри, як розмір програмного продукту, досвід команди, тривалість попередніх проєктів, складність завдань тощо. Це дозволяє їй виявляти приховані закономірності та робити точніші прогнози зусиль для нових проєктів.

2.3. Адаптація до змін у проєкті. Використання нейронних мереж дозволяє адаптуватися до змінних умов проєкту, таких як зміна вимог або збільшення складності. Мережа може оновлювати свої параметри на основі нових даних, що дозволяє керівникам проєктів оперативно реагувати на зміни та коригувати прогноз зусиль.

3. Інтеграція нечіткої логіки та нейронних мереж.

3.1. Поєднання переваг обох підходів. Метод поєднує здатність нечіткої логіки обробляти невизначеність із можливістю нейронних мереж моделювати

складні нелінійні залежності. Наприклад, на етапі кластеризації проєктів нечітка логіка може використовуватися для визначення подібності між проєктами на основі різних критеріїв, а нейронні мережі — для точного прогнозування зусиль на основі виділених ознак.

3.2. Нечітко-нейронний підхід. Це підхід, у якому нечітка логіка використовується на початкових етапах для формування правил та розподілу даних, а нейронна мережа застосовується для прогнозування та корекції результатів. Така інтеграція дозволяє зменшити обчислювальну складність процесу, підвищити точність та забезпечити більш гнучке управління проєктами.

Переваги та практичні аспекти методу:

– підвищення точності прогнозування зусиль;
– оптимізація використання ресурсів;
– адаптивність до динамічних змін;
– прийняття обґрунтованих рішень.

Метод управління проєктами з розробки програмного забезпечення на основі нечіткої логіки та нейронних мереж є потужним інструментом для прогнозування зусиль у сучасних програмних проєктах. Він забезпечує високу точність та гнучкість у змінних умовах, що дозволяє зменшити ризики, пов’язані з невизначеністю, та підвищити ефективність реалізації програмних проєктів. Це робить його особливо цінним для великих IT-компаній, які працюють у конкурентному середовищі та прагнуть оптимізувати свої процеси управління проєктами.

Література

1. Mittas, N., Angelis, L. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. IEEE Transactions on Software Engineering. 2013. 39(4). 537–551.

2. Ezghari, S., Zahi, A. Uncertainty management in software effort estimation using a consistent fuzzy analogy-based method. Applied Soft Computing. 2018. 67. 540–557.

3. Nhung, H. L. T. K., Van Hai, V., Silhavy, R., Prokopova, Z., Silhavy, P. Parametric software effort estimation based on optimizing correction factors and multiple linear regression. IEEE Access. 2022. 10. 2963–2986.