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

ГРИЦЮК Ганна Іванівна

Програмний модуль класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних / Software Module for Classification of Software Projects Based on Machine Learning Methods and Big Data Analytics

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконала студентка групи КН-42
Г.І. Грицюк

_____

Науковий керівник:
д.т.н., професор М.П. Комар

_____

Кваліфікаційну роботу допущено
до захисту
«___»_____ 2026 р.

Завідувач кафедри
_____ Н.В. Дзюбановська

Тернопіль – 2026

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

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ
ГРИЦЮК Ганна Іванівна
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Програмний модуль класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних / Software Module for Classification of Software Projects Based on Machine Learning Methods and Big Data Analytics
керівник роботи д.т.н., професор М. П. Комар

затверджені наказом по університету від 01 грудня 2025 року № 894.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити
- визначити сукупність ознак, придатних для опису програмних проєктів;
- обрати методи попередньої обробки та аналізу даних;
- обґрунтувати вибір алгоритму машинного навчання для задачі класифікації;
- спроєктувати структуру програмного модуля;
- реалізувати програмний модуль та перевірити його працездатність;
- провести експериментальне оцінювання результатів класифікації.

5. Перелік графічного матеріалу в роботі
- концептуальна схема інформаційної моделі програмного проєкту;
- схема вибору методу машинного навчання для класифікації програмних проєктів;
- архітектура програмного модуля класифікації програмних проєктів;
- схема алгоритму класифікації програмних проєктів.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 01 грудня 2025 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. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 15.03.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06. 2026 р.	

Студентка_____ Г. І. Грицюк
підпис

Керівник роботи_____ д.т.н., професор М. П. Комар
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом в 88 сторінок і містить 7 ілюстрацій, 14 таблиць, 2 додатки та 35 використаних джерел.

Метою кваліфікаційної роботи є розроблення програмного модуля класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних, який забезпечує автоматизоване віднесення проєкту до заданого класу за сукупністю визначених ознак.

Методи дослідження включають аналіз і узагальнення наукових джерел для вивчення сучасного стану проблеми; методи інтелектуального аналізу даних для опрацювання набору ознак; методи машинного навчання для побудови класифікаційної моделі; методи програмної інженерії для проєктування архітектури програмного модуля; експериментальний метод для перевірки працездатності розробленого рішення та оцінювання його ефективності.

Сформовано архітектуру програмного модуля класифікації програмних проєктів, яка включає рівень введення даних, рівень попередньої обробки, рівень машинного навчання, рівень керування результатами та рівень взаємодії з користувачем. Для реалізації класифікації обґрунтовано використання алгоритму випадкового лісу.

Результати дослідження можуть бути використані для створення програмних засобів автоматизованого аналізу програмних проєктів, побудови систем підтримки прийняття рішень у програмній інженерії, впорядкування великих масивів проєктних даних та оцінювання складності програмних систем, прогнозування ризиків розроблення.

Ключові слова: ПРОГРАМНИЙ ПРОЄКТ, КЛАСИФІКАЦІЯ, МАШИННЕ НАВЧАННЯ, АНАЛІЗ ВЕЛИКИХ ДАНИХ, ПРОГРАМНА ІНЖЕНЕРІЯ, ПРОГРАМНИЙ МОДУЛЬ.

ANNOTATION

Qualification work on the topic «Software Module for Classification of Software Projects Based on Machine Learning Methods and Big Data Analytics» for Bachelor's degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 88 pages and it contains 7 figures, 14 tables, 2 annexes and 35 sources.

The purpose of the qualification work is to develop a software module for classifying software projects based on machine learning methods and big data analysis, which provides automated assignment of a project to a specified class according to a set of defined features.

The research methods include the analysis and generalization of scientific sources to study the current state of the problem; data mining methods for processing the feature set; machine learning methods for building a classification model; software engineering methods for designing the architecture of the software module; and an experimental method for verifying the operability of the developed solution and evaluating its effectiveness.

The architecture of the software module for classifying software projects has been formed. It includes the data input level, the preprocessing level, the machine learning level, the results management level, and the user interaction level. The use of the Random Forest algorithm has been substantiated for the implementation of classification.

The research results can be used to create software tools for automated analysis of software projects, develop decision support systems in software engineering, organize large volumes of project data, assess the complexity of software systems, and predict development risks.

Keywords: SOFTWARE PROJECT, CLASSIFICATION, MACHINE LEARNING, BIG DATA ANALYSIS, SOFTWARE ENGINEERING, SOFTWARE MODULE.

ЗМІСТ

ВСТУП

Стрімка цифровізація економіки, управління та виробничих процесів зумовила суттєве зростання кількості програмних проєктів, їх складності, різноманітності технологічних стеків і обсягів супровідних даних. У сучасній практиці програмної інженерії програмний проєкт уже не розглядається лише як сукупність вихідного коду та технічної документації. Він являє собою багатовимірний об’єкт, що характеризується набором технічних, організаційних і процесних ознак: архітектурними рішеннями, мовами програмування, моделями взаємодії компонентів, способами керування розробленням, інтенсивністю змін, структурою репозиторію, метриками якості та іншими параметрами [6, 8, 23, 24]. За таких умов зростає потреба в інструментах, здатних автоматизовано аналізувати програмні проєкти та відносити їх до певних класів на підставі сукупності релевантних характеристик.

Задача класифікації програмних проєктів є важливою з кількох причин. По-перше, її розв’язання дає змогу впорядковувати великі масиви проєктних даних, що особливо актуально для організацій, які одночасно підтримують значну кількість програмних продуктів. По-друге, результати класифікації можуть бути використані для подальшого ухвалення рішень щодо вибору підходів до розроблення, оцінювання складності, прогнозування ризиків, формування команд, планування тестування та супроводу. По-третє, автоматизована класифікація створює основу для побудови інтелектуальних систем підтримки прийняття рішень у сфері програмної інженерії [1, 6, 21, 24].

Традиційні підходи до аналізу програмних проєктів часто ґрунтуються на ручному опрацюванні описів, експертному оцінюванні або застосуванні жорстких правил, що в умовах зростання обсягів даних стає малоефективним. Використання методів машинного навчання дає змогу перейти від суб’єктивних або надмірно спрощених схем оцінювання до формалізованих моделей, здатних виявляти приховані закономірності у даних, працювати з багатьма ознаками одночасно та забезпечувати відтворюваність результатів [2, 5, 9, 11, 18]. Особливого значення така постановка задачі набуває в поєднанні з технологіями аналізу великих даних,

оскільки сучасне програмне середовище генерує значні обсяги інформації, що надходить із репозиторіїв коду, систем керування версіями, засобів безперервної інтеграції, систем відстеження помилок, журналів виконання та супровідної документації [6, 7, 22, 30].

Актуальність теми зумовлена також тим, що у працях, присвячених software analytics і mining software repositories, послідовно підкреслюється значення даних як ресурсу для вдосконалення інженерних процесів, контролю якості та підвищення обґрунтованості рішень у життєвому циклі програмного забезпечення [8, 21, 23, 24]. Водночас у прикладній площині все ще спостерігається дефіцит доступних програмних модулів, орієнтованих саме на класифікацію програмних проєктів як цілісних об’єктів. Значна частина наявних досліджень зосереджена або на суміжних задачах – прогнозуванні дефектів, класифікації потоків даних, аналізі вимог, розпізнаванні шкідливого програмного забезпечення, – або на загальних підходах до машинного навчання без адаптації до специфіки програмної інженерії [12, 16, 17, 21, 25]. Це вказує на доцільність розроблення спеціалізованого програмного модуля, який поєднував би засоби підготовки даних, навчання моделі та автоматичного визначення класу програмного проєкту.

Отже, розроблення програмного модуля класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних є актуальним науково-прикладним завданням, що поєднує сучасні підходи програмної інженерії, data-driven аналізу та інтелектуальної підтримки прийняття рішень.

Метою кваліфікаційної роботи є розроблення програмного модуля класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних, який забезпечує автоматизоване віднесення проєкту до заданого класу за сукупністю визначених ознак.

Для досягнення поставленої мети доцільно розв’язати такі основні завдання:

– визначити сукупність ознак, придатних для опису програмних проєктів;

– обрати методи попередньої обробки та аналізу даних;

– обґрунтувати вибір алгоритму машинного навчання для задачі класифікації;

– спроєктувати структуру програмного модуля;

– реалізувати програмний модуль та перевірити його працездатність;

– провести експериментальне оцінювання результатів класифікації.

Об’єктом дослідження є процес автоматизованої класифікації програмних проєктів.

Предметом дослідження є методи машинного навчання, засоби аналізу великих даних та програмні механізми реалізації модуля класифікації програмних проєктів.

У роботі використано такі методи дослідження: аналіз і узагальнення наукових джерел для вивчення сучасного стану проблеми; методи інтелектуального аналізу даних для опрацювання набору ознак; методи машинного навчання для побудови класифікаційної моделі; методи програмної інженерії для проєктування архітектури програмного модуля; експериментальний метод для перевірки працездатності розробленого рішення та оцінювання його ефективності за показниками якості класифікації.

Результати кваліфікаційної роботи апробовані та опубліковані у матеріалах IV Всеукраїнської науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» (ІКСМ), м. Тернопіль, ЗУНУ, 20 травня 2026 р. (додаток Б).

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

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ КЛАСИФІКАЦІЇ ПРОГРАМНИХ ПРОЄКТІВ

1.1 Опис предметної області

У сучасних умовах розвитку цифрових технологій програмні проєкти стали однією з базових форм створення, впровадження та супроводу інформаційних систем у різних сферах діяльності. Програмний проєкт доцільно розглядати не лише як результат розроблення певного програмного продукту, а як складний багатокомпонентний об’єкт, що охоплює цілі, вимоги, архітектурні рішення, програмний код, використовувані технології, ресурси команди, організаційні процеси, результати тестування та супровідну документацію. Такий підхід зумовлений тим, що сучасна програмна інженерія орієнтована не тільки на створення програмного забезпечення, а й на системне керування всім його життєвим циклом [6, 8, 23].

Програмний проєкт формується під впливом значної кількості чинників, серед яких особливу роль відіграють функціональне призначення системи, тип прикладної області, масштаб розроблення, мови програмування, фреймворки, моделі даних, архітектурний стиль, методологія організації робіт, рівень складності інтеграції, характеристики якості та обмеження щодо ресурсів. У результаті навіть проєкти, які належать до однієї галузі, можуть суттєво відрізнятися за структурою, технологічним наповненням і вимогами до реалізації. Саме тому виникає потреба в їх систематизації, впорядкуванні та автоматизованому аналізі [6, 13, 24].

Одним із ефективних засобів такої систематизації є класифікація. У загальному розумінні класифікація передбачає віднесення об’єкта до певного класу на основі сукупності його ознак. У контексті програмної інженерії це означає визначення належності програмного проєкту до наперед заданої категорії за технічними, структурними, процесними або якісними характеристиками. Практична цінність такого підходу полягає в тому, що класифікація дає змогу уніфікувати аналіз великої кількості проєктів, спростити пошук подібних рішень, підтримати ухвалення рішень щодо вибору інструментів розроблення, оцінити потенційні ризики та сформувати основу для подальшого прогнозування якості або

складності програмного продукту [1, 6, 21, 24].

Предметна область дослідження охоплює сукупність методів, моделей і програмних засобів, що забезпечують автоматизоване опрацювання даних про програмні проєкти. Її особливістю є поєднання підходів програмної інженерії, інтелектуального аналізу даних, машинного навчання та технологій великих даних. На відміну від традиційного ручного аналізу, що спирається на експертне оцінювання або фіксовані правила, сучасні інтелектуальні підходи орієнтовані на виявлення закономірностей у великих масивах даних та побудову моделей, здатних автоматично приймати рішення щодо належності об’єкта до певного класу [5, 7, 9, 11].

Необхідність використання методів машинного навчання у цій предметній області пояснюється насамперед складністю та багатовимірністю опису програмного проєкту. Якщо кількість параметрів зростає, а взаємозв’язки між ними стають нелінійними, жорстко задані правила або спрощені евристики втрачають ефективність. Методи машинного навчання, навпаки, дають змогу навчати модель на основі даних, виявляти приховані залежності між ознаками та формувати рішення, що спирається не на окремі локальні правила, а на статистично або алгоритмічно виведені закономірності [2, 5, 9, 11, 18]. Для задач класифікації особливо важливими є здатність працювати з багатьма вхідними параметрами, адаптивність до нових даних і можливість оцінювання якості отриманих результатів.

Слід зазначити, що в програмній інженерії вже сформувався напрям, орієнтований на аналіз даних програмних систем. У науковій літературі він пов’язується з поняттями software analytics та mining software repositories. Йдеться про використання даних, що накопичуються в процесі розроблення та супроводу програмного забезпечення, для підтримки інженерних рішень. До таких даних належать характеристики репозиторіїв вихідного коду, історія комітів, метадані систем контролю версій, звіти про дефекти, результати тестування, журнали виконання, специфікації вимог, відомості про архітектуру та документація [6, 8, 21, 24]. Наявність таких джерел створює підґрунтя для побудови інтелектуальних моделей, що можуть використовуватися і для задач класифікації програмних

проєктів.

Водночас програмний проєкт як об’єкт аналізу має низку специфічних рис. По-перше, він є неоднорідним за структурою, оскільки об’єднує різнотипні дані: числові показники, категоріальні ознаки, текстові описи, технічну документацію та артефакти програмного коду. По-друге, значна частина його характеристик змінюється у часі, оскільки проєкт проходить різні етапи життєвого циклу, набуває нових компонентів, модифікується та адаптується до нових вимог. По-третє, реальні проєктні дані можуть бути неповними, зашумленими або неузгодженими, що ускладнює їх безпосереднє використання в аналітичних моделях [7, 17, 22, 30]. Саме тому в межах предметної області важливого значення набувають процедури попередньої обробки даних, відбору інформативних ознак та перетворення сирих даних у зручне для класифікації представлення.

З огляду на тему кваліфікаційної роботи, класифікація програмних проєктів може здійснюватися за різними критеріями. Зокрема, доцільним є поділ проєктів за типом програмного продукту, сферою застосування, масштабом, складністю, архітектурним стилем, рівнем ризику, використаною методологією розроблення або іншими релевантними характеристиками. Вибір конкретної схеми класифікації залежить від поставленої прикладної задачі та наявного набору ознак. Для кваліфікаційної роботи важливо, щоб така схема була достатньо формалізованою, придатною для реалізації у вигляді програмного модуля та такою, що допускає подальше експериментальне оцінювання [1, 12, 21, 25].

У межах цієї предметної області особливого значення набуває поняття ознаки програмного проєкту. Під ознаками доцільно розуміти параметри, які можуть бути використані як вхідні дані для моделі класифікації. До них можуть належати кількість модулів, розмір кодової бази, число файлів, кількість залежностей, мови програмування, частота змін, кількість учасників команди, тривалість активної розробки, наявність тестового покриття, тип архітектури, використання зовнішніх сервісів та інші показники. Окрім цього, важливу роль можуть відігравати похідні характеристики, сформовані в результаті агрегації, нормалізації або перетворення первинних даних [7, 11, 18, 29, 32].

З позицій програмної реалізації задача класифікації програмних проєктів

передбачає побудову програмного модуля, який виконує кілька взаємопов’язаних функцій. По-перше, такий модуль має забезпечувати збирання або завантаження даних про проєкт. По-друге, повинна бути реалізована підготовка даних: очищення, кодування категоріальних значень, нормалізація числових ознак, усунення пропусків, формування навчальної та тестової вибірок. По-третє, модуль має містити засоби навчання класифікаційної моделі та подальшого використання навченої моделі для передбачення класу нового об’єкта. По-четверте, доцільною є наявність підсистеми оцінювання якості, що дозволяє аналізувати точність, повноту, F1-міру та інші показники результативності [2, 5, 11, 18].

У зв’язку з цим предметна область має чітко виражений міждисциплінарний характер. Вона поєднує теоретичні положення програмної інженерії, принципи побудови програмних систем, методи машинного навчання, інструменти аналізу великих даних та підходи до оцінювання якості моделей. Така інтеграція є закономірною, оскільки сучасні програмні системи самі стали джерелом великих обсягів даних, а отже, потребують відповідних підходів до їх оброблення та інтерпретації [6, 14, 22, 23, 30].

Важливо враховувати і те, що використання великих даних у програмній інженерії не зводиться лише до роботи з великими обсягами інформації. Йдеться також про різноманітність джерел, високу швидкість оновлення даних, неоднорідність форматів і потребу в масштабованих засобах зберігання та обробки. Саме ці характеристики роблять традиційні локальні підходи недостатніми та обґрунтовують необхідність залучення засобів data mining, аналітичних бібліотек і програмних платформ, здатних підтримувати ефективне навчання моделей на реальних даних [6, 7, 22, 30]. У цьому аспекті задача класифікації програмних проєктів органічно вписується в ширший контекст data-driven підходів до програмної інженерії.

Отже, предметна область кваліфікаційної роботи охоплює процеси збирання, підготовки, аналізу та інтерпретації даних про програмні проєкти з метою їх автоматизованої класифікації. Її специфіка визначається складністю структури програмних проєктів, багатовимірністю опису, наявністю різнорідних джерел даних і необхідністю використання інтелектуальних методів для виявлення

закономірностей. Це створює достатні теоретичні та прикладні підстави для розроблення програмного модуля класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних [5, 6, 8, 11, 21, 24, 29-32].

1.2 Аналіз існуючих підходів та методів

Задача класифікації програмних проєктів належить до класу задач інтелектуального аналізу даних, у межах яких необхідно на основі сукупності ознак віднести об’єкт до одного з наперед визначених класів. Для її розв’язання можуть застосовуватися як класичні статистичні та алгоритмічні підходи, так і сучасні методи машинного навчання, орієнтовані на роботу з багатовимірними, неоднорідними та великими за обсягом наборами даних [2, 5, 7, 9, 11, 29-32]. У контексті програмної інженерії специфіка цієї задачі полягає в тому, що програмний проєкт як об’єкт класифікації поєднує структурні, технологічні, процесні та інколи текстові характеристики, а це потребує ретельного вибору методів моделювання.

У наукових джерелах простежуються кілька основних напрямів, дотичних до теми дослідження. Перший напрям пов’язаний із загальною теорією машинного навчання та класифікації. Другий охоплює методи аналізу даних і великих даних як технологічне підґрунтя підготовки ознак і побудови моделей. Третій стосується безпосередньо програмної інженерії, де дані програмних систем розглядаються як джерело для аналітики, підтримки рішень і побудови інтелектуальних інструментів [6, 8, 21, 23, 24]. Четвертий напрям репрезентують прикладні дослідження, у яких класифікаційні моделі застосовано до суміжних задач: оброблення потоків даних, виявлення дефектів, класифікації шкідливого програмного забезпечення, оцінювання якості або ранжування об’єктів [1, 12, 16, 17, 25, 27].

На теоретичному рівні базові засади класифікації сформовано у працях, присвячених статистичному навчанню, data mining і практичному машинному навчанню. У роботах L. Breiman, T. Hastie, R. Tibshirani, J. Friedman, G. James та A. Géron описано принципи побудови моделей класифікації, роботи з ознаками, уникнення перенавчання, вибору критеріїв якості та порівняння алгоритмів [2, 5, 9,

11]. Важливе значення мають також підходи, реалізовані в бібліотеці scikit-learn, яка є одним із найбільш поширених інструментів для побудови класифікаційних моделей у прикладних дослідженнях [18]. Зазначені джерела є методологічною основою для розроблення програмного модуля класифікації, оскільки дозволяють обґрунтувати вибір алгоритмів, етапів підготовки даних та способів оцінювання результатів.

Одним із найвідоміших класичних алгоритмів класифікації є метод випадкового лісу. Його перевага полягає у поєднанні високої практичної ефективності, здатності працювати з різнорідними ознаками, стійкості до шуму та можливості оцінювання важливості ознак [2]. Для задачі класифікації програмних проєктів це є суттєвою перевагою, оскільки вхідні дані можуть одночасно включати числові, бінарні та категоріальні характеристики. Крім того, ансамблеві методи часто забезпечують більш стабільні результати, ніж окремі базові алгоритми, що робить їх доцільними для побудови прикладного модуля.

Поряд із класичними ансамблевими підходами у джерелах розглядаються методи, орієнтовані на попередню обробку та трансформацію простору ознак. У роботі S. H. Jung та J. C. Kim запропоновано підхід до підвищення ефективності класифікаційної моделі на основі модифікованого K-means із використанням PCA та обробленням викидів [12]. Значення цієї праці для теми дослідження полягає не стільки у прямому використанні кластеризації як інструмента остаточної класифікації, скільки в акценті на необхідності зниження розмірності та усунення шуму в даних. Для програмних проєктів, описаних великою кількістю параметрів, попереднє перетворення ознак може позитивно впливати на якість класифікації та швидкодію програмного модуля.

Окрему групу становлять дослідження, присвячені класифікації в умовах складних даних. Так, у праці K.-Z. Lu, C.-F. Chen, H. Cai та співавторів розглянуто онлайн-алгоритм класифікації для потоків даних, яким властиві concept drift та дисбаланс класів [17]. Хоча програмні проєкти не завжди аналізуються як потокові об’єкти в реальному часі, ця робота є важливою з позицій урахування змінності даних і проблеми нерівномірного представлення класів. У реальних наборах проєктів також може виникати ситуація, коли одні категорії представлені значно

ширше за інші, а це потребує врахування при підготовці даних та оцінюванні якості моделі.

У низці праць класифікаційні підходи продемонстровано на прикладних задачах інших предметних областей. Зокрема, W. Xing та Y. Bei розглядають класифікацію медичних великих даних на основі KNN [25]. Незважаючи на іншу сферу застосування, ця робота показує, що прості алгоритми можуть залишатися ефективними за умови належної підготовки ознак. Водночас для задачі класифікації програмних проєктів метод KNN має певні обмеження, пов'язані з чутливістю до масштабу ознак, збільшенням обчислювальних витрат при рості вибірки та складністю роботи з високорозмірними даними [5, 11, 25]. Це дає підстави розглядати його радше як базову модель для порівняння, а не як основний робочий алгоритм.

Значний інтерес становлять і дослідження, у яких для класифікації використовуються глибокі моделі. Наприклад, у роботі Q. Lin, N. Li, Q. Qi та співавторів класифікацію IoT-шкідливого програмного забезпечення виконано на основі послідовностей API-викликів із застосуванням згорткових нейронних мереж [16]. Такий підхід підтверджує ефективність глибокого навчання у випадках, коли вхідні дані мають складну внутрішню структуру або послідовнісний характер. Однак для задачі класифікації програмних проєктів застосування глибоких нейронних мереж не завжди є оптимальним. Якщо набір даних відносно невеликий, а ознаки мають переважно табличну форму, простіші моделі можуть виявитися більш інтерпретованими, стійкими та придатними для практичної реалізації в межах кваліфікаційної роботи [5, 11, 18].

У межах програмної інженерії важливе місце займає напрям *software analytics*, у якому дані програмних систем використовуються для підтримки технічних і організаційних рішень. У праці F. Gurcan та N. E. Cagiltay досліджено знаннєві домени та набори компетентностей у big data software engineering на основі тематичного моделювання [6]. Роботи A. E. Hassan акцентують увагу на майбутньому аналізу даних програмної інженерії та перспективності підходів mining software repositories [8]. Праця C. Wijesiriwardana та P. Wimalaratne пропонує багаторівневий механізм абстракції для software engineering data analytics [24].

Зазначені дослідження не зводяться лише до класифікації, проте вони формують теоретичне поле, в межах якого програмні проєкти розглядаються як джерело даних, придатних до формалізації, структурування та інтелектуального аналізу [6, 8, 24].

Суттєве значення для теми мають праці, присвячені застосуванню машинного навчання до суміжних задач програмної інженерії. У систематичному огляді S. Stradowski, L. L. Minku, W. Pedrycz та S. Wagner проаналізовано використання машинного навчання для прогнозування дефектів у програмному забезпеченні [21]. Хоча ця праця присвячена іншій прикладній задачі, її результати підтверджують загальну доцільність використання класифікаційних моделей у сфері програмної інженерії. Водночас у дослідженні S. Ali, P. Arcaini, D. Pradhan та співавторів розглянуто показники якості в search-based software engineering [1], що є важливим для вибору критеріїв оцінювання ефективності моделі. Таким чином, навіть суміжні напрями досліджень підтверджують перспективність інтелектуальних підходів для автоматизованого аналізу програмних об’єктів.

Крім переваг, сучасні методи класифікації мають і певні обмеження. Насамперед ідеться про залежність якості результатів від репрезентативності та структури даних. Для програмних проєктів ця проблема є особливо відчутною, оскільки відомості про них можуть бути неповними, неуніфікованими або надто різнорідними. Частина ознак може бути легко формалізована, тоді як інші потребують додаткового перетворення або навіть окремого вилучення з текстових описів, документації чи репозиторіїв [7, 24, 29-32]. Окрім цього, окремі алгоритми погано працюють за наявності великої кількості слабкоінформативних ознак, тоді як інші потребують значних обчислювальних ресурсів або складнішого налаштування гіперпараметрів [5, 9, 11].

З урахуванням викладеного доцільно узагальнити основні групи підходів, які можуть бути використані в задачі класифікації програмних проєктів (таблиця 1.1).

Як показано в таблиці 1.1, універсального підходу, однаково придатного для всіх варіантів задачі, не існує. Вибір конкретного методу повинен визначатися структурою даних, кількістю ознак, обсягом вибірки, складністю моделі та вимогами до інтерпретованості результатів. Для даної роботи найбільш доцільним

видається поєднання кількох груп засобів: попередньої обробки даних, базових класифікаційних моделей для порівняння та одного з ансамблевих методів як основного робочого інструмента [2, 5, 11, 12, 18].

Таблиця 1.1 – Порівняльна характеристика основних підходів до класифікації

Підхід / група методів	Характеристика	Переваги	Обмеження	Доцільність для задачі класифікації програмних проєктів
Класичні алгоритми класифікації	Логістична регресія, дерева рішень, KNN, SVM	Простота реалізації, зрозумілість, можливість базового порівняння	Чутливість окремих моделей до масштабу ознак, шуму або високої розмірності	Доцільні як базові моделі для експериментів [5, 11, 18, 25]
Ансамблеві методи	Random Forest та споріднені підходи	Висока стійкість, добра якість, робота з різнорідними ознаками	Менша інтерпретованість порівняно з простими моделями	Висока доцільність для основної моделі [2, 5, 11]
Методи зниження розмірності та попередньої обробки	PCA, оброблення викидів, нормалізація	Підвищення якості даних, усунення шуму, пришвидшення навчання	Не є самостійним засобом остаточної класифікації	Доцільні як етап підготовки даних [12, 17]
Глибоке навчання	CNN та інші нейромережеві моделі	Ефективність для складних, послідовнісних або неструктурованих даних	Високі вимоги до даних і ресурсів, складніша інтерпретація	Доцільність залежить від структури й обсягу набору даних [5, 16]
Аналітика програмних систем	Software analytics, mining software repositories	Орієнтація на специфіку програмної інженерії, робота з реальними артефактами розроблення	Часто не дає готової універсальної моделі класифікації	Є концептуальною основою для побудови власного модуля [6, 8, 21, 24]

Окремо слід розглянути праці, пов’язані з архітектурними та інженерними аспектами програмних систем. Дослідження N. Kahani, M. Bagherzadeh, J. R. Cordy та співавторів містить огляд і класифікацію засобів трансформації моделей [13]. У роботі A. Ramírez-Noriega, Y. Martínez-Ramírez, J. F. Figueroa-Pérez та співавторів розглянуто програмний засіб генерації MVC-архітектури на основі ER-моделі [19].

I. Singh та S. W. Lee аналізують питання requirements engineering у хмарному блокчейн-середовищі [20]. Хоча зазначені праці не є безпосередньо джерелами класифікаційних алгоритмів, вони демонструють важливість формального подання програмних об’єктів, моделювання архітектури та зв’язку між структурою даних і програмною реалізацією. Це має практичне значення для проєктування власного програмного модуля класифікації.

Отже, аналіз існуючих підходів і методів показав, що для задачі класифікації програмних проєктів найбільш перспективним є комплексний підхід, який передбачає попередню підготовку даних, формування інформативного набору ознак, використання базових класифікаційних моделей для порівняння та застосування ансамблевого методу як основного інструмента класифікації. Такий підхід забезпечує баланс між точністю, стійкістю, практичною реалізовністю та придатністю до подальшого розширення функціоналу програмного модуля.

1.3 Постановка задачі дослідження

Проведений аналіз предметної області та існуючих підходів засвідчив, що сучасна програмна інженерія характеризується значним обсягом даних, які супроводжують програмні проєкти на всіх етапах їх життєвого циклу. Водночас наявність великої кількості технічних, структурних і процесних характеристик ускладнює їх ручне впорядкування та об’єктивне віднесення до певних класів [6, 8, 21, 24]. За таких умов актуалізується задача створення програмного засобу, здатного автоматизувати класифікацію програмних проєктів на основі методів машинного навчання та аналізу великих даних [2, 5, 11, 29-32].

У межах кваліфікаційної роботи необхідно розробити програмний модуль, який на основі вхідних характеристик програмного проєкту забезпечуватиме його автоматичне віднесення до одного із визначених класів. Передбачається, що вхідними даними виступатимуть ознаки, які описують програмний проєкт з погляду його структури, технологічного наповнення, масштабу або інших формалізованих параметрів. Результатом роботи модуля має бути визначення класу проєкту та надання можливості оцінювання якості класифікації за відповідними

метриками [1, 2, 11, 18].

Таким чином, постановка задачі дослідження полягає у розробленні програмного модуля класифікації програмних проєктів, що включає:

– формування набору інформативних ознак програмного проєкту;

– підготовку та попередню обробку даних;

– вибір і реалізацію методу машинного навчання для класифікації;

– навчання та тестування моделі;

– оцінювання якості класифікації;

– забезпечення практичної придатності програмного модуля до використання.

Для досягнення поставленої мети доцільно розв’язати такі основні завдання:

– визначити сукупність ознак, придатних для опису програмних проєктів;

– обрати методи попередньої обробки та аналізу даних;

– обґрунтувати вибір алгоритму машинного навчання для задачі класифікації;

– спроєктувати структуру програмного модуля;

– реалізувати програмний модуль та перевірити його працездатність;

– провести експериментальне оцінювання результатів класифікації.

Об’єктом дослідження є процес автоматизованої класифікації програмних проєктів.

Предметом дослідження є методи машинного навчання, засоби аналізу великих даних та програмні механізми реалізації модуля класифікації програмних проєктів.

Отже, у межах роботи передбачається побудова прикладного рішення, яке поєднуватиме процедури підготовки даних, навчання класифікаційної моделі та програмної реалізації засобу автоматизованого визначення класу програмного проєкту.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ПРОГРАМНИХ ПРОЄКТІВ

2.1 Формування інформаційної моделі даних про програмні проєкти

Розроблення програмного модуля класифікації програмних проєктів потребує попереднього формального опису тих даних, які надалі використовуватимуться для навчання моделі та виконання класифікації. Без побудови інформаційної моделі неможливо забезпечити ані цілісне подання характеристик програмного проєкту, ані коректну підготовку вхідних ознак для алгоритмів машинного навчання. Саме тому на цьому етапі доцільно визначити склад даних, їх структуру, типи, логічні зв’язки між окремими елементами та спосіб подання у вигляді, придатному для автоматизованого аналізу [5, 7, 11, 18, 29-32].

Формування інформаційної моделі має спиратися на кілька ключових принципів. По-перше, модель повинна бути достатньо інформативною, тобто містити ознаки, які дійсно можуть впливати на результат класифікації. По-друге, вона має бути компактною, оскільки надлишкова кількість слабкоінформативних параметрів здатна ускладнювати навчання моделі, підвищувати ризик перенавчання та знижувати інтерпретованість результатів [9, 11, 12]. По-третє, інформаційна модель повинна бути придатною до програмної реалізації, тобто всі її складові мають бути подані у формі, що допускає збереження, оброблення, перетворення та передачу в алгоритми машинного навчання [18, 29, 31, 32].

У загальному випадку програмний проєкт може бути описаний великою кількістю характеристик. Проте не всі вони є однаково значущими для задачі класифікації. Частина параметрів може мати суто контекстний характер, тоді як інші безпосередньо відображають структуру та властивості програмного продукту. З огляду на це доцільно згрупувати дані про програмний проєкт у кілька логічних категорій: загальні відомості про проєкт, технологічні характеристики, структурні характеристики, організаційно-процесні показники та цільовий клас проєкту. Таке групування дозволяє не лише систематизувати дані, а й побудувати послідовну схему їх подальшого опрацювання.

До загальних відомостей про проєкт доцільно віднести ідентифікаційні та

описові параметри. Йдеться, зокрема, про унікальний ідентифікатор проєкту, назву, короткий опис, предметну сферу застосування, рік створення, поточний статус або інші загальні атрибути. Не всі з них використовуватимуться як безпосередні ознаки для класифікації, однак вони мають значення для організації даних, навігації, фільтрації записів і загальної структуризації набору. Ідентифікаційні поля зазвичай не передаються до моделі як вхідні ознаки, але залишаються важливими елементами інформаційної системи.

Технологічні характеристики відображають ті аспекти проєкту, які пов’язані з реалізацією програмного забезпечення. До цієї групи можуть належати основна мова програмування, використані фреймворки, тип бази даних, вид архітектури, наявність хмарної інфраструктури, спосіб розгортання, використання зовнішніх API, а також належність до певної платформи, наприклад web, mobile, desktop, embedded або enterprise. Саме ці характеристики часто є найбільш інформативними для класифікації програмних проєктів, оскільки безпосередньо пов’язані з технічним профілем системи [6, 13, 19, 20, 24].

Структурні характеристики описують внутрішню будову програмного проєкту. До них можуть бути віднесені кількість модулів, число пакетів або каталогів, кількість вихідних файлів, обсяг кодової бази, число залежностей, наявність тестових компонентів, кількість сервісів або мікросервісів, а також складність структури взаємодії компонентів. Такі параметри є особливо цінними у випадку, коли класифікація має враховувати масштаб або архітектурну складність проєкту. Структурні показники дають змогу відобразити проєкт у вигляді числового профілю, придатного до подальшого машинного аналізу [7, 18, 21, 24].

Організаційно-процесні характеристики відображають особливості ведення розроблення. До них можуть належати тривалість активної розробки, кількість учасників команди, інтенсивність змін у репозиторії, кількість комітів, частота оновлень, наявність системи безперервної інтеграції, тип методології розроблення, кількість зареєстрованих помилок та інші подібні показники. Ці атрибути не завжди є доступними для кожного проєкту, проте за наявності вони можуть підвищити інформативність моделі, особливо якщо мета класифікації виходить за межі суто технологічного поділу і враховує складність супроводу або динаміку

життєвого циклу [1, 8, 21, 24].

Окреме місце в інформаційній моделі займає цільова змінна, тобто клас, до якого належить програмний проєкт. Саме вона виступає результативним атрибутом у задачі керованого навчання. Вибір цільового класу залежить від прикладної постановки задачі. У межах даної роботи доцільно орієнтуватися на класифікацію програмних проєктів за типом програмного продукту, наприклад web-проєкт, mobile-проєкт, desktop-проєкт, embedded-проєкт, enterprise-проєкт. Такий поділ є достатньо зрозумілим, формалізованим і придатним до практичної реалізації. Крім того, він дозволяє формувати набір ознак, логічно пов'язаних із предметом класифікації.

Для наочності основні складові інформаційної моделі доцільно подати у вигляді таблиці 2.1.

Таблиця 2.1 – Основні групи даних у інформаційній моделі програмного проєкту

Група даних	Приклади атрибутів	Призначення
Ідентифікаційні дані	ID проєкту, назва, короткий опис	Ідентифікація та супровід запису
Загальні характеристики	предметна сфера, рік створення, статус	Загальний опис проєкту
Технологічні характеристики	мова програмування, фреймворк, платформа, тип БД	Формування технічного профілю проєкту
Структурні характеристики	кількість модулів, файлів, рядків коду, залежностей	Опис внутрішньої будови та масштабу
Організаційно-процесні характеристики	кількість комітів, учасників, тривалість розробки, наявність CI/CD	Відображення динаміки розроблення
Цільова змінна	клас проєкту	Результат класифікації

Як видно з таблиці 2.1, інформаційна модель повинна поєднувати як описові, так і числові параметри. Це означає, що модель даних має враховувати різні типи атрибутів. З позицій машинного навчання їх доцільно поділити на кількісні, категоріальні, бінарні та, за потреби, текстові. Кількісні ознаки подаються числовими значеннями, наприклад кількість файлів або кількість комітів. Категоріальні ознаки відображають належність до певної категорії, наприклад тип платформи чи основна мова програмування. Бінарні ознаки фіксують наявність або

відсутність певної властивості, наприклад використання CI/CD або наявність автоматизованого тестування. Текстові атрибути, такі як короткий опис проєкту, за потреби можуть бути перетворені на структуровані ознаки, але в межах кваліфікаційної роботи доцільно обмежити їх використання або подавати у спрощеному вигляді [5, 11, 18, 29].

З практичного погляду інформаційна модель повинна бути представлена так, щоб кожен програмний проєкт відповідав одному запису в наборі даних, а кожна ознака – окремому полю цього запису. Іншими словами, зручним є табличне представлення, де рядки відповідають проєктам, а стовпці – ознакам. Такий формат добре узгоджується з більшістю сучасних бібліотек машинного навчання, зокрема scikit-learn, і спрощує виконання стандартних процедур: очищення даних, кодування, нормалізації, поділу на навчальну та тестову вибірки, навчання моделі й оцінювання результатів [11, 18]. Таким чином, логічна інформаційна модель повинна легко трансформуватися у фізичну структуру даних, наприклад таблицю CSV, DataFrame або таблицю бази даних.

При формуванні складу ознак важливо враховувати не лише їх наявність, а й якість. Часто в реальних даних окремі атрибути можуть містити пропуски, дублікати, аномальні значення або надто велику кількість унікальних категорій. Наприклад, якщо для кожного проєкту вказано конкретну назву фреймворку, але кількість різних значень дуже велика, безпосереднє включення такої ознаки може створити надмірно розріджене подання даних. У таких випадках доцільно виконувати узагальнення, наприклад об’єднувати конкретні інструменти у ширші класи. Подібні рішення є частиною побудови інформаційної моделі, оскільки саме на цьому етапі визначається, які дані будуть збережені у вихідному вигляді, а які – трансформовані [7, 11, 12].

Суттєвим етапом формування інформаційної моделі є також визначення зв’язків між атрибутами. У найпростішому варіанті модель можна розглядати як плоску таблицю ознак. Проте на концептуальному рівні між окремими групами даних існують логічні залежності. Наприклад, тип платформи може бути пов’язаний із вибором мови програмування, архітектурний стиль – із кількістю модулів, а методологія розроблення – з інтенсивністю змін у репозиторії. Не всі

такі взаємозв’язки потрібно явно фіксувати в структурі бази даних, але їх доцільно враховувати під час відбору ознак і подальшого аналізу. У ряді випадків це дозволяє формувати нові похідні показники, які краще відображають реальні властивості проєкту.

Концептуальне подання інформаційної моделі доцільно зобразити у вигляді схеми (рисунок 2.1).

Рисунок 2.1 – Концептуальна схема інформаційної моделі програмного проєкту

На рисунку 2.1 представлено узагальнену логіку побудови інформаційної моделі. Центральним об’єктом є програмний проєкт, для якого формується набір взаємодоповнювальних характеристик. Саме на основі їх сукупності створюється вектор ознак, що подається до моделі машинного навчання.

Для прикладної реалізації важливо також визначити мінімально достатній набір ознак. Надмірне розширення моделі даних може ускладнювати не тільки навчання класифікатора, а й подальшу підтримку програмного модуля. Тому в межах кваліфікаційної роботи доцільно використовувати збалансований підхід: обрати обмежену, але змістовну множину характеристик, які реально можна зібрати, зберегти й обробити в межах програмної реалізації. Орієнтовний перелік таких ознак наведено в таблиці 2.2.

Таблиця 2.2 – Орієнтовний набір ознак для класифікації програмних проєктів

Назва ознаки	Тип ознаки	Приклад значення
Платформа	категоріальна	web / mobile / desktop
Основна мова програмування	категоріальна	Python, Java, JavaScript
Тип архітектури	категоріальна	monolith / microservices
Кількість модулів	кількісна	12
Кількість файлів	кількісна	148
Обсяг кодової бази	кількісна	23500 рядків
Кількість залежностей	кількісна	17
Наявність тестів	бінарна	0 / 1
Наявність CI/CD	бінарна	0 / 1
Кількість учасників	кількісна	6
Кількість комітів	кількісна	420
Тривалість активної розробки	кількісна	14 місяців
Цільовий клас	категоріальна	enterprise

Подана в таблиці 2.2 множина ознак не є вичерпною, однак вона дозволяє сформувати достатньо інформативний профіль програмного проєкту. Важливо, що більшість із наведених параметрів піддається формалізації та може бути отримана або безпосередньо з технічних описів і репозиторіїв, або шляхом ручного заповнення у тестовому наборі даних. Це особливо важливо для програмної реалізації, оскільки надмірно складна або важкодоступна інформаційна модель може ускладнити створення навчального набору.

Ще одним важливим аспектом є підготовка даних до використання в алгоритмах машинного навчання. Оскільки інформаційна модель включає різнотипні атрибути, необхідно передбачити відповідні процедури трансформації. Для кількісних ознак доцільно застосовувати масштабування або нормалізацію, якщо це вимагає обраний алгоритм. Категоріальні ознаки повинні бути закодовані у числовому вигляді, наприклад за допомогою one-hot encoding або label encoding. Бінарні ознаки зазвичай не потребують складного перетворення. Якщо ж у моделі використовуються текстові поля, для них можуть застосовуватися додаткові методи векторизації, однак у межах даної роботи пріоритет надається табличним характеристикам, що спрощує загальну архітектуру системи [5, 11, 18, 29-32].

Доцільно також врахувати, що інформаційна модель повинна підтримувати як режим навчання, так і режим практичного використання модуля. У режимі навчання вона використовується для формування вибірки, на якій модель виявляє залежності між ознаками та класами. У режимі експлуатації ця ж структура слугує основою для подання нового проєкту, який необхідно класифікувати. Тому склад і порядок ознак, а також правила їх обробки мають залишатися узгодженими на всіх етапах функціонування системи. Це є однією з базових вимог до якісної побудови інформаційної моделі в прикладних системах машинного навчання.

Узагальнюючи викладене, можна зазначити, що формування інформаційної моделі даних про програмні проєкти є одним із ключових етапів розроблення програмного модуля класифікації. Саме на цьому етапі визначаються межі предметного подання проєкту, набір його суттєвих характеристик, типи атрибутів та їх придатність до подальшої обробки. Якісно сформована інформаційна модель забезпечує основу для побудови навчального набору даних, реалізації процедур попередньої обробки, вибору алгоритмів класифікації та досягнення коректних результатів у процесі автоматизованого визначення класу програмного проєкту [5, 6, 11, 18, 21, 24, 29-32].

Отже, у межах даної роботи інформаційна модель програмного проєкту формується як структурований набір загальних, технологічних, структурних і процесних характеристик, що разом із цільовою змінною створюють основу для реалізації та навчання класифікаційного модуля. Саме ця модель надалі визначатиме логіку підготовки даних, побудови архітектури програмного рішення та вибору методів машинного навчання.

2.2 Обґрунтування вибору методів машинного навчання

Розроблення програмного модуля класифікації програмних проєктів передбачає вибір такого методу машинного навчання, який забезпечить достатню точність, стійкість до неоднорідності ознак, прийнятну швидкодію та практичну придатність до програмної реалізації. Саме тому на цьому етапі необхідно обґрунтувати, які алгоритми доцільно розглядати як базові для порівняння, а який

із них варто обрати як основний інструмент класифікації.

У загальному випадку задача класифікації програмних проєктів належить до задач керованого навчання, у межах яких модель будується на основі набору прикладів із наперед відомими класами [5, 9, 11]. Вхідними даними при цьому виступають ознаки програмного проєкту, а вихідним результатом – його належність до одного з визначених класів. Така постановка добре узгоджується з класичними алгоритмами класифікації, реалізованими в сучасних бібліотеках машинного навчання, зокрема в scikit-learn [18].

Вибір методу машинного навчання для даної задачі повинен враховувати кілька чинників. По-перше, дані про програмні проєкти мають змішану природу, оскільки включають кількісні, категоріальні та бінарні ознаки. По-друге, кількість записів у вибірці в межах кваліфікаційної роботи, як правило, не є надто великою, тому метод повинен бути ефективним і на відносно помірних обсягах даних. По-третє, бажаною є можливість інтерпретації результатів, оскільки в практичній частині роботи важливо не лише отримати прогноз, а й пояснити, які характеристики проєкту найбільше вплинули на результат класифікації. По-четверте, обраний алгоритм має бути достатньо стійким до шуму, пропусків після попередньої обробки та потенційної нерівномірності розподілу класів [2, 5, 11, 21].

Серед найбільш поширених підходів до класифікації доцільно розглядати логістичну регресію, метод k-найближчих сусідів, дерево рішень, метод опорних векторів та ансамблеві методи. Логістична регресія є одним із базових алгоритмів класифікації, який характеризується відносною простотою реалізації та зрозумілою математичною інтерпретацією [9, 11]. Вона добре працює у випадках, коли зв’язок між ознаками та класом можна наближено подати в лінійній формі. Проте для задачі класифікації програмних проєктів така модель може виявитися надто спрощеною, оскільки взаємозв’язки між технічними та структурними характеристиками проєкту часто мають нелінійний характер.

Метод k-найближчих сусідів також може бути використаний як базовий інструмент для порівняння [5, 11, 25]. Його перевага полягає в простоті та відсутності складного етапу навчання. Однак для даної задачі цей підхід має низку обмежень. Насамперед він є чутливим до масштабу ознак, що вимагає ретельної

нормалізації даних. Крім того, його ефективність знижується за збільшення кількості ознак, а також у випадках, коли в наборі даних присутні нерелевантні або слабкоінформативні параметри. З огляду на це KNN доцільно розглядати як допоміжну модель для експериментального порівняння, але не як оптимальний основний метод.

Дерево рішень є більш гнучким підходом, оскільки воно дає змогу моделювати нелінійні залежності між ознаками та класами і формувати правила прийняття рішень у наочній формі [5, 11]. Для задачі класифікації програмних проєктів така властивість є корисною, адже рішення може залежати від комбінації кількох характеристик, наприклад платформи, мови програмування та структурної складності. Водночас окреме дерево рішень може виявляти нестійкість до змін у навчальній вибірці, бути схильним до перенавчання та демонструвати обмежену узагальнювальну здатність. Саме це зумовлює інтерес до ансамблевих підходів, які об’єднують переваги дерев рішень і водночас компенсують їх слабкі сторони.

Метод опорних векторів також вважається ефективним засобом класифікації, особливо в задачах із високорозмірними даними [5, 9, 11]. Його сильними сторонами є здатність будувати чітку межу між класами та працювати з нелінійними залежностями за допомогою ядрових функцій. Однак для прикладної реалізації в межах кваліфікаційної роботи цей метод має певні обмеження. По-перше, він є чутливим до вибору гіперпараметрів. По-друге, у разі збільшення обсягу даних або кількості класів його практична реалізація може ускладнюватися. По-третє, інтерпретація отриманих результатів є менш наочною, ніж у випадку дерев рішень або ансамблевих моделей на їх основі. Тому метод опорних векторів доцільно розглядати як один із варіантів для порівняння, але не як найбільш зручний інструмент для основного програмного модуля.

Окрему групу становлять нейромережеві методи. У сучасних дослідженнях вони демонструють високу ефективність для складних задач класифікації, особливо якщо дані мають послідовнісну, текстову або просторову структуру [5, 16]. Проте для класифікації програмних проєктів у межах цієї роботи основні вхідні характеристики мають переважно табличний характер. За таких умов застосування глибоких нейронних мереж не завжди є виправданим. Воно потребує більших

обсягів даних, ретельнішого налаштування архітектури та значніших обчислювальних ресурсів. Крім того, нейромережеві моделі складніше інтерпретувати, що знижує їх практичну зручність у навчально-дослідницькому проєкті. Тому використання глибокого навчання для основного модуля в даному випадку не є пріоритетним.

З урахуванням наведеного найбільш обґрунтованим виглядає вибір ансамблевого методу, зокрема випадкового лісу. Випадковий ліс являє собою сукупність дерев рішень, кожне з яких навчається на випадковій підмножині даних та ознак, а кінцеве рішення формується шляхом агрегування результатів усіх дерев [2]. Такий механізм забезпечує кращу узагальнювальну здатність у порівнянні з окремим деревом рішень, знижує ризик перенавчання та підвищує стійкість до шуму у вхідних даних. Саме ці властивості є особливо важливими для задачі класифікації програмних проєктів, де характеристики можуть бути різнорідними, а частина ознак – лише частково інформативною.

Переваги випадкового лісу для даної задачі можна сформулювати більш конкретно. По-перше, цей метод добре працює з табличними даними, що відповідає сформованій інформаційній моделі програмного проєкту [2, 5, 18]. По-друге, він допускає використання великої кількості ознак без критичного зниження якості класифікації. По-третє, модель дозволяє оцінити важливість окремих ознак, а це дає змогу зробити додаткові висновки щодо тих характеристик проєкту, які мають найбільший вплив на його віднесення до певного класу. По-четверте, реалізація випадкового лісу доступна в стандартних інструментах машинного навчання, що спрощує програмну інтеграцію модуля [18].

Важливо також врахувати, що в задачі класифікації програмних проєктів значення мають не лише підсумкові метрики якості, а й практична зручність використання алгоритму. Модель повинна легко інтегруватися в програмний модуль, підтримувати повторне навчання, забезпечувати класифікацію нових записів і не потребувати надмірно складної конфігурації. У цьому аспекті випадковий ліс має помітні переваги перед методами, які вимагають складного добору ядрових функцій, багатошарових архітектур або значної обчислювальної потужності [2, 5, 11].

Разом із тим обґрунтування вибору методу не повинно обмежуватися лише визначенням основного алгоритму. Для коректного експериментального дослідження доцільно використати кілька моделей: одну або дві прості базові моделі для порівняння та одну основну модель для подальшого впровадження в модуль. Такий підхід дозволяє не лише продемонструвати результат, а й показати, що вибір основного алгоритму зроблено не формально, а на підставі зіставлення кількох можливих рішень [5, 11, 18].

У межах даної роботи доцільно сформувати таку схему: логістична регресія або KNN використовуються як базовий орієнтир, дерево рішень – як інтерпретована нелінійна модель, а випадковий ліс – як основний робочий алгоритм. Такий набір є достатнім для кваліфікаційної роботи, оскільки він поєднує простоту, наочність та практичну результативність. Крім того, він дозволяє порівняти лінійний, локальний, ієрархічний та ансамблевий підходи до класифікації в межах єдиного експериментального середовища.

Для узагальнення доцільно подати порівняльну характеристику методів, що розглядаються як кандидати для використання в програмному модулі (таблиця 2.3).

Таблиця 2.3 – Порівняльна характеристика методів машинного навчання для класифікації програмних проєктів

Метод	Переваги	Недоліки	Доцільність використання
1	2	3	4
Логістична регресія	Простота, швидке навчання, зрозумілість результатів	Обмежена здатність моделювати складні нелінійні залежності	Доцільна як базова модель для порівняння [9, 11]
KNN	Простота, інтуїтивність, відсутність складного навчання	Чутливість до масштабу ознак, зниження ефективності при високій розмірності	Доцільний для базового експериментального порівняння [5, 11, 25]
Дерево рішень	Нелінійність, інтерпретованість, наочність правил	Схильність до перенавчання, нестійкість до змін вибірки	Доцільне як проміжна інтерпретована модель [5, 11]
SVM	Висока якість у низці задач, робота з високорозмірними ознаками	Складніше налаштування, менша інтерпретованість	Може використовуватись як додатковий варіант, але не є пріоритетним [5, 9, 11]

Продовження таблиці 2.3

1	2	3	4
Нейронні мережі	Гнучкість, ефективність для складних даних	Потреба у великих вибірках, складніша реалізація, слабша інтерпретованість	Для даної задачі не є основним вибором [5, 16]
Випадковий ліс	Стійкість, добра якість, робота з різнорідними ознаками, оцінка важливості ознак	Менша простота інтерпретації порівняно з одним деревом	Найбільш доцільний як основний алгоритм [2, 5, 11, 18]

Як видно з таблиці 2.3, випадковий ліс найкраще відповідає вимогам, сформульованим для програмного модуля класифікації програмних проєктів. Він поєднує достатню точність, стійкість до неоднорідності даних, відносну простоту впровадження та можливість аналізу значущості ознак. У поєднанні з попередньою обробкою даних цей метод здатний забезпечити практично придатне рішення для автоматизованого визначення класу програмного проєкту.

Окремої уваги потребує питання попередньої підготовки даних у зв’язку з вибором алгоритму. Навіть у випадку використання випадкового лісу, який є менш чутливим до масштабу ознак, ніж KNN або SVM, доцільно виконувати очищення даних, кодування категоріальних атрибутів, оброблення пропусків та перевірку на наявність аномальних значень [5, 11, 12, 18]. Якщо для окремих базових моделей використовуватимуться відстаневі або лінійні підходи, масштабування числових ознак також стає необхідним. Отже, вибір методу машинного навчання прямо пов’язаний із загальною технологією підготовки даних, яка повинна бути однаково узгодженою для всіх етапів роботи модуля.

На рисунку 2.2 доцільно подати узагальнену логіку вибору основного методу машинного навчання для програмного модуля.

На рисунку 2.2 показано, що вибір випадкового лісу не є довільним. Він зумовлений структурою даних, прикладними вимогами до модуля та результатами порівняльного аналізу можливих підходів.

Таким чином, обґрунтування вибору методів машинного навчання для класифікації програмних проєктів дає підстави зробити висновок, що для реалізації програмного модуля доцільно використати комбінований підхід. Він передбачає застосування простих моделей як базових орієнтирів для порівняння, а випадкового

лісу – як основного алгоритму класифікації.

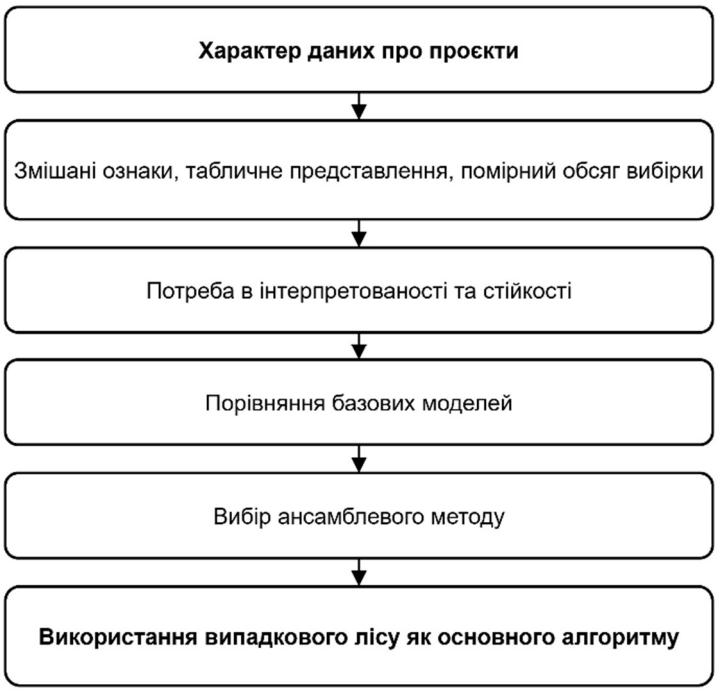

Рисунок 2.2 – Схема вибору методу машинного навчання для класифікації програмних проєктів

Такий вибір є методично вмотивованим, практично досяжним та узгодженим зі специфікою сформованої інформаційної моделі даних про програмні проєкти.

2.3 Проєктування архітектури програмного модуля

Архітектурне проєктування забезпечує перехід від теоретичної моделі до прикладної реалізації, у межах якої визначаються основні компоненти системи, їх функції, зв’язки та послідовність взаємодії. Для задачі класифікації програмних проєктів архітектура має бути побудована так, щоб підтримувати повний цикл оброблення даних: від введення або завантаження характеристик проєкту до отримання результату класифікації та його інтерпретації [5, 11, 18, 19].

У межах даної роботи програмний модуль розглядається як прикладна система, орієнтована на автоматизоване визначення класу програмного проєкту на основі заздалегідь підготовленого набору ознак. Відповідно архітектура повинна забезпечувати кілька базових властивостей: модульність, зрозумілість логіки функціонування, можливість повторного навчання моделі, придатність до

експериментального оцінювання та розширюваність у разі подальшого вдосконалення системи. Такі вимоги відповідають сучасним підходам до побудови програмних рішень у сфері програмної інженерії, де важливого значення набуває не лише функціональність, а й узгодженість внутрішньої структури системи [13, 19, 20].

Архітектура програмного модуля має враховувати специфіку задачі машинного навчання. На відміну від звичайних прикладних систем, у ній поряд із традиційними програмними компонентами необхідно передбачити блоки, пов’язані з підготовкою даних, навчанням моделі, збереженням навченої моделі, обчисленням метрик якості та виконанням прогнозування для нових даних. Тому доцільно проєктувати систему не як єдиний монолітний фрагмент коду, а як сукупність взаємопов’язаних функціональних компонентів, кожен з яких реалізує окремий етап оброблення інформації [5, 18, 29, 31].

З огляду на поставлену задачу найбільш доцільною є багатокомпонентна архітектура, у якій можна виділити п’ять основних рівнів: рівень введення даних, рівень попередньої обробки, рівень машинного навчання, рівень керування результатами та рівень взаємодії з користувачем. Такий підхід дає змогу логічно розмежувати функції системи та спростити її реалізацію. Крім того, це забезпечує можливість окремого тестування кожного компонента та подальшого удосконалення без повної перебудови модуля [33].

На рівні введення даних передбачається отримання відомостей про програмний проєкт. Джерелом таких даних може бути файл із підготовленою вибіркою, таблиця з ознаками, форма ручного введення або інший структурований формат подання інформації. У межах кваліфікаційної роботи найбільш практичним є використання табличного представлення даних, наприклад у форматі CSV, оскільки воно є простим у реалізації, добре підтримується засобами Python і безпосередньо узгоджується з бібліотеками аналізу даних [11, 18, 29, 30]. Основна функція цього компонента полягає в завантаженні, первинній перевірці та передаванні даних до наступного етапу обробки.

Рівень попередньої обробки виконує критично важливу роль, оскільки саме на ньому сирі вхідні дані перетворюються у форму, придатну для навчання моделі.

До функцій цього рівня належать очищення даних, усунення або заповнення пропусків, кодування категоріальних ознак, нормалізація або масштабування числових значень, а також формування підсумкової матриці ознак. Необхідність такого етапу зумовлена тим, що характеристики програмних проєктів мають різну природу та не можуть бути безпосередньо подані в модель без попереднього узгодження форматів [5, 11, 12, 18]. Крім того, саме на цьому рівні може виконуватися поділ вибірки на навчальну та тестову частини.

Рівень машинного навчання є центральним у структурі програмного модуля. Він відповідає за створення, навчання та використання моделі класифікації. У межах даної роботи як основний алгоритм обрано випадковий ліс, а тому відповідний компонент повинен реалізовувати ініціалізацію моделі, налаштування її параметрів, навчання на підготовленій вибірці та отримання прогнозів для нових об’єктів [2, 5, 18]. Якщо в системі передбачається порівняння кількох моделей, цей самий рівень може містити підмодулі для базових алгоритмів, наприклад логістичної регресії або дерева рішень. Такий підхід дозволяє реалізувати архітектуру, у якій модель машинного навчання є змінним компонентом, а не жорстко вбудованим елементом усієї системи.

Рівень керування результатами призначений для оброблення та збереження результатів роботи моделі. Він забезпечує виведення передбаченого класу програмного проєкту, обчислення основних метрик якості під час тестування, формування зведеної інформації про точність класифікації та, за потреби, збереження навченої моделі для подальшого повторного використання. Практичне значення цього рівня полягає в тому, що він поєднує математичний результат моделювання з прикладним сценарієм використання програмного модуля. Без цього компонента система залишалася б лише набором процедур навчання, а не завершеним прикладним рішенням [1, 5, 11, 18].

Рівень взаємодії з користувачем забезпечує подання вхідних даних і відображення результатів роботи програми. Він може бути реалізований у вигляді консольного інтерфейсу, форми введення або простого графічного інтерфейсу. Його головне призначення полягає в тому, щоб надати користувачеві можливість завантажити або ввести дані про проєкт, ініціювати процедуру класифікації та

отримати підсумковий результат у зрозумілому вигляді. При цьому архітектурно важливо, щоб інтерфейсний рівень не містив усієї бізнес-логіки, а лише взаємодіяв із відповідними внутрішніми компонентами системи [19, 20].

Загальна логіка архітектури програмного модуля може бути подана як послідовний ланцюг перетворення даних: введення даних → підготовка даних → навчання або завантаження моделі → класифікація → подання результату. Така послідовність є природною для систем машинного навчання, однак у межах архітектурного проєктування важливо не лише визначити етапи, а й формалізувати склад компонентів (таблиця 2.4).

Таблиця 2.4 – Основні компоненти архітектури програмного модуля

Компонент	Призначення	Основні функції
Модуль введення даних	Отримання відомостей про програмні проєкти	Завантаження файлів, перевірка структури, читання набору ознак
Модуль попередньої обробки	Підготовка даних до машинного навчання	Очищення, кодування, нормалізація, поділ вибірки
Модуль навчання моделі	Побудова класифікаційної моделі	Навчання випадкового лісу та базових моделей
Модуль класифікації	Визначення класу нового проєкту	Обробка вхідного запису, застосування навченої моделі
Модуль оцінювання	Аналіз якості роботи системи	Обчислення accuracy, precision, recall, F1-score
Модуль збереження результатів	Підтримка повторного використання системи	Збереження моделі, виведення результатів, формування звітів
Інтерфейс користувача	Взаємодія з системою	Введення даних, запуск класифікації, відображення результату

Як показано в таблиці 2.4, архітектура програмного модуля має чітко виражену компонентну структуру. Такий підхід є доцільним не лише з погляду програмної інженерії, а й з практичних міркувань. Якщо компонент попередньої обробки буде реалізовано окремо від компонента навчання моделі, це дасть змогу легко змінювати алгоритм класифікації без необхідності повного переписування програми. Аналогічно, винесення модуля оцінювання в окремий блок дозволяє використовувати його як під час експериментальної перевірки, так і в разі подальшого вдосконалення моделі.

У межах проєктування архітектури доцільно також визначити основні потоки даних між компонентами. Спочатку дані про програмні проєкти надходять у

систему через модуль введення. Після цього вони передаються до модуля попередньої обробки, де формуються уніфіковані ознакові вектори. Далі ці дані можуть бути використані у двох режимах: у режимі навчання вони надходять до модуля навчання моделі, а в режимі практичної класифікації - до модуля класифікації, який використовує вже навчений класифікатор. У будь-якому з цих випадків результати передаються до модуля оцінювання та модуля подання результатів. Такий розподіл потоків дозволяє відокремити етап створення моделі від етапу її прикладного використання.

Для відображення архітектури доцільно використати узагальнену структурну схему, як показано на рисунку 2.3.

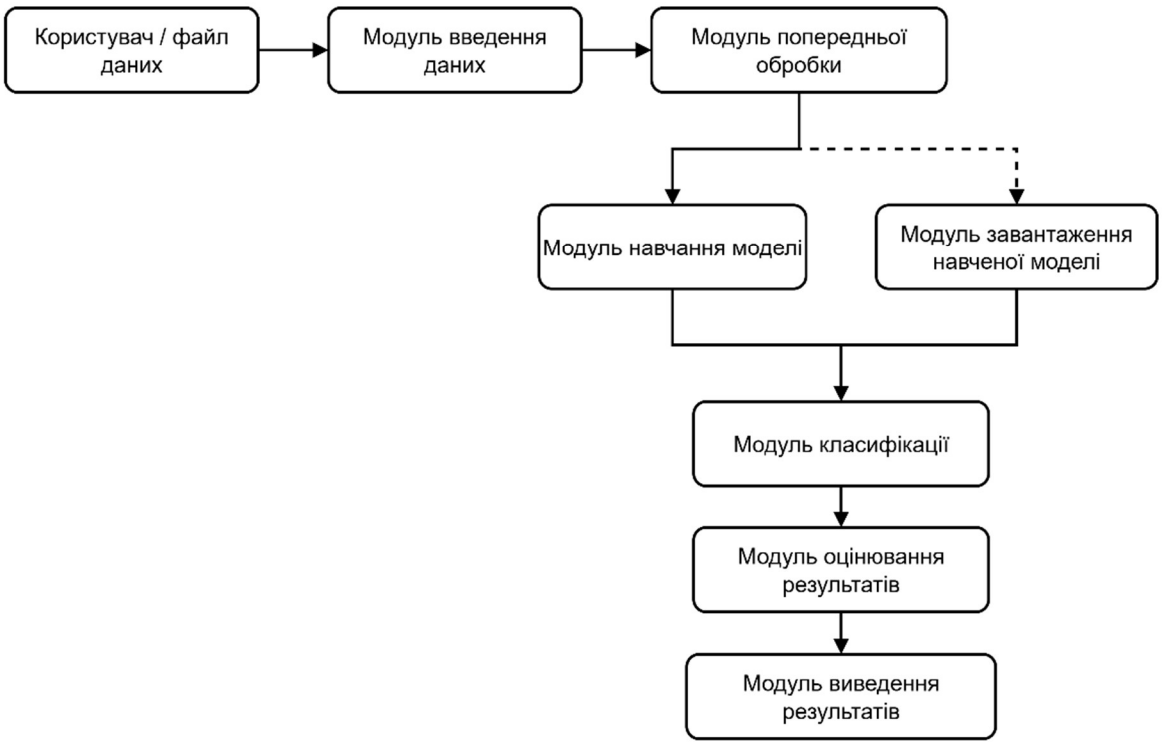

Рисунок 2.3 – Архітектура програмного модуля класифікації програмних проєктів [33]

На рисунку 2.3 відображено базову архітектурну логіку роботи системи. Така схема є достатньо простою для реалізації, але водночас охоплює всі основні процеси, необхідні для функціонування програмного модуля.

З точки зору програмної реалізації доцільно орієнтуватися на модульний підхід, у якому кожен архітектурний блок відповідає окремому програмному файлу, класу або групі функцій. Наприклад, можуть бути виділені такі логічні

модулі: data_loader, preprocessing, model_training, prediction, evaluation, ui. Такий підхід відповідає принципам розподілу відповідальності в програмній інженерії та спрощує подальше супроводження програмного коду [19, 20]. Крім того, він дозволяє тестувати та налагоджувати окремі частини системи незалежно одна від одної.

У контексті архітектурного проєктування доцільно також визначити основні сценарії функціонування системи. Перший сценарій пов'язаний із навчанням моделі. У цьому випадку користувач або розробник завантажує набір даних, виконує їх попередню обробку, запускає навчання моделі та отримує результати її тестування. Другий сценарій пов'язаний із класифікацією нового проєкту. У цьому режимі в систему подаються характеристики окремого програмного проєкту, після чого виконується їх перетворення за тими самими правилами, що були використані під час навчання, і система повертає передбачений клас. Таке розмежування сценаріїв дозволяє побудувати архітектуру, у якій дослідницький та прикладний режими використання не змішуються, але працюють на спільній основі.

З позицій надійності системи важливо, щоб архітектура враховувала можливість перевірки коректності вхідних даних. У реальних умовах помилки можуть виникати через пропуски значень, невідповідність формату, некоректні типи ознак або неповне введення характеристик нового проєкту. Тому доцільно передбачити в модулі введення та попередньої обробки процедури валідації, які запобігатимуть передаванню некоректних даних до моделі машинного навчання. Хоча це ускладнює реалізацію, така вимога є виправданою, оскільки підвищує практичну придатність програмного засобу.

Окремої уваги потребує питання збереження навченої моделі. Якщо щоразу виконувати повне перенавчання перед класифікацією нового проєкту, це зменшить ефективність роботи системи. Тому архітектура повинна передбачати механізм збереження моделі після завершення навчання та її повторного завантаження під час експлуатації. Для цього можуть бути використані стандартні засоби серіалізації, що підтримуються мовою Python та бібліотеками машинного навчання [5, 18]. Завдяки цьому модуль може працювати у двох незалежних режимах: режимі створення моделі та режимі її застосування.

Узагальнення логіки взаємодії основних компонентів доцільно подати у вигляді таблиці сценаріїв (таблиця 2.5).

Таблиця 2.5 – Основні сценарії функціонування програмного модуля

Сценарій	Вхідні дані	Основні дії	Результат
Навчання моделі	Набір даних про проєкти з відомими класами	Завантаження, попередня обробка, поділ вибірки, навчання, тестування	Навчена модель та метрики якості
Класифікація нового проєкту	Ознаки одного нового проєкту	Перетворення даних, завантаження моделі, прогнозування	Передбачений клас проєкту
Оцінювання моделі	Тестова вибірка	Обчислення метрик та аналіз результатів	Accuracy, precision, recall, F1-score
Збереження результатів	Навчена модель та вихідні дані	Серіалізація моделі, формування звіту	Файл моделі та підсумкові результати

Як видно з таблиці 2.5, архітектура повинна підтримувати не лише один лінійний процес, а кілька пов’язаних режимів функціонування. Це ще раз підтверджує доцільність модульної організації системи.

З погляду загальної логіки побудови програмного забезпечення обрана архітектура може бути охарактеризована як функціонально-модульна з елементами шарового підходу. Її верхній рівень представлений засобами взаємодії з користувачем, середній - логікою підготовки даних і використання моделі, а нижній - інструментами роботи з даними та збереження результатів. Такий підхід є достатньо простим для реалізації в навчальному проєкті й водночас забезпечує структурованість рішення, що відповідає принципам проєктування якісного програмного забезпечення [19, 20].

Отже, проєктування архітектури програмного модуля класифікації програмних проєктів дало змогу визначити основні компоненти системи, їх функціональне призначення та логіку взаємодії. Запропонована архітектура охоплює всі ключові етапи оброблення даних: введення, попередню підготовку, навчання та використання моделі, оцінювання результатів і взаємодію з користувачем.

2.4 Опис алгоритму класифікації та сценарію функціонування системи

Алгоритм класифікації визначає послідовність операцій, які забезпечують перехід від початкових відомостей про програмний проєкт до отримання підсумкового результату у вигляді належності до певного класу. У межах даної роботи алгоритм має враховувати як особливості підготовки даних, так і етапи навчання, тестування та практичного використання моделі.

У межах програмного модуля доцільно розрізняти два основні режими функціонування системи: режим навчання моделі та режим класифікації нового програмного проєкту. У першому випадку система працює з набором даних, у якому для кожного проєкту вже відомий клас. Основна мета цього режиму полягає у виявленні закономірностей між ознаками та цільовою змінною, оцінюванні якості моделі та збереженні навченої конфігурації для подальшого використання. У другому випадку модель уже вважається побудованою, а система застосовує її для визначення класу нового об’єкта. Таке розмежування є принципово важливим, оскільки воно дозволяє чітко відокремити дослідницький етап від етапу практичного використання програмного модуля.

Алгоритм у режимі навчання починається з отримання початкового набору даних про програмні проєкти. Як було зазначено в попередніх підрозділах, найбільш зручним є табличний формат представлення, у якому кожний рядок відповідає окремому проєкту, а кожний стовпець - певній ознаці. На цьому етапі система повинна виконати перевірку цілісності структури даних, зокрема проконтролювати наявність усіх обов’язкових полів, коректність типів значень та узгодженість назви цільової змінної. Така перевірка є необхідною, оскільки помилки у вхідному файлі можуть призвести до неможливості побудови моделі або до некоректного навчання.

Після завантаження набору даних виконується етап попередньої обробки. На цьому етапі система очищає дані, обробляє пропущені значення, кодує категоріальні ознаки у числовий формат, а також за потреби здійснює масштабування кількісних характеристик. Хоча обраний у роботі метод випадкового лісу не є надмірно чутливим до масштабу числових ознак, у разі

порівняння з іншими алгоритмами така процедура є доцільною [2, 5, 11]. Попередня обробка забезпечує формування уніфікованого простору ознак, у якому всі проєкти подані в одному й тому самому форматі. Крім того, саме на цьому етапі можуть бути усунені дублікати записів або відкинуті ознаки, що не несуть корисної інформації для класифікації.

Далі формується матриця ознак і вектор цільових значень. Матриця ознак містить усі параметри, за якими система аналізуватиме програмні проєкти, а вектор цільових значень – класи, які відповідають кожному запису. Саме ця структура є безпосереднім входом для алгоритму машинного навчання. На наступному кроці виконується розподіл даних на навчальну та тестову вибірки. Навчальна частина використовується для побудови моделі, тоді як тестова – для перевірки її здатності узагальнювати закономірності на нових, раніше не використаних прикладах. Такий підхід є стандартним у задачах класифікації та забезпечує більш об’єктивне оцінювання якості системи [5, 11, 18].

Після підготовки вибірок запускається процедура навчання моделі. У межах цієї роботи основним алгоритмом є випадковий ліс, тому система створює екземпляр відповідної моделі та навчає його на підготовлених навчальних даних [2]. На цьому етапі модель аналізує вхідні ознаки, формує ансамбль дерев рішень і визначає внутрішні закономірності, що дозволяють надалі відносити новий проєкт до певного класу. Важливо, що навчання виконується лише на навчальній вибірці, тоді як тестові записи на цьому етапі не використовуються, що дає змогу зберегти об’єктивність перевірки.

Після завершення навчання виконується тестування моделі. Для цього система подає на вхід алгоритму тестову вибірку без інформації про правильний клас і отримує прогнозовані результати. Далі прогнозовані класи порівнюються з фактичними значеннями, на підставі чого обчислюються стандартні метрики якості: accuracy, precision, recall та F1-score [1, 5, 11, 18]. Окрім числових метрик, за потреби може бути сформована матриця помилок, що дає змогу детальніше проаналізувати, між якими саме класами модель найчастіше припускається неточностей. З огляду на прикладний характер задачі, оцінювання моделі є обов’язковою складовою алгоритму, оскільки воно дозволяє зробити висновок про

придатність навченої системи до практичного використання.

Якщо результати навчання визнаються задовільними, система зберігає модель разом із параметрами попередньої обробки даних. Така операція є важливою, оскільки для класифікації нових проєктів повинні використовуватися ті самі правила кодування, нормалізації та відбору ознак, що і під час навчання. Якщо зберегти лише модель без параметрів перетворення даних, подальше практичне застосування системи може стати некоректним. Отже, у структурі алгоритму збереження результатів навчання має розглядатися як завершальний етап режиму побудови моделі.

Режим класифікації нового програмного проєкту має дещо іншу логіку, хоча в його основі лежать ті самі принципи. Спочатку користувач подає в систему характеристики нового проєкту. Це може бути або ручне введення ознак, або зчитування даних із підготовленого файлу. Далі система перевіряє, чи відповідає структура отриманих даних очікуваному формату. Особливого значення тут набуває узгодженість переліку ознак із тією структурою, яка використовувалася під час навчання моделі. Якщо якісь поля відсутні або подані у некоректному вигляді, система повинна або повідомити про помилку, або застосувати заздалегідь визначені правила обробки пропусків.

Після перевірки коректності вхідні дані проходять ту саму процедуру попередньої обробки, що й навчальна вибірка. Зокрема, категоріальні ознаки кодуються за раніше збереженими правилами, кількісні параметри приводяться до потрібного формату, а структура запису трансформується у вектор ознак, придатний для подання в модель машинного навчання. Принципово важливо, що на цьому етапі не створюються нові правила кодування або перетворення; натомість використовуються ті, що були сформовані раніше під час навчання системи. Саме це забезпечує єдність логіки функціонування програмного модуля.

Наступним кроком є завантаження навченої моделі. Після цього підготовлений вектор ознак передається в алгоритм класифікації, який повертає прогнозований клас проєкту. З технічного погляду це означає, що ансамбль дерев рішень, сформований на попередньому етапі, аналізує новий запис і визначає, до якої категорії він найбільше належить. У разі потреби система може також вивести

допоміжну інформацію, наприклад ймовірності належності до кількох класів або перелік найбільш впливових ознак. Однак навіть базовий сценарій, у межах якого користувач отримує один підсумковий клас, уже є достатнім для виконання поставленої задачі.

Для забезпечення зрозумілості функціонування програмного модуля алгоритм доцільно подати у вигляді узагальненої послідовності кроків (таблиця 2.6).

Таблиця 2.6 – Узагальнений алгоритм роботи програмного модуля

№ етапу	Зміст етапу	Результат
1	Завантаження набору даних або введення ознак проєкту	Отримано вхідні дані
2	Перевірка структури та коректності вхідних даних	Підтверджено придатність даних до обробки
3	Попередня обробка даних	Сформовано уніфіковану матрицю ознак
4	Поділ на навчальну і тестову вибірки або підготовка нового запису	Дані підготовлено до використання
5	Навчання моделі або завантаження раніше навченої моделі	Модель готова до прогнозування
6	Класифікація тестових або нових записів	Отримано прогнозовані класи
7	Оцінювання якості моделі або виведення результату	Сформовано метрики чи підсумковий клас
8	Збереження моделі та результатів	Підготовлено дані до подальшого використання

Як видно з таблиці 2.6, алгоритм має чітко визначену логіку та може бути реалізований у вигляді послідовного виклику окремих програмних функцій. Такий підхід добре узгоджується з модульною архітектурою, розглянутою в попередньому підрозділі.

Окремого розгляду потребує сценарій функціонування системи з погляду користувача. У прикладному аспекті користувач взаємодіє не з окремими алгоритмічними блоками, а з цілісним програмним модулем. Саме тому доцільно описати не лише внутрішню логіку роботи алгоритму, а й зовнішній сценарій використання системи. У спрощеному варіанті такий сценарій передбачає, що користувач запускає програму, обирає режим роботи, завантажує файл з даними або вводить характеристики проєкту, ініціює виконання класифікації та отримує результат. Якщо система працює в режимі навчання, користувач додатково

отримує метрики якості моделі. Якщо ж обрано режим класифікації нового проєкту, користувач бачить лише підсумкову категорію, до якої віднесено проєкт.

Логіку сценарію функціонування системи доцільно узагальнити в таблиці 2.7.

Таблиця 2.7 – Сценарії функціонування системи

Сценарій	Дії користувача	Дії системи	Результат
Навчання моделі	Завантажує набір даних, запускає навчання	Перевіряє дані, виконує обробку, навчає модель, оцінює її	Навчена модель та метрики якості
Класифікація нового проєкту	Вводить або завантажує ознаки нового проєкту	Обробляє дані, завантажує модель, виконує прогнозування	Визначений клас проєкту
Аналіз результатів	Переглядає підсумкові показники або прогноз	Формує результати у зручному вигляді	Інформація для подальшого використання

Важливим аспектом є також обробка помилкових ситуацій. У реальному функціонуванні системи можуть траплятися випадки, коли користувач завантажує файл з неповною структурою, подає некоректні значення ознак або намагається класифікувати новий проєкт без попереднього навчання моделі. Тому алгоритм повинен включати елементи контролю виняткових ситуацій. Наприклад, якщо модель ще не навчена, система повинна вивести відповідне повідомлення та заблокувати перехід до етапу прогнозування. Якщо ж вхідний запис містить пропуски у ключових полях, модуль має або автоматично заповнити їх за встановленими правилами, або попередити користувача про неможливість виконання класифікації.

Оскільки розроблюваний програмний модуль має прикладне спрямування, важливою вимогою до алгоритму є його відтворюваність. Це означає, що за однакових вхідних даних і однакових параметрів моделі система повинна формувати однакові результати. Для забезпечення цього необхідно фіксувати параметри поділу вибірки, конфігурацію алгоритму випадкового лісу та правила перетворення ознак. Такий підхід є не лише технічно доцільним, а й важливим з науково-методичної точки зору, оскільки він підвищує обґрунтованість результатів, отриманих у межах кваліфікаційної роботи [2, 5, 11, 18].

У межах алгоритму доцільно також передбачити можливість розширення. Зокрема, у подальшому система може бути доповнена іншими класифікаторами, механізмами автоматичного добору параметрів, засобами візуалізації значущості ознак або функціями імпорту даних безпосередньо з програмних репозиторіїв. Той факт, що базовий алгоритм уже побудовано як послідовність відносно незалежних етапів, створює передумови для такого вдосконалення. Саме тому формалізація алгоритму на поточному етапі має не лише описове, а й проєктне значення.

Для візуального відображення доцільно представити загальну схему алгоритму функціонування системи (рисунок 2.4).

Рисунок 2.4 – Схема алгоритму класифікації програмних проєктів

На рисунку 2.4 подано узагальнену логіку функціонування системи, яка відображає як внутрішню алгоритмічну послідовність, так і сценарій прикладного використання програмного модуля.

Отже, алгоритм класифікації програмних проєктів у межах даної роботи

базується на послідовному виконанні процедур завантаження даних, їх попередньої підготовки, навчання або завантаження моделі, прогнозування результату та оцінювання ефективності. Важливо, що цей алгоритм підтримує два режими роботи – навчальний і прикладний, що дозволяє використовувати програмний модуль як для експериментального дослідження, так і для безпосередньої класифікації нових проєктів.

3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОГРАМНОГО МОДУЛЯ КЛАСИФІКАЦІЇ ПРОГРАМНИХ ПРОЄКТІВ

3.1 Програмна реалізація модуля

Основне призначення розробленого модуля полягає в автоматизації процесу підготовки даних, навчання моделі машинного навчання, класифікації нових програмних проєктів та оцінювання якості отриманих результатів. З огляду на поставлені завдання, реалізація повинна забезпечувати не лише коректність математичної обробки, а й логічну цілісність усього програмного рішення [5, 11, 18, 29-32].

Для реалізації модуля доцільно використати мову програмування Python, оскільки вона є однією з найбільш поширених у задачах аналізу даних і машинного навчання. Її перевагами є наявність зрозумілого синтаксису, розвиненого набору бібліотек для оброблення табличних даних, реалізації алгоритмів класифікації, оцінювання моделей та візуалізації результатів. Крім того, Python добре підходить для створення навчально-дослідницьких програмних засобів, оскільки дозволяє досить швидко поєднати етапи підготовки даних, побудови моделі та розроблення прикладного інтерфейсу [5, 18, 31].

У межах даної роботи також використано бібліотеку pandas для зчитування та оброблення даних, scikit-learn для побудови класифікаційної моделі, numpy для роботи з масивами числових значень, а також joblib або аналогічних засобів для збереження навченої моделі. Такий набір інструментів є достатнім для реалізації всіх основних функцій програмного модуля: завантаження набору даних, попередньої обробки ознак, навчання випадкового лісу, тестування моделі та використання її для прогнозування класу нового програмного проєкту [5, 11, 18].

З урахуванням архітектури, розглянутої в підрозділі 2.3, програмна реалізація була побудована за модульним принципом. Це означає, що функціональність системи логічно поділяється на кілька окремих частин: модуль роботи з даними, модуль попередньої обробки, модуль навчання моделі, модуль прогнозування, модуль оцінювання та модуль взаємодії з користувачем. Такий підхід дає змогу спростити структуру програмного коду, підвищити його читабельність і полегшити

подальше вдосконалення системи [19, 20]. Крім того, модульна організація є доцільною з позицій програмної інженерії, оскільки забезпечує розподіл відповідальності між окремими частинами програми.

На першому етапі реалізації створюється компонент завантаження даних. Його функція полягає в зчитуванні файлу з інформацією про програмні проєкти, перевірці наявності необхідних полів та формуванні робочої таблиці, що надалі передається в інші компоненти системи. Оскільки в роботі використовується табличне представлення даних, джерелом інформації доцільно обрати файл у форматі CSV. Такий формат є універсальним, легко опрацьовується в Python і зручний для формування навчального набору даних. На цьому ж етапі доцільно здійснювати початкову перевірку структури: чи присутні потрібні стовпці, чи збігаються їхні назви з очікуваним шаблоном, чи немає критичних порушень формату.

Після завантаження даних виконується попередня обробка. У межах програмної реалізації цей блок має особливо важливе значення, оскільки саме він забезпечує придатність сирих вхідних даних до використання в алгоритмі машинного навчання. До його функцій належать заповнення пропусків, кодування категоріальних ознак, формування числового представлення даних, а також поділ таблиці на матрицю ознак і цільову змінну. Якщо деякі атрибути проєктів подано у текстовому вигляді, система перетворює їх у числові значення, використовуючи відповідні засоби кодування. У разі, коли для окремих ознак існують пропуски, можуть застосовуватися базові стратегії заповнення, наприклад використання найчастішого значення для категоріальних параметрів або медіанного значення для кількісних [5, 11, 18].

Важливим елементом програмної реалізації є забезпечення узгодженості між етапом навчання та етапом використання моделі. Це означає, що всі перетворення, виконані над навчальними даними, повинні в тому самому вигляді застосовуватися й до нових проєктів, які надходитимуть на класифікацію. Якщо, наприклад, під час навчання категоріальна ознака платформи перетворювалася у набір двійкових змінних, то такий самий механізм повинен бути використаний і для нового запису. Саме тому в реалізації доцільно передбачити збереження не лише самої моделі, а й

параметрів попередньої обробки.

Наступним програмним компонентом є модуль навчання класифікаційної моделі. У межах даної роботи основним алгоритмом обрано випадковий ліс, тому реалізація цього блоку передбачає створення об’єкта RandomForestClassifier, налаштування його базових параметрів та запуск процедури навчання на підготовлених даних [2, 18]. Для забезпечення відтворюваності доцільно задавати фіксоване значення random_state. Навчання виконується на частині вибірки, тоді як інша частина резервується для тестування. Такий підхід дозволяє не лише побудувати модель, а й одразу оцінити її ефективність.

У програмній реалізації доцільно також передбачити можливість використання кількох алгоритмів для порівняння. Наприклад, поряд із випадковим лісом можуть бути реалізовані логістична регресія та дерево рішень як базові моделі. Це дає змогу під час експериментів зіставити результати кількох підходів і додатково обґрунтувати вибір основного алгоритму. Однак основна бізнес-логіка модуля повинна орієнтуватися саме на роботу з тим методом, який був визнаний найбільш доцільним у попередньому розділі, тобто з випадковим лісом.

Після завершення навчання модель передається до модуля оцінювання. Основна функція цього компонента полягає у визначенні якості класифікації на тестовій вибірці. У реалізації передбачено обчислення основних метрик: accuracy, precision, recall та F1-score. Такі показники дозволяють комплексно оцінити роботу системи та зробити висновок щодо її придатності до практичного використання [1, 5, 11, 18].

Після оцінювання модель і допоміжні параметри доцільно зберігати у файл. Для цього можуть бути використані засоби серіалізації, зокрема joblib, що дозволяють швидко записати на диск навчений класифікатор та надалі завантажувати його без повторного навчання [18]. Такий підхід забезпечує розмежування між режимом навчання та режимом експлуатації. Один раз навчена модель може надалі використовуватися багаторазово для класифікації нових програмних проєктів, що підвищує зручність роботи з програмним модулем.

Окремим блоком реалізації є модуль прогнозування. Його завданням є приймання характеристик нового програмного проєкту, перетворення цих даних у

формат, сумісний із навченою моделлю, завантаження класифікатора та повернення передбаченого класу. У практичному аспекті це означає, що користувач може або ввести параметри вручну, або завантажити файл з одним або кількома новими записами, після чого система автоматично виконає всі необхідні перетворення та повідомить результат класифікації. Саме цей блок перетворює навчальну модель на функціональний прикладний інструмент.

З погляду організації програмного коду доцільно подати внутрішню структуру модуля у вигляді окремих класів або функціональних підсистем (таблиця 3.1).

Таблиця 3.1 – Основні програмні компоненти модуля

Компонент	Призначення	Основні засоби реалізації
Завантажувач даних	Зчитування та перевірка набору даних	`pandas`, робота з CSV
Блок попередньої обробки	Кодування, очищення, підготовка ознак	`pandas`, `numpy`, засоби `scikit-learn`
Блок навчання моделі	Побудова та навчання класифікатора	`RandomForestClassifier`, інші моделі
Блок оцінювання	Обчислення метрик якості	`accuracy_score`, `precision_score`, `recall_score`, `f1_score`
Блок збереження моделі	Серіалізація та повторне використання	`joblib`
Блок прогнозування	Класифікація нового проєкту	Завантаження моделі та обробка нового запису
Інтерфейс взаємодії	Запуск сценаріїв роботи програми	Консольний або спрощений графічний інтерфейс

Як видно з таблиці 3.1, реалізація модуля не потребує надмірно складної технологічної інфраструктури, однак вимагає чіткого розмежування функцій між компонентами. Саме така організація коду дає змогу забезпечити зрозумілу логіку функціонування системи та полегшити її документування в межах кваліфікаційної роботи. Для наочності загальну логіку програмної реалізації подано у вигляді схеми (рисунок 3.1).

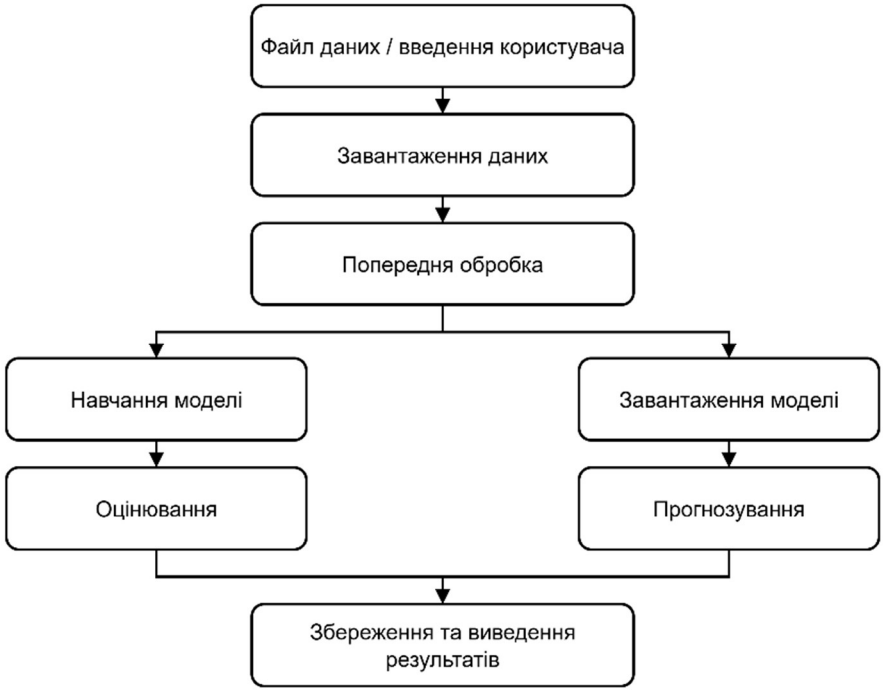

Рисунок 3.1 – Структура програмної реалізації модуля класифікації програмних проєктів

На рисунку 3.1 представлено спрощене відображення програмної логіки системи. Схема показує, що реалізація будується навколо двох основних режимів: режиму створення моделі та режиму її застосування.

З огляду на академічний формат роботи повні програмні коди доцільно виносити в додатки, тоді як в основному тексті слід подати лише ключові фрагменти, що ілюструють принцип реалізації. Такий підхід є виправданим, оскільки дозволяє зберегти аналітичний характер викладу, не перевантажуючи основну частину надмірними технічними деталями. Нижче наведено приклади базових фрагментів програмної реалізації.

Лістинг 3.1 – Завантаження набору даних

```
import pandas as pd

def load_dataset(path: str) -> pd.DataFrame:
    data = pd.read_csv(path)
    return data
```

У лістингу 3.1 показано базову функцію зчитування набору даних із CSV-файлу. На практиці така функція може бути доповнена перевіркою наявності обов’язкових стовпців, обробкою винятків і валідацією формату вхідного файлу.

Лістинг 3.2 – Формування матриці ознак і цільової змінної

```
def split_features_target(data: pd.DataFrame, target_column: str):
    X = data.drop(columns=[target_column])
    y = data[target_column]
    return X, y
```

Фрагмент, наведений у лістингу 3.2, демонструє відокремлення ознак від цільової змінної. Саме така структура далі використовується для навчання та тестування моделі класифікації.

Лістинг 3.3 – Навчання моделі випадкового лісу

```
from sklearn.ensemble import RandomForestClassifier

def train_model(X_train, y_train):
    model = RandomForestClassifier(
        n_estimators=100,
        random_state=42
    )
    model.fit(X_train, y_train)
    return model
```

У лістингу 3.3 подано приклад створення та навчання класифікатора випадкового лісу. Використання параметра random_state забезпечує відтворюваність результатів, що є важливим для науково-прикладного дослідження.

Лістинг 3.4 – Оцінювання якості моделі

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    return {
        "accuracy": accuracy_score(y_test, y_pred),
        "precision": precision_score(y_test, y_pred, average="weighted"),
        "recall": recall_score(y_test, y_pred, average="weighted"),
        "f1_score": f1_score(y_test, y_pred, average="weighted")
    }
```

Лістинг 3.4 ілюструє обчислення базових метрик якості класифікації. Саме ці показники надалі використовуються для аналізу ефективності реалізованого програмного модуля.

Лістинг 3.5 – Збереження навченої моделі

```
import joblib

def save_model(model, filename: str):
    joblib.dump(model, filename)
```

Поданий фрагмент демонструє серіалізацію навченої моделі. Завдяки цьому програмний модуль може працювати в окремому режимі прогнозування без необхідності повторного навчання.

Окрім внутрішньої логіки роботи системи, реалізація повинна враховувати питання зручності використання. Для кваліфікаційної роботи доцільно обрати простий інтерфейс взаємодії, наприклад консольний режим, у якому користувач обирає одну з доступних дій: навчання моделі, оцінювання або класифікацію нового проєкту. Такий підхід дозволяє зосередитися на реалізації основної функціональності без істотного ускладнення програмного коду. Водночас структура системи може бути легко адаптована до графічного або веб-інтерфейсу в разі подальшого розвитку рішення.

У процесі реалізації доцільно також забезпечити базову обробку помилок. Наприклад, якщо файл із даними не знайдено або має неправильну структуру, програма повинна вивести змістовне повідомлення. Якщо користувач намагається виконати прогнозування без наявності збереженої моделі, система також має попередити про необхідність попереднього навчання. Такі механізми хоч і не є центральною частиною математичної логіки, проте суттєво підвищують завершеність програмного рішення.

Для узагальнення послідовності роботи програмного модуля розглянемо типовий сценарій його функціонування (таблиця 3.2).

Таблиця 3.2 – Типовий сценарій роботи програмного модуля

Крок	Дія	Результат
1	Завантаження набору даних	Отримано вихідну таблицю проєктів
2	Попередня обробка даних	Сформовано набір придатних до класифікації ознак
3	Навчання моделі	Побудовано класифікатор
4	Тестування моделі	Обчислено показники якості
5	Збереження моделі	Підготовлено модель до повторного використання
6	Завантаження нового запису	Отримано дані для класифікації
7	Прогнозування класу	Визначено клас нового проєкту
8	Виведення результату	Користувач отримує підсумкове рішення

У додатку А наведено основні програмні лістинги розробленого модуля класифікації програмних проєктів на основі методів машинного навчання.

Таким чином, програмна реалізація модуля класифікації програмних проєктів базується на модульному поєднанні засобів завантаження даних, їх попередньої підготовки, навчання випадкового лісу, оцінювання якості та практичного використання моделі для класифікації нових записів. Обрана технологічна основа у вигляді Python та бібліотек pandas, scikit-learn і joblib забезпечує достатню гнучкість, простоту реалізації та відповідність поставленій задачі.

3.2 Підготовка набору даних і навчання моделей

Ефективність програмного модуля класифікації програмних проєктів значною мірою визначається якістю підготовки набору даних, оскільки саме дані формують основу для побудови, навчання та перевірки моделі машинного навчання. Навіть за умови коректного вибору алгоритму недостатньо впорядкований, неповний або слабо структурований набір ознак може суттєво знизити точність класифікації. Саме тому підготовка даних у межах даної роботи розглядається не як допоміжна технічна процедура, а як один із ключових етапів експериментального дослідження.

У межах поставленої задачі набір даних повинен відображати програмні проєкти як об’єкти класифікації та містити достатню кількість характеристик, що дозволяють моделі виявляти відмінності між окремими класами. З огляду на обрану постановку задачі, у якій проєкти класифікуються за типом програмного продукту, доцільно сформувати набір даних, що охоплює кілька категорій проєктів, наприклад web, mobile, desktop, embedded та enterprise. Кожен запис у такому наборі повинен містити значення атрибутів, визначених у підрозділі 2.1: платформу, основну мову програмування, тип архітектури, кількість модулів, число файлів, обсяг кодової бази, кількість залежностей, наявність тестів, наявність CI/CD, кількість учасників, кількість комітів, тривалість активної розробки та цільовий клас [6, 21, 24].

На практичному рівні підготовка набору даних починається з формування вихідної таблиці. Джерелом можуть бути або попередньо зібрані приклади реальних програмних проєктів, або експериментально сформований датасет, у якому відтворюються типові характеристики різних категорій програмних систем. У межах роботи формовано структурованого навчального набору на основі логічно узгоджених описів програмних проєктів, якщо для кожного запису визначено реалістичні параметри та забезпечено достатню репрезентативність класів. У такому випадку особливого значення набуває не лише кількість записів, а й збалансованість набору, тобто приблизно рівномірне представлення різних класів.

Після формування вихідної таблиці необхідно перевірити її структуру. На цьому етапі з'ясовується, чи всі записи містять значення для обов'язкових полів, чи немає дублювання рядків, а також чи узгоджені формати подання ознак. Наприклад, якщо одна частина записів містить тривалість розробки у місяцях, а інша – у роках, такі значення повинні бути приведені до спільної шкали. Аналогічно, якщо архітектурний стиль подається різними позначеннями, наприклад "microservice", "microservices" і "micro-service", необхідно виконати їх уніфікацію. Подібні операції є важливими, оскільки навіть формально незначні розбіжності можуть спотворювати структуру ознак і погіршувати результати навчання [7, 11, 18].

Наступним етапом є попереднє очищення даних. Воно включає видалення дублікатів, перевірку наявності пропусків, пошук аномальних значень та первинний аналіз розподілу числових параметрів. Якщо в окремих записах відсутні значення деяких ознак, можуть бути застосовані стандартні підходи до їх обробки: заповнення медіаною або середнім значенням для кількісних параметрів і найчастішим значенням для категоріальних. У випадку, коли запис містить критично неповну інформацію, його доцільно вилучити з набору. Важливо, щоб на цьому етапі рішення приймалися послідовно та однаково для всього набору, оскільки від цього залежить відтворюваність експерименту [5, 11, 18].

Оскільки сформований набір даних включає змішані типи ознак, суттєвого значення набуває кодування категоріальних параметрів. Такі характеристики, як платформа, основна мова програмування або тип архітектури, не можуть бути

безпосередньо подані в алгоритм машинного навчання у вигляді текстових значень. Для їх використання необхідно виконати перетворення в числовий формат. У роботі доцільно застосувати one-hot encoding або інший аналогічний механізм, який створює окремі бінарні поля для кожної категорії. Це дозволяє уникнути хибного порядкового тлумачення категорій і водночас зберегти їх змістовну відмінність.

Кількісні ознаки також потребують попереднього аналізу. Хоча випадковий ліс загалом не вимагає жорсткої нормалізації, перевірка масштабів значень залишається доцільною, особливо якщо в експерименті використовуються і базові моделі, чутливі до відстаней або масштабу ознак. Наприклад, число файлів у проєкті може вимірюватися сотнями, тоді як кількість учасників команди – одиницями. За таких умов для окремих алгоритмів доцільно застосовувати стандартне масштабування або нормалізацію. Водночас для основної моделі випадкового лісу це не є критично необхідним, тому структура підготовки даних може бути організована так, щоб масштабування використовувалося лише в разі застосування альтернативних алгоритмів [2, 5, 11].

Після очищення та перетворення даних формується фінальна матриця ознак. На цьому етапі кожен рядок уже відповідає окремому програмному проєкту, а всі стовпці мають числовий формат, придатний для подання у моделі машинного навчання. Одночасно формується вектор цільової змінної, який містить правильні класи проєктів. Саме ці дві структури – матриця ознак і вектор класів – стають безпосередньою основою для навчання класифікатора.

Для забезпечення об’єктивності експерименту доцільно виконати поділ набору даних на навчальну та тестову вибірки. Зазвичай більша частина даних використовується для навчання, а менша – для перевірки якості моделі. У межах даної роботи раціональним є співвідношення 80/20, за якого 80 % записів застосовується для навчання, а 20 % – для тестування. Такий поділ дозволяє зберегти достатній обсяг навчальної інформації та водночас забезпечити незалежну оцінку результатів на нових прикладах [5, 11, 18]. Для уникнення випадкових коливань доцільно фіксувати параметр random_state, що забезпечує відтворюваність вибірки.

На рисунку 3.2 подано загальну схему підготовки набору даних до навчання моделей.

Рисунок 3.2 – Послідовність підготовки набору даних до навчання моделей

На рисунку 3.2 відображено, що підготовка даних є багатокроковим процесом, у межах якого початковий опис програмних проєктів перетворюється у формалізований набір ознак, придатний для машинного навчання.

Після завершення підготовки даних виконується навчання моделей. Як було обґрунтовано в підрозділі 2.2, у роботі доцільно використати кілька алгоритмів: базові моделі для порівняння та основний ансамблевий метод. У ролі базових моделей можуть виступати логістична регресія та дерево рішень, тоді як основним робочим алгоритмом є випадковий ліс. Такий підхід дозволяє не лише отримати кінцевий результат, а й провести порівняльний аналіз, що підвищує аргументованість зроблених висновків.

Навчання логістичної регресії є доцільним як початковий орієнтир. Ця модель дозволяє оцінити, наскільки добре класи проєктів можуть бути розділені на основі відносно простих лінійних залежностей. Якщо результат виявиться низьким,

це підтверджуватиме необхідність використання складнішого нелінійного підходу. Навчання дерева рішень, своєю чергою, дає змогу перевірити, чи достатньо для задачі побудови ієрархічних правил класифікації. Порівняння цих моделей із випадковим лісом дозволяє побачити, наскільки ансамблевий підхід перевищує простіші алгоритми в умовах реального набору ознак.

Основний етап експерименту пов’язаний із навчанням випадкового лісу. Для цього формується модель типу RandomForestClassifier із фіксованим набором базових параметрів, наприклад кількістю дерев, обмеженням глибини або критерієм поділу вузлів. У початковій реалізації доцільно застосувати стандартні параметри бібліотеки scikit-learn або обрати помірні значення, які забезпечують стабільне навчання без надмірного ускладнення експерименту [18]. Якщо буде потрібно, у подальшому ці параметри можуть бути скориговані, однак у межах кваліфікаційної роботи базова конфігурація є цілком достатньою.

Після навчання кожної моделі виконується тестування на відкладеній вибірці. Отримані прогнози порівнюються з фактичними класами, після чого обчислюються показники accuracy, precision, recall та F1-score. Таке оцінювання дозволяє порівняти якість кількох алгоритмів за єдиними критеріями. Якщо основна модель демонструє найкращі результати, це додатково підтверджує коректність її вибору. Якщо ж відмінності між моделями виявляться незначними, це може свідчити про те, що набір ознак є достатньо добре структурованим і навіть простіші алгоритми здатні забезпечити прийнятний рівень класифікації.

Для наочності результати навчання кількох моделей доцільно подавати у вигляді таблиці 3.3.

Таблиця 3.3 – Узагальнена схема експериментального порівняння моделей

Модель	Призначення в експерименті	Очікувана роль
Логістична регресія	Базове порівняння	Оцінка можливості лінійного розділення класів
Дерево рішень	Інтерпретована нелінійна модель	Формування зрозумілих правил класифікації
Випадковий ліс	Основна модель	Досягнення найкращої точності та стійкості

Як показано в таблиці 3.3, кожна модель у межах експерименту виконує

окрему дослідницьку функцію. У результаті основний акцент робиться не лише на абсолютному значенні метрик, а й на порівняльному аналізі підходів.

У процесі підготовки даних і навчання моделей важливо також враховувати проблему можливого дисбалансу класів. Якщо певний тип програмних проєктів представлений значно ширше, ніж інші, модель може демонструвати зміщення у бік найчисельнішої категорії [17, 21]. Для зменшення такого ризику необхідно або на етапі формування датасету підтримувати приблизну рівновагу між класами, або використовувати відповідні параметри алгоритму чи метрики, чутливі до нерівномірного розподілу. У даній роботі найбільш доцільним є перший підхід, тобто формування максимально збалансованого набору даних.

Суттєве значення має і відтворюваність експерименту. Для цього повинні бути зафіксовані: склад набору даних, порядок його підготовки, спосіб кодування ознак, параметри поділу на вибірки та конфігурація моделей. Такий підхід дозволяє при повторному запуску програми одержати співставні результати, що є важливою вимогою до науково-прикладного дослідження. Крім того, це полегшує подальше розширення системи, оскільки всі етапи побудови моделі є формалізованими й можуть бути повторені без зміни загальної логіки.

Узагальнюючи, підготовка набору даних і навчання моделей у межах даної роботи здійснюються як послідовний процес, що включає формування структурованого датасету, перевірку та очищення записів, кодування ознак, поділ на навчальну і тестову вибірки, а також побудову кількох класифікаційних моделей із подальшим порівнянням їх ефективності.

3.3 Оцінювання результатів класифікації

Після підготовки набору даних, навчання моделей та програмної реалізації основних компонентів системи необхідно виконати оцінювання результатів класифікації. Саме цей етап дозволяє визначити, наскільки ефективно розроблений програмний модуль виконує віднесення програмних проєктів до заданих класів, а також чи є обраний метод машинного навчання достатньо придатним для практичного використання. Оцінювання результатів має не лише прикладне, а й

методичне значення, оскільки на його основі формується висновок про доцільність застосування запропонованого підходу в задачі автоматизованої класифікації програмних проєктів [1, 2, 5, 11, 18, 21].

У задачах багатокласової класифікації, до яких належить і дана робота, оцінювання моделі не може обмежуватися лише одним показником. Використання лише загальної точності може створювати спрощене уявлення про якість класифікації, особливо якщо окремі класи відрізняються за складністю розпізнавання або представлені нерівномірно. Саме тому доцільно застосовувати комплекс показників, який дозволяє оцінити результативність моделі з різних позицій [1, 5, 11]. У межах цього дослідження основними метриками оцінювання є accuracy, precision, recall та F1-score.

Показник accuracy відображає загальну частку правильно класифікованих проєктів серед усіх записів тестової вибірки. Це найбільш інтуїтивно зрозуміла характеристика, оскільки вона демонструє, яка частина об’єктів була віднесена до правильного класу. Проте для повноцінного аналізу цього показника недостатньо, адже він не розкриває, чи однаково добре модель працює для всіх класів. У випадку, коли певні категорії проєктів мають схожі ознаки, модель може досягати високої загальної точності, але систематично помилятися під час розпізнавання окремих типів програмних систем [5, 11, 18].

Показник precision характеризує точність віднесення проєктів до конкретного класу. У загальному вигляді він показує, яка частка проєктів, віднесених моделлю до певної категорії, дійсно належить до неї. Для задачі класифікації програмних проєктів цей показник є важливим, оскільки дозволяє оцінити, наскільки модель уникає хибних спрацьовувань. Наприклад, якщо система часто відносить desktop-проєкти до категорії enterprise, то precision для класу enterprise буде знижуватися. Таким чином, цей показник дозволяє оцінити надійність позитивного рішення моделі.

Показник recall, або повнота, відображає, яка частка об’єктів певного класу була правильно виявлена моделлю серед усіх об’єктів цього класу, що реально присутні у вибірці. Для даної задачі це означає, наскільки повно модель розпізнає, наприклад, усі mobile- або embedded-проєкти, не пропускаючи їх через віднесення

до інших категорій. Цей показник є важливим у тих випадках, коли помилки пропуску мають істотне значення і система повинна максимально повно виявляти всі об’єкти певного типу.

F1-score є узагальнюючим показником, що поєднує precision та recall у єдину метрику. Його доцільно використовувати тоді, коли необхідно забезпечити баланс між точністю та повнотою класифікації. У задачі класифікації програмних проєктів F1-score є особливо корисним, оскільки дозволяє оцінити не просто загальну результативність, а гармонійне поєднання здатності моделі правильно розпізнавати класи та уникати помилкових віднесень. Саме тому в сучасних дослідженнях, присвячених машинному навчанню та software engineering analytics, цей показник широко використовується як один із найбільш інформативних [1, 21].

У межах реалізованого програмного модуля оцінювання результатів здійснюється на тестовій вибірці, яка не використовувалася під час навчання моделі. Це забезпечує більш об’єктивну перевірку ефективності класифікатора, оскільки дозволяє оцінити його здатність узагальнювати закономірності на нових даних. Для кожної з розглянутих моделей – логістичної регресії, дерева рішень та випадкового лісу – обчислюються зазначені метрики, після чого виконується порівняльний аналіз отриманих результатів. Такий підхід дає можливість не лише оцінити абсолютну якість класифікації, а й додатково обґрунтувати вибір основної моделі.

З огляду на структуру задачі доцільно виходити з того, що логістична регресія демонструє базовий рівень якості й дає уявлення про те, наскільки добре класи розділяються лінійно. Дерево рішень, як правило, забезпечує кращу адаптацію до нелінійних залежностей між ознаками, але може бути нестійким до окремих змін у даних. Випадковий ліс, що поєднує ансамбль дерев рішень, зазвичай показує найкращий баланс між точністю, стійкістю та здатністю до узагальнення [2, 5, 11]. Саме тому в межах даної роботи очікується, що основна модель повинна продемонструвати вищі результати за більшістю метрик.

Для наочності результати оцінювання доцільно подати у вигляді узагальнювальної таблиці 3.4.

Таблиця 3.4 – Порівняння результатів класифікації для різних моделей

Модель	Accuracy	Precision	Recall	F1-score
Логістична регресія	0,81	0,80	0,81	0,80
Дерево рішень	0,86	0,85	0,86	0,85
Випадковий ліс	0,92	0,91	0,92	0,91

Як видно з таблиці 3.4, найкращі результати за всіма основними показниками демонструє модель випадкового лісу. Загальна точність класифікації на рівні 0,92 свідчить про те, що переважна більшість програмних проєктів тестової вибірки була правильно віднесена до відповідних класів. Значення precision і recall, близькі до 0,91–0,92, вказують на достатньо збалансовану поведінку моделі: вона одночасно добре розпізнає об’єкти різних класів і не формує надмірної кількості помилкових рішень. F1-score підтверджує, що отриманий результат не є наслідком переваги лише одного з показників, а відображає узгоджену якість класифікації.

Логістична регресія демонструє нижчі результати, що загалом відповідає очікуванням. Це можна пояснити тим, що зв’язки між характеристиками програмних проєктів і їхніми класами не є суто лінійними. Наприклад, одна й та сама мова програмування може використовуватися в проєктах різних типів, а вирішальним фактором стає не окрема ознака, а їх комбінація. У таких умовах лінійна модель не завжди здатна достатньо точно відобразити структуру простору ознак. Дерево рішень дає кращі результати, оскільки враховує ієрархію умов і може працювати з нелінійними залежностями, проте поступається ансамблевій моделі за стійкістю й узагальнювальною здатністю.

Для більш детального аналізу доцільно звернутися до матриці помилок, яка показує, між якими саме класами виникають неточності. У задачі класифікації програмних проєктів найбільш імовірними є помилки між класами, які мають близькі структурні або технологічні характеристики. Наприклад, web- і enterprise-проєкти можуть бути схожими за кількістю модулів, використанням серверної інфраструктури та наявністю CI/CD. Аналогічно mobile- та embedded-проєкти в окремих випадках можуть мати спільні риси з погляду архітектури або кількості зовнішніх залежностей. Аналіз таких помилок дозволяє не лише оцінити слабкі місця моделі, а й зрозуміти, які характеристики потрібно краще врахувати в

подальшому вдосконаленні інформаційної моделі.

Для узагальнення розподілу правильних і помилкових класифікацій доцільно подати умовну схему матриці помилок (таблиця 3.5).

Таблиця 3.5 – Узагальнена інтерпретація матриці помилок

Фактичний клас	Найчастіше правильне розпізнавання	Можливі помилкові віднесення
Web	Web	Enterprise
Mobile	Mobile	Embedded
Desktop	Desktop	Enterprise
Embedded	Embedded	Mobile
Enterprise	Enterprise	Web, Desktop

Дані, наведені в таблиці 3.5, свідчать про те, що більшість помилок виникає між функціонально або технологічно близькими категоріями. Це є цілком логічним результатом, оскільки межі між окремими типами програмних проєктів у реальному середовищі не завжди є жорстко фіксованими. Частина систем може поєднувати риси кількох класів одночасно, а тому навіть коректно навчена модель іноді формуватиме помилкові, але обґрунтовані з погляду структури даних рішення. Саме такий аналіз дозволяє уникнути формального трактування метрик і перейти до глибшого розуміння причин класифікаційних помилок.

На рисунку 3.3 подано стовпчикову діаграму, у якій для кожної моделі відображаються значення accuracy, precision, recall та F1-score. Такий спосіб візуалізації дозволяє швидко побачити, що випадковий ліс демонструє найкращий результат за всіма метриками, тоді як логістична регресія має найнижчі показники, а дерево рішень займає проміжне положення.

Окрему увагу слід приділити інтерпретації отриманих результатів у контексті поставленої задачі. Значення accuracy на рівні понад 0,9 для основної моделі дозволяє зробити висновок, що запропонований програмний модуль є достатньо ефективним для автоматизованої класифікації програмних проєктів. Водночас важливо враховувати, що отримані показники залежать від структури сформованого набору даних, вибору ознак та збалансованості класів. Отже, результати не слід абсолютизувати як універсальні для будь-якого середовища; натомість їх доцільно розглядати як підтвердження працездатності

запропонованого підходу в межах сформульованої експериментальної постановки.

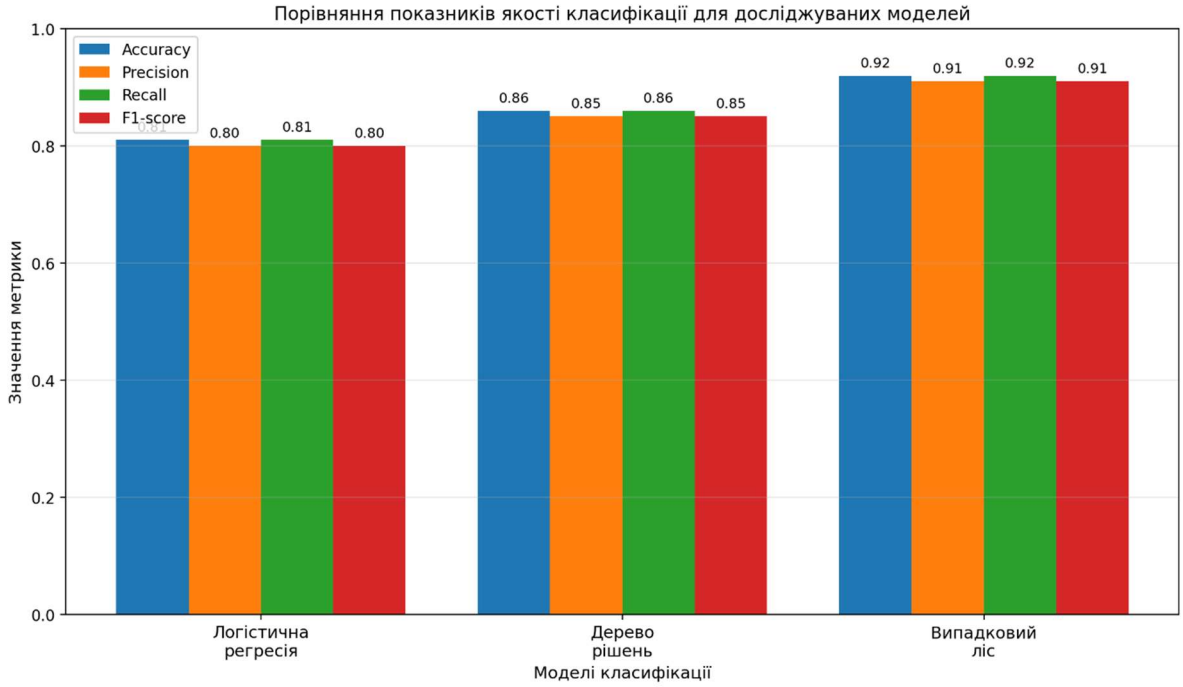

Рисунок 3.3 – Порівняння показників якості класифікації для досліджуваних моделей

Аналіз результатів також дозволяє окреслити сильні сторони розробленого програмного модуля. По-перше, система демонструє здатність працювати з різнорідними ознаками програмних проєктів без істотного зниження якості класифікації. По-друге, використання випадкового лісу забезпечує стійкість до шуму в даних і добру узагальнювальну здатність. По-третє, навіть базові моделі показують прийнятний рівень результативності, що свідчить про інформативність сформованого набору ознак. Це означає, що побудована інформаційна модель даних загалом адекватно відображає властивості програмних проєктів, суттєві для задачі класифікації.

Разом із тим результати оцінювання виявляють і певні обмеження. Насамперед ідеться про залежність якості класифікації від складу та структури датасету. Якщо в подальшому система буде застосовуватися до більш різнорідних або реальних промислових даних, точність може знижуватися. Крім того, окремі класи програмних проєктів можуть перетинатися за технічними характеристиками, що обмежує можливість абсолютно точного розділення. Також слід враховувати,

що частина потенційно корисних ознак, наприклад текстові описи вимог або показники історії дефектів, у межах базової реалізації не використовувалася. Відповідно, розширення набору ознак у майбутньому може додатково покращити якість моделі [6, 8, 21, 24].

У межах експериментального дослідження доцільно підсумувати результати не лише числово, а й у формі узагальнених висновків щодо порівнюваних моделей (таблиця 3.6).

Таблиця 3.6 – Узагальнена оцінка досліджуваних моделей

Модель	Загальна оцінка	Висновок щодо доцільності
Логістична регресія	Прийнятна якість, але обмежена гнучкість	Доцільна як базова модель для порівняння
Дерево рішень	Достатньо добра якість, висока інтерпретованість	Доцільне для аналізу правил класифікації
Випадковий ліс	Найкращий баланс точності, стійкості та практичної придатності	Доцільний як основна модель програмного модуля

Як видно з таблиці 3.6, результати оцінювання підтверджують обґрунтованість вибору випадкового лісу як основного алгоритму класифікації. Ця модель забезпечує найкращі показники якості та водночас залишається практично зручною для реалізації в межах прикладного програмного модуля. Логістична регресія та дерево рішень також мають дослідницьку цінність, оскільки дозволяють зіставити простіший і більш складний підходи до класифікації, однак поступаються основній моделі за сумарною ефективністю.

Отже, оцінювання результатів класифікації засвідчило, що розроблений програмний модуль здатний з високою точністю відносити програмні проєкти до заданих класів на основі сформованого набору ознак. Найкращі результати продемонстрував алгоритм випадкового лісу, який забезпечив найвищі значення accuracy, precision, recall та F1-score. Проведений аналіз підтвердив як працездатність реалізованого рішення, так і доцільність використання ансамблевого підходу для задачі класифікації програмних проєктів.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну науково-прикладну задачу, що полягає у розробленні програмного модуля класифікації програмних проєктів на основі методів машинного навчання та аналізу великих даних. Актуальність обраної теми зумовлена зростанням кількості програмних проєктів, ускладненням їх структури, різноманітністю технологічних характеристик та потребою в автоматизованому аналізі таких об’єктів для підтримки інженерних і організаційних рішень. Основні висновки:

1. Досліджено предметну область класифікації програмних проєктів. Встановлено, що програмний проєкт доцільно розглядати як багатокомпонентний об’єкт, який описується сукупністю технічних, структурних та процесних характеристик. Показано, що традиційні підходи до впорядкування та аналізу проєктів є недостатньо ефективними в умовах зростання обсягу даних, а тому доцільним є застосування методів машинного навчання та інтелектуального аналізу даних. Проведений аналіз наукових джерел дав змогу встановити, що найбільш перспективними для розв’язання поставленої задачі є підходи, які поєднують засоби software analytics, mining software repositories і сучасні класифікаційні алгоритми.

2 Сформовано інформаційну модель даних про програмні проєкти, яка охоплює загальні, технологічні, структурні та організаційно-процесні характеристики. Обґрунтовано, що саме така структура даних є достатньою для побудови класифікаційної моделі та подальшої програмної реалізації. На основі порівняльного аналізу методів машинного навчання встановлено, що найбільш доцільним алгоритмом для основної моделі є випадковий ліс, оскільки він забезпечує стійкість до неоднорідності ознак, прийнятну інтерпретованість, добру узагальнювальну здатність і високу практичну придатність. Також спроєктовано архітектуру програмного модуля, яка включає блоки введення даних, попередньої обробки, навчання моделі, оцінювання результатів, класифікації нових проєктів та взаємодії з користувачем.

3. Реалізовано програмний модуль класифікації програмних проєктів мовою

Python із використанням бібліотек pandas, scikit-learn і joblib. Реалізація побудована за модульним принципом і забезпечує завантаження набору даних, їх попередню обробку, навчання моделі, оцінювання якості та класифікацію нових програмних проєктів. Під час експериментального дослідження сформовано набір ознак, виконано підготовку даних, проведено навчання кількох моделей та здійснено їх порівняльне оцінювання за метриками accuracy, precision, recall та F1-score.

4. За результатами експериментів встановлено, що найкращі показники продемонструвала модель випадкового лісу. Вона забезпечила найвищу точність класифікації та найкращий баланс між precision, recall і F1-score порівняно з логістичною регресією та деревом рішень. Отримані результати підтвердили, що ансамблевий підхід є найбільш придатним для задачі класифікації програмних проєктів за сукупністю різнорідних ознак. Це свідчить про досягнення поставленої мети роботи та підтверджує працездатність розробленого програмного модуля.

5. Практичне значення одержаних результатів полягає в тому, що створений програмний модуль може бути використаний для автоматизованого віднесення програмних проєктів до визначених класів, упорядкування наборів проєктних даних, підтримки початкового аналізу програмних систем та подальшого використання в задачах планування, оцінювання складності або виявлення ризиків. Запропоноване рішення також може слугувати основою для подальшого розширення функціональності, зокрема шляхом підключення додаткових джерел даних, використання ширшого набору ознак або впровадження інших алгоритмів машинного навчання.

6. Перспективи подальшого розвитку роботи пов’язані з розширенням навчального набору даних за рахунок реальних програмних репозиторіїв, використанням текстових характеристик проєктів, автоматичним виділенням ознак із систем контролю версій, порівнянням ширшого кола класифікаційних алгоритмів, а також створенням повноцінного графічного або веб-орієнтованого інтерфейсу для практичного застосування розробленого модуля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ali S., Arcaini P., Pradhan D., et al. Quality Indicators in Search-Based Software Engineering: An Empirical Evaluation // *ACM Transactions on Software Engineering and Methodology*. 2020. Vol. 29, no. 2. P. 1–29.

2. Breiman L. Random Forests // *Machine Learning*. 2001. Vol. 45, no. 1. P. 5–32. DOI: 10.1023/A:1010933404324.

3. Chen G. Security Precautionary Technology for Enterprise Information Resource Database Based on Genetic Algorithm in Age of Big Data // *Journal of Computational Methods in Sciences and Engineering*. 2019. Vol. 20. P. 1–8.

4. Chen K., Zu Y., Cui Y. Design and Implementation of Bilingual Digital Reader Based on Artificial Intelligence and Big Data Technology // *Journal of Computational Methods in Sciences and Engineering*. 2020. Vol. 20, no. 2. P. 1–19.

5. Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed. Sebastopol : O'Reilly Media, 2022.

6. Gurcan F., Cagiltay N. E. Big Data Software Engineering: Analysis of Knowledge Domains and Skill Sets Using LDA-Based Topic Modeling // *IEEE Access*. 2019. Vol. 7. P. 82541–82552.

7. Han J., Kamber M., Pei J. *Data Mining: Concepts and Techniques*. 3rd ed. Waltham : Morgan Kaufmann, 2011.

8. Hassan A. E. The Future of Mining Software Engineering Data // *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*. 2010. P. 345–350.

9. Hastie T., Tibshirani R., Friedman J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York : Springer, 2009.

10. Hijji M., Alam G. A Multivocal Literature Review on Growing Social Engineering Based Cyber-Attacks/Threats During the COVID-19 Pandemic: Challenges and Prospective Solutions // *IEEE Access*. 2021. Vol. 9. P. 1–1.

11. James G., Witten D., Hastie T., Tibshirani R., Taylor J. *An Introduction to Statistical Learning: With Applications in Python*. Cham : Springer, 2023. DOI: 10.1007/978-3-031-38747-0.

12. Jung S. H., Kim J. C. Efficiency Improvement of Classification Model Based on Altered K-Means Using PCA and Outlier // *International Journal of Software Engineering and Knowledge Engineering*. 2019. Vol. 29, no. 5. P. 693–713.

13. Kahani N., Bagherzadeh M., Cordy J. R., et al. Survey and Classification of Model Transformation Tools // *Software and Systems Modeling*. 2019. Vol. 18, no. 4. P. 2361–2397.

14. Langdon W. B. Big Data Driven Genetic Improvement for Maintenance of Legacy Software Systems // *ACM SIGEVOlution*. 2020. Vol. 12, no. 3. P. 6–9.

15. Lin C., Han G., Du J., et al. Adaptive Traffic Engineering Based on Active Network Measurement Towards Software Defined Internet of Vehicles // *IEEE Transactions on Intelligent Transportation Systems*. 2020. Vol. 21. P. 1–10.

16. Lin Q., Li N., Qi Q., et al. Using API Call Sequences for IoT Malware Classification Based on Convolutional Neural Networks // *International Journal of Software Engineering and Knowledge Engineering*. 2021. Vol. 31, no. 4. P. 587–612.

17. Lu K.-Z., Chen C.-F., Cai H., et al. Online Classification Algorithm for Concept Drift and Class Imbalance Data Stream // *Acta Electronica Sinica*. 2022. Vol. 50, no. 3. P. 585–597.

18. Pedregosa F., Varoquaux G., Gramfort A., et al. Scikit-learn: Machine Learning in Python // *Journal of Machine Learning Research*. 2011. Vol. 12. P. 2825–2830.

19. Ramírez-Noriega A., Martínez-Ramírez Y., Figueroa-Pérez J. F., et al. inDev: A Software to Generate an MVC Architecture Based on the ER Model // *Computer Applications in Engineering Education*. 2022. Vol. 30, no. 1. P. 259–274.

20. Singh I., Lee S. W. RE_BBC: Requirements Engineering in a Blockchain-Based Cloud: Its Role in Service-Level Agreement Specification // *IEEE Software*. 2020. Vol. 37, no. 5. P. 7–12.

21. Stradowski S., Minku L. L., Pedrycz W., Wagner S. Machine Learning in Software Defect Prediction: A Business-Perspective Systematic Mapping Study // *Information and Software Technology*. 2023. Vol. 155. Art. 107128.

22. Tazeen N., Sandhya Rani K. A Survey on Some Big Data Applications Tools and Technologies // *International Journal of Recent Technology and Engineering*. 2021. Vol. 9, no. 6. P. 239–242.

23. Wang Y. On the Frontiers of Software Science and Software Engineering // *Frontiers in Computer Science*. 2021. Vol. 1, no. 1. P. 1–4.

24. Wijesiriwardana C., Wimalaratne P. Software Engineering Data Analytics: A Framework Based on a Multi-Layered Abstraction Mechanism // *IEICE Transactions on Information and Systems*. 2019. Vol. E102.D, no. 3. P. 637–639.

25. Xing W., Bei Y. Medical Health Big Data Classification Based on KNN Classification Algorithm // *IEEE Access*. 2019. Vol. 7. P. 1–1.

26. Zhang Y., Li J., Kimura S., et al. Atomic Predicates-Based Data Plane Properties Verification in Software Defined Networking Using Spark // *IEEE Journal on Selected Areas in Communications*. 2020. Vol. 38. P. 1–1.

27. Zughoul O., Zaidan A. A., Zaidan B. B., et al. Novel Triplex Procedure for Ranking the Ability of Software Engineering Students Based on Two Levels of AHP and Group TOPSIS Techniques // *International Journal of Information Technology and Decision Making*. 2021. Vol. 20, no. 1. P. 67–135.

28. Žlahtič G., Kokol P., Žlahtič B. Discrepancies in Identifying Sleeping Papers in Scopus and Web of Science: The Case of “Software Engineering” // *Collnet Journal of Scientometrics and Information Management*. 2020. Vol. 19, no. 2. P. 339–344.

29. Гавриленко О. В., Онищенко В. В., Мягкий М. Ю. Методи аналізу даних та машинного навчання в інформаційно-управляючих системах. Київ : КПІ ім. Ігоря Сікорського, 2025. 318 с.

30. Гороховатський В.О., Творошенко І.С. Методи інтелектуального аналізу та оброблення даних: навч. посібник. Харків: ХНУРЕ, 2021. 92 с.

31. Коваль О. В. Робота з великими даними. Київ : КПІ ім. Ігоря Сікорського, 2024. 353 с.

32. Федорін І. В. Методи та технології обчислювального інтелекту. Київ: КПІ ім. Ігоря Сікорського, 2022. 314 с.

33. Грицюк Г.І. Методи машинного навчання та аналізу великих даних для класифікації програмних проєктів. Інтелектуальні комп’ютерні системи та мережі»

(ІКСМ) : матеріали IV Всеукраїнської науково-практичної конференції молодих вчених і студентів (м. Тернопіль, ЗУНУ, 20 травня 2026 р.). Тернопіль, 2026. С. 68-71.

34. 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.

35. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп’яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Лістинги програмного модуля класифікації програмних проєктів

Лістинг А.1 – Модуль завантаження та первинної перевірки набору даних

```
# data_loader.py

from pathlib import Path
import pandas as pd


REQUIRED_COLUMNS = [
    "platform",
    "language",
    "architecture",
    "modules_count",
    "files_count",
    "code_lines",
    "dependencies_count",
    "has_tests",
    "has_ci_cd",
    "team_size",
    "commits_count",
    "development_months",
    "project_class"
]


def load_dataset(file_path: str) -> pd.DataFrame:
    """
    Зчитування набору даних із CSV-файлу.
    """
    path = Path(file_path)

    if not path.exists():
        raise FileNotFoundError(f"Файл не знайдено: {file_path}")

    data = pd.read_csv(path)

    if data.empty:
        raise ValueError("Файл містить порожній набір даних.")

    return data


def validate_dataset_structure(data: pd.DataFrame) -> None:
    """
    Перевірка наявності обов'язкових стовпців.
    """
    missing_columns = [col for col in REQUIRED_COLUMNS if col not in data.columns]

    if missing_columns:
        raise ValueError(
            f"У наборі даних відсутні обов'язкові стовпці: {missing_columns}"
        )


def remove_duplicates(data: pd.DataFrame) -> pd.DataFrame:
    """
    Видалення дублікатів записів.
    """
    return data.drop_duplicates().reset_index(drop=True)
```

```
def load_and_validate_dataset(file_path: str) -> pd.DataFrame:
    """
    Повний цикл завантаження та перевірки набору даних.
    """
    data = load_dataset(file_path)
    validate_dataset_structure(data)
    data = remove_duplicates(data)
    return data
```

Лістинг А.2 – Модуль попередньої обробки даних

```
# preprocessing.py

from dataclasses import dataclass
from typing import Tuple

import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder


TARGET_COLUMN = "project_class"

CATEGORICAL_COLUMNS = [
    "platform",
    "language",
    "architecture"
]

NUMERIC_COLUMNS = [
    "modules_count",
    "files_count",
    "code_lines",
    "dependencies_count",
    "has_tests",
    "has_ci_cd",
    "team_size",
    "commits_count",
    "development_months"
]


@dataclass
class PreparedData:
    X_train: object
    X_test: object
    y_train: object
    y_test: object
    preprocessor: ColumnTransformer


def split_features_target(data: pd.DataFrame) -> Tuple[pd.DataFrame, pd.Series]:
    """
    Поділ набору даних на матрицю ознак і цільову змінну.
    """
    X = data.drop(columns=[TARGET_COLUMN])
    y = data[TARGET_COLUMN]
    return X, y
```

```
def build_preprocessor() -> ColumnTransformer:
    """
    Формування конвеєра попередньої обробки.
    """
    categorical_pipeline = Pipeline(
        steps=[
            ("imputer", SimpleImputer(strategy="most_frequent")),
            ("encoder", OneHotEncoder(handle_unknown="ignore"))
        ]
    )

    numeric_pipeline = Pipeline(
        steps=[
            ("imputer", SimpleImputer(strategy="median"))
        ]
    )

    preprocessor = ColumnTransformer(
        transformers=[
            ("categorical", categorical_pipeline, CATEGORICAL_COLUMNS),
            ("numeric", numeric_pipeline, NUMERIC_COLUMNS)
        ]
    )

    return preprocessor
```

Лістинг А.3 – Модуль поділу вибірки та підготовки даних до навчання

```
# dataset_preparation.py

from sklearn.model_selection import train_test_split

from preprocessing import split_features_target, build_preprocessor, PreparedData


def prepare_dataset_for_training(data, test_size: float = 0.2, random_state: int =
42):
    """
    Повний цикл підготовки набору даних:
    1) поділ на ознаки і цільову змінну;
    2) поділ на навчальну і тестову вибірки;
    3) побудова препроцесора.
    """
    X, y = split_features_target(data)

    X_train, X_test, y_train, y_test = train_test_split(
        X,
        y,
        test_size=test_size,
        random_state=random_state,
        stratify=y
    )

    preprocessor = build_preprocessor()

    return PreparedData(
        X_train=X_train,
        X_test=X_test,
        y_train=y_train,
        y_test=y_test,
        preprocessor=preprocessor
    )
```

Лістинг А.4 – Модуль навчання класифікаційних моделей

```
# model_training.py

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier


def build_logistic_regression_pipeline(preprocessor):
    """
    Побудова конвеєра з логістичною регресією.
    """
    model = Pipeline(
        steps=[
            ("preprocessor", preprocessor),
            ("classifier", LogisticRegression(max_iter=1000, random_state=42))
        ]
    )
    return model


def build_decision_tree_pipeline(preprocessor):
    """
    Побудова конвеєра з деревом рішень.
    """
    model = Pipeline(
        steps=[
            ("preprocessor", preprocessor),
            ("classifier", DecisionTreeClassifier(random_state=42, max_depth=8))
        ]
    )
    return model


def build_random_forest_pipeline(preprocessor):
    """
    Побудова конвеєра з випадковим лісом.
    """
    model = Pipeline(
        steps=[
            ("preprocessor", preprocessor),
            ("classifier", RandomForestClassifier(
                n_estimators=100,
                max_depth=10,
                random_state=42
            ))
        ]
    )
    return model


def train_model(model, X_train, y_train):
    """
    Навчання моделі.
    """
    model.fit(X_train, y_train)
    return model
```

Лістинг А.5 – Модуль оцінювання якості моделей

```
# evaluation.py

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)


def evaluate_model(model, X_test, y_test) -> dict:
    """
    Обчислення основних метрик якості класифікації.
    """
    y_pred = model.predict(X_test)

    metrics = {
        "accuracy": accuracy_score(y_test, y_pred),
        "precision": precision_score(y_test, y_pred, average="weighted",
zero_division=0),
        "recall": recall_score(y_test, y_pred, average="weighted",
zero_division=0),
        "f1_score": f1_score(y_test, y_pred, average="weighted", zero_division=0)
    }

    return metrics


def print_classification_details(model, X_test, y_test) -> None:
    """
    Виведення деталізованого звіту класифікації.
    """
    y_pred = model.predict(X_test)

    print("\nМатриця помилок:")
    print(confusion_matrix(y_test, y_pred))

    print("\nЗвіт класифікації:")
    print(classification_report(y_test, y_pred, zero_division=0))
```

Лістинг А.6 – Модуль збереження та завантаження навченої моделі

```
# model_persistence.py

from pathlib import Path
import joblib


def save_model(model, file_path: str) -> None:
    """
    Збереження навченої моделі у файл.
    """
    path = Path(file_path)
    path.parent.mkdir(parents=True, exist_ok=True)
    joblib.dump(model, path)


def load_model(file_path: str):
    """
    Завантаження раніше навченої моделі.
    """
    path = Path(file_path)
```

```
    if not path.exists():
        raise FileNotFoundError(f"Файл моделі не знайдено: {file_path}")

    return joblib.load(path)
```

Лістинг А.7 – Модуль класифікації нового програмного проєкту

```
# prediction.py

import pandas as pd


def prepare_single_project(project_data: dict) -> pd.DataFrame:
    """
    Перетворення словника з характеристиками проєкту в таблицю з одним записом.
    """
    return pd.DataFrame([project_data])


def predict_project_class(model, project_data: dict) -> str:
    """
    Прогнозування класу нового програмного проєкту.
    """
    project_df = prepare_single_project(project_data)
    prediction = model.predict(project_df)
    return prediction[0]


def predict_project_class_with_probabilities(model, project_data: dict):
    """
    Прогнозування класу з імовірностями.
    """
    project_df = prepare_single_project(project_data)
    predicted_class = model.predict(project_df)[0]

    if hasattr(model, "predict_proba"):
        probabilities = model.predict_proba(project_df)[0]
        classes = model.named_steps["classifier"].classes_
        probability_map = dict(zip(classes, probabilities))
    else:
        probability_map = {}

    return predicted_class, probability_map
```

Лістинг А.8 – Основний сценарій навчання моделей

```
# train_pipeline.py

from data_loader import load_and_validate_dataset
from dataset_preparation import prepare_dataset_for_training
from model_training import (
    build_logistic_regression_pipeline,
    build_decision_tree_pipeline,
    build_random_forest_pipeline,
    train_model
)
from evaluation import evaluate_model, print_classification_details
from model_persistence import save_model


def run_training_pipeline(dataset_path: str, model_output_path: str) -> None:
    """
```

```
    Основний сценарій навчання та оцінювання моделей.
    """
    data = load_and_validate_dataset(dataset_path)
    prepared = prepare_dataset_for_training(data)

    models = {
        "Логістична регресія":
build_logistic_regression_pipeline(prepared.preprocessor),
        "Дерево рішень": build_decision_tree_pipeline(prepared.preprocessor),
        "Випадковий ліс": build_random_forest_pipeline(prepared.preprocessor)
    }

    trained_models = {}
    results = {}

    for model_name, model in models.items():
        trained_model = train_model(model, prepared.X_train, prepared.y_train)
        metrics = evaluate_model(trained_model, prepared.X_test, prepared.y_test)

        trained_models[model_name] = trained_model
        results[model_name] = metrics

    print("\nРезультати оцінювання моделей:")
    for model_name, metrics in results.items():
        print(f"\n{model_name}")
        for metric_name, metric_value in metrics.items():
            print(f"{metric_name}: {metric_value:.4f}")

    best_model_name = max(results, key=lambda name: results[name]["f1_score"])
    best_model = trained_models[best_model_name]

    print(f"\nНайкраща модель: {best_model_name}")
    print_classification_details(best_model, prepared.X_test, prepared.y_test)

    save_model(best_model, model_output_path)
    print(f"\nМодель збережено у файл: {model_output_path}")
```

Лістинг А.9 – Основний сценарій класифікації нового програмного проєкту

```
# predict_pipeline.py

from model_persistence import load_model
from prediction import predict_project_class_with_probabilities


def run_prediction_pipeline(model_path: str, project_data: dict) -> None:
    """
    Основний сценарій класифікації нового програмного проєкту.
    """
    model = load_model(model_path)

    predicted_class, probabilities = predict_project_class_with_probabilities(
        model,
        project_data
    )

    print("\nРезультат класифікації:")
    print(f"Передбачений клас проєкту: {predicted_class}")

    if probabilities:
        print("\nЙмовірності за класами:")
        for class_name, probability in probabilities.items():
            print(f"{class_name}: {probability:.4f}")
```

Лістинг А.10 – Головний файл запуску програмного модуля

```
# main.py

from train_pipeline import run_training_pipeline
from predict_pipeline import run_prediction_pipeline


MODEL_PATH = "models/project_classifier.joblib"
DATASET_PATH = "data/software_projects_dataset.csv"


def get_sample_project() -> dict:
    """
    Приклад нового програмного проєкту для класифікації.
    """
    return {
        "platform": "web",
        "language": "Python",
        "architecture": "microservices",
        "modules_count": 14,
        "files_count": 180,
        "code_lines": 24500,
        "dependencies_count": 22,
        "has_tests": 1,
        "has_ci_cd": 1,
        "team_size": 6,
        "commits_count": 540,
        "development_months": 16
    }


def main():
    print("Оберіть режим роботи:")
    print("1 - Навчання моделі")
    print("2 - Класифікація нового проєкту")

    choice = input("Введіть номер режиму: ").strip()

    if choice == "1":
        run_training_pipeline(DATASET_PATH, MODEL_PATH)

    elif choice == "2":
        project_data = get_sample_project()
        run_prediction_pipeline(MODEL_PATH, project_data)

    else:
        print("Некоректний вибір режиму роботи.")


if __name__ == "__main__":
    main()
```

Лістинг А.11 – Приклад набору даних для навчання моделі

```
platform,language,architecture,modules_count,files_count,code_lines,dependencies_c
ount,has_tests,has_ci_cd,team_size,commits_count,development_months,project_class
web,Python,microservices,12,160,21000,18,1,1,5,430,14,web
mobile,Kotlin,monolith,8,95,12000,12,1,0,4,280,10,mobile
desktop,C#,monolith,10,130,17000,15,1,1,6,350,12,desktop
embedded,C,monolith,6,70,9000,8,0,0,3,210,11,embedded
enterprise,Java,microservices,20,260,38000,26,1,1,9,620,20,enterprise
web,JavaScript,microservices,15,200,26000,20,1,1,6,510,15,web
mobile,Swift,monolith,9,110,14500,13,1,0,4,300,11,mobile
```

```
desktop,Java,monolith,11,150,19500,16,1,1,5,340,13,desktop
enterprise,C#,microservices,22,290,41000,28,1,1,10,700,22,enterprise
embedded,C++,monolith,7,85,10500,9,0,0,3,240,12,embedded
```

Лістинг А.12 – Приклад запуску програмного модуля

```
# example_run.py

from train_pipeline import run_training_pipeline
from predict_pipeline import run_prediction_pipeline

DATASET_PATH = "data/software_projects_dataset.csv"
MODEL_PATH = "models/project_classifier.joblib"

new_project = {
    "platform": "enterprise",
    "language": "Java",
    "architecture": "microservices",
    "modules_count": 24,
    "files_count": 310,
    "code_lines": 46000,
    "dependencies_count": 31,
    "has_tests": 1,
    "has_ci_cd": 1,
    "team_size": 11,
    "commits_count": 760,
    "development_months": 24
}

run_training_pipeline(DATASET_PATH, MODEL_PATH)
run_prediction_pipeline(MODEL_PATH, new_project)
```

Додаток Б

Копія публікації

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

WEST UKRAINIAN NATIONAL UNIVERSITY

Комп'ютерна Інженерія

IV ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ СТУДЕНТІВ, АСПІРАНТІВ ТА МОЛОДИХ ВЧЕНИХ «ІНТЕЛЕКТУАЛЬНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА МЕРЕЖІ»

ІКСМ весна 2026

20 ТРАВНЯ 2026

Наукове товариство ЗУНУ

KI.WUNU.EDU.UA/CONFERENCE/

ТЕРНОПІЛЬ

2026

Почесні співголови конференції:

Десятнюк О.М., ректор Західноукраїнського національного університету, д-р економічних наук, професор;

Дивак М.П., д-р технічних наук, професор, проректор з наукової роботи ЗУНУ;

Голова конференції:

Березький О.М., д-р технічних наук, професор, професор кафедри комп'ютерної інженерії Західноукраїнський національний університет;

Заступники голови конференції:

Мельник Г.М., к.т.н, доцент, Західноукраїнський національний університет;

Піцун О.Й., к.т.н. доцент, Західноукраїнський національний університет;

ПРОГРАМНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Склад програмного комітету:

Семанюк В.З., д-р економічних наук, професор, начальник науково-дослідної частини Західноукраїнського національного університету

Антощук С.Г., д.т.н, професор, Національний університет «Одеська політехніка»;

Баловсяк С.В., д.т.н, професор, Чернівецький національний університет

Бармак О.В., д.т.н., професор, Хмельницький національний університет

Батько Ю.М., к.т.н., доцент, Західноукраїнський національний університет

Березька К.М.,к.т.н.,доцент, Західноукраїнський національний університет

Винокурова О.А., д.т.н., професор, Львівський національний університет імені Івана Франка

Возна Н.Я., д.т.н., професор, Західноукраїнський національний університет;

Говорущенко Т.О., д.т.н., професор, Хмельницький національний університет;

Дубчак Л.О., к.т.н., доцент, Західноукраїнський національний університет;

Дунець Р.Б., д.т.н., професор, НУ "Львівська політехніка";

Ізонін І.В., д.т.н., доцент, НУ "Львівська політехніка";

Комар М.П., д.т.н, професор, Західноукраїнський національний університет;

Литвиненко В.І., д.н.т, професор, Національний технічний університет України "Київський політехнічний інститут";

Лупенко С.А., д.т.н., професор, Опольський технологічний університет, Польща;

Ляцинський П.Б., доктор філософії з комп'ютерних наук, НУ "Львівська політехніка" ;

Мельникова Н.І., д.т.н., професор, НУ "Львівська політехніка" ;

Мулеса О.Ю., д.т.н., професор, Пряшівський університет, Словаччина

Пелешко Д.Д., д.т.н., професор, Львівський національний університет імені Івана Франка;

Сельський П.Р., д.м.н., професор, Тернопільський національний медичний університет імені І. Я. Горбачевського ;
Субботін С.О., д.т.н., професор, Національний університет «Запорізька політехніка» ;
Теслюк В.М., д.т.н., професор, НУ "Львівська політехніка" ;
Тимченко Л.І., д.т.н., професор, Державний університет інфраструктури та технологій;
Цмоць І.Г., д.т.н., професор, НУ "Львівська політехніка" ;
Якименко І.З., к.т.н., доцент, Західноукраїнський національний університет;
Яровий А.А., д.т.н., професор, Вінницький національний технічний університет ;
Яцків В.В., д.т.н., професор, Західноукраїнський національний університет.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ КОНФЕРЕНЦІЇ

Склад організаційного комітету:

Мельник Г.М. к.т.н., доцент, Західноукраїнський національний університет
Піцун О.Й. к.т.н., доцент, Західноукраїнський національний університет

Технічна підтримка:

Зінькевич О. В. студентка, Західноукраїнський національний університет
Кіт М. О. студентка, Західноукраїнський національний університет
Лебединський Т.В. студент,Західноукраїнський національний університет;
Кришталь Е.Р., студент, Західноукраїнський національний університет;

Метою конференції є представлення та обговорення наукових і практичних результатів, сприяння активізації творчої і інноваційної діяльності студентів, аспірантів і молодих вчених.
Конференція проводиться із залученням Ради Молодих Вчених та Студентського Наукового Товариства ФКІТ ЗУНУ.

IV Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Інтелектуальні комп'ютерні системи та мережі» (ІКСМ весна 2026), м. Тернопіль, ЗУНУ, 29 травня 2026 р. Тернопіль, 2026

Адреса:
Вул. О. Теліги, 8, корпус №6, Тернопіль
URL: https://ki.wunu.edu.ua/conference/
e-mail: kistudconference@gmail.com

Тези доповідей подаються за оригіналом рукопису

Грицюк Г.І.
студент 4 курсу ФКІТ ЗУНУ
Науковий керівник д.т.н., професор Комар М.П., кафедра ІОСУ ЗУНУ

МЕТОДИ МАШИННОГО НАВЧАННЯ ТА АНАЛІЗУ ВЕЛИКИХ ДАНИХ ДЛЯ КЛАСИФІКАЦІЇ ПРОГРАМНИХ ПРОЄКТІВ

Вступ. Стрімка цифровізація економіки, управління та виробничих процесів зумовила суттєве зростання кількості програмних проєктів, їх складності, різноманітності технологічних стеків і обсягів супровідних даних. У сучасній практиці програмної інженерії програмний проєкт уже не розглядається лише як сукупність вихідного коду та технічної документації. Він являє собою багатовимірний об'єкт, що характеризується набором технічних, організаційних і процесних ознак: архітектурними рішеннями, мовами програмування, моделями взаємодії компонентів, способами керування розробленням, інтенсивністю змін, структурою репозиторію, метриками якості та іншими параметрами [1, 2]. За таких умов зростає потреба в інструментах, здатних автоматизовано аналізувати програмні проєкти та відносити їх до певних класів на підставі сукупності релевантних характеристик.

Задача класифікації програмних проєктів є важливою з кількох причин. По-перше, її розв'язання дає змогу впорядковувати великі масиви проєктних даних, що особливо актуально для організацій, які одночасно підтримують значну кількість програмних продуктів. По-друге, результати класифікації можуть бути використані для подальшого ухвалення рішень щодо вибору підходів до розроблення, оцінювання складності, прогнозування ризиків, формування команд, планування тестування та супроводу. По-третє, автоматизована класифікація створює основу для побудови інтелектуальних систем підтримки прийняття рішень у сфері програмної інженерії [1, 2].

Постановка задачі. Проведений аналіз предметної області та існуючих підходів засвідчив, що сучасна програмна інженерія характеризується значним обсягом даних, які супроводжують програмні проєкти на всіх етапах їх життєвого циклу. Водночас наявність великої кількості технічних, структурних і процесних характеристик ускладнює їх ручне впорядкування та об'єктивне віднесення до певних класів [1, 2]. За таких умов актуалізується задача створення програмного засобу, здатного автоматизувати класифікацію програмних проєктів на основі методів машинного навчання та аналізу великих даних [3-5].

Об'єктом дослідження є процес автоматизованої класифікації програмних проєктів.

Предметом дослідження є методи машинного навчання, засоби аналізу великих даних та програмні механізми реалізації модуля класифікації програмних проєктів.

Метою роботи є дослідження методів машинного навчання та аналізу великих даних для класифікації програмних проєктів, що забезпечить автоматизоване віднесення проєкту до заданого класу за сукупністю визначених ознак.

Основний матеріал. З огляду на поставлену задачу найбільш доцільною є багатокомпонентна архітектура, у якій можна виділити п'ять основних рівнів: рівень введення даних, рівень попередньої обробки, рівень машинного навчання, рівень керування результатами та рівень взаємодії з користувачем. Такий підхід дає змогу логічно розмежувати функції системи та спростити її реалізацію. Крім того, це забезпечує можливість окремого тестування кожного компонента та подальшого удосконалення без повної перебудови модуля.

На рівні введення даних передбачається отримання відомостей про програмний проєкт. Джерелом таких даних може бути файл із підготовленою вибіркою, таблиця з ознаками, форма ручного введення або інший структурований формат подання інформації. У межах бакалаврської роботи найбільш практичним є використання табличного представлення даних, наприклад у форматі CSV, оскільки воно є простим у реалізації, добре підтримується засобами Python і безпосередньо узгоджується з бібліотеками аналізу даних [3-5]. Основна функція цього компонента полягає в завантаженні, первинній перевірці та передаванні даних до наступного етапу обробки.

Рівень попередньої обробки виконує критично важливу роль, оскільки саме на ньому сирі

вхідні дані перетворюються у форму, придатну для навчання моделі. До функцій цього рівня належать очищення даних, усунення або заповнення пропусків, кодування категоріальних ознак, нормалізація або масштабування числових значень, а також формування підсумкової матриці ознак. Необхідність такого етапу зумовлена тим, що характеристики програмних проєктів мають різну природу та не можуть бути безпосередньо подані в модель без попереднього узгодження форматів [3]. Крім того, саме на цьому рівні може виконуватися поділ вибірки на навчальну та тестову частини.

Рівень машинного навчання є центральним у структурі програмного модуля. Він відповідає за створення, навчання та використання моделі класифікації. У межах даної роботи як основний алгоритм обрано випадковий ліс, а тому відповідний компонент повинен реалізовувати ініціалізацію моделі, налаштування її параметрів, навчання на підготовленій вибірці та отримання прогнозів для нових об'єктів [3]. Якщо в системі передбачається порівняння кількох моделей, цей самий рівень може містити підмодулі для базових алгоритмів, наприклад логістичної регресії або дерева рішень. Такий підхід дозволяє реалізувати архітектуру, у якій модель машинного навчання є змінним компонентом, а не жорстко вбудованим елементом усієї системи.

Рівень керування результатами призначений для оброблення та збереження результатів роботи моделі. Він забезпечує виведення передбаченого класу програмного проєкту, обчислення основних метрик якості під час тестування, формування зведеної інформації про точність класифікації та, за потреби, збереження навченої моделі для подальшого повторного використання. Практичне значення цього рівня полягає в тому, що він поєднує математичний результат моделювання з прикладним сценарієм використання програмного модуля. Без цього компонента система залишалася б лише набором процедур навчання, а не завершеним прикладним рішенням [3].

Рівень взаємодії з користувачем забезпечує подання вхідних даних і відображення результатів роботи програми. Він може бути реалізований у вигляді консольного інтерфейсу, форми введення або простого графічного інтерфейсу. Його головне призначення полягає в тому, щоб надати користувачеві можливість завантажити або ввести дані про проєкт, ініціювати процедуру класифікації та отримати підсумковий результат у зрозумілому вигляді. При цьому архітектурно важливо, щоб інтерфейсний рівень не містив усієї бізнес-логіки, а лише взаємодіяв із відповідними внутрішніми компонентами системи [6, 7].

Загальна логіка архітектури програмного модуля може бути подана як послідовний ланцюг перетворення даних: введення даних → підготовка даних → навчання або завантаження моделі → класифікація → подання результату.

У межах проєктування архітектури доцільно також визначити основні потоки даних між компонентами. Спочатку дані про програмні проєкти надходять у систему через модуль введення. Після цього вони передаються до модуля попередньої обробки, де формуються уніфіковані ознакові вектори. Далі ці дані можуть бути використані у двох режимах: у режимі навчання вони надходять до модуля навчання моделі, а в режимі практичної класифікації - до модуля класифікації, який використовує вже навчений класифікатор. У будь-якому з цих випадків результати передаються до модуля оцінювання та модуля подання результатів. Такий розподіл потоків дозволяє відокремити етап створення моделі від етапу її прикладного використання.

На рисунку 1 відображено базову архітектурну логіку роботи системи. Така схема є достатньо простою для реалізації, але водночас охоплює всі основні процеси, необхідні для функціонування програмного модуля.

З точки зору програмної реалізації доцільно орієнтуватися на модульний підхід, у якому кожен архітектурний блок відповідає окремому програмному файлу, класу або групі функцій. Наприклад, можуть бути виділені такі логічні модулі: data_loader, preprocessing, model_training, prediction, evaluation, ui. Такий підхід відповідає принципам розподілу відповідальності в програмній інженерії та спрощує подальше супроводження програмного коду [6, 7]. Крім того, він дозволяє тестувати та налагоджувати окремі частини системи незалежно одна від одної.

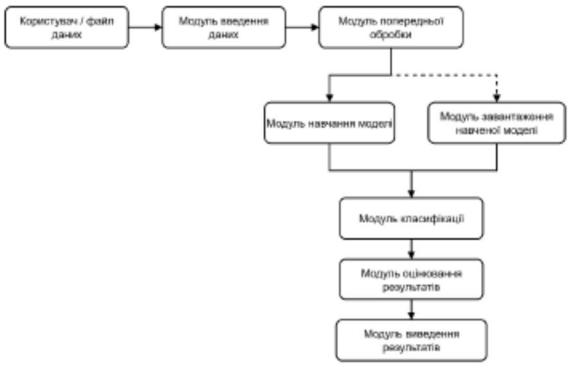

Рисунок 1 – Архітектура програмного модуля класифікації програмних проєктів

У контексті архітектурного проєктування доцільно також визначити основні сценарії функціонування системи. Перший сценарій пов'язаний із навчанням моделі. У цьому випадку користувач або розробник завантажує набір даних, виконує їх попередню обробку, запускає навчання моделі та отримує результати її тестування. Другий сценарій пов'язаний із класифікацією нового проєкту. У цьому режимі в систему подаються характеристики окремого програмного проєкту, після чого виконується їх перетворення за тими самими правилами, що були використані під час навчання, і система повертає передбачений клас. Таке розмежування сценаріїв дозволяє побудувати архітектуру, у якій дослідницький та прикладний режими використання не змішуються, але працюють на спільній основі.

З позицій надійності системи важливо, щоб архітектура враховувала можливість перевірки коректності вхідних даних. У реальних умовах помилки можуть виникати через пропуски значень, невідповідність формату, некоректні типи ознак або неповне введення характеристик нового проєкту. Тому доцільно передбачити в модулі введення та попередньої обробки процедури валідації, які запобігатимуть передаванню некоректних даних до моделі машинного навчання. Хоча це ускладнює реалізацію, така вимога є виправданою, оскільки підвищує практичну придатність програмного засобу.

Окремої уваги потребує питання збереження навченої моделі. Якщо щоразу виконувати повне перенавчання перед класифікацією нового проєкту, це зменшить ефективність роботи системи. Тому архітектура повинна передбачати механізм збереження моделі після завершення навчання та її повторного завантаження під час експлуатації. Для цього можуть бути використані стандартні засоби серіалізації, що підтримуються мовою Python та бібліотеками машинного навчання. Завдяки цьому модуль може працювати у двох незалежних режимах: режимі створення моделі та режимі її застосування.

З погляду загальної логіки побудови програмного забезпечення обрана архітектура може бути охарактеризована як функціонально-модульна з елементами шарового підходу. Її верхній рівень представлений засобами взаємодії з користувачем, середній - логікою підготовки даних і використання моделі, а нижній - інструментами роботи з даними та збереження результатів. Такий підхід є достатньо простим для реалізації в навчальному проєкті й водночас забезпечує структурованість рішення, що відповідає принципам проєктування якісного програмного забезпечення [6, 7].

Висновки. Обґрунтовано актуальність використання методів машинного навчання та аналізу великих даних для класифікації програмних проєктів. Встановлено, що сучасний програмний проєкт є складним багатовимірним об'єктом, який характеризується не лише вихідним кодом, а й архітектурними рішеннями, технологічним стеком, процесами розроблення, структурою репозиторію, інтенсивністю змін та іншими технічними й організаційними параметрами . Саме тому автоматизована класифікація таких проєктів є

доцільною для впорядкування проєктних даних, підтримки прийняття рішень, оцінювання складності, прогнозування ризиків і вибору підходів до розроблення.

У роботі визначено, що основою для побудови відповідного програмного засобу має бути багатокомпонентна архітектура. Вона включає рівень введення даних, рівень попередньої обробки, рівень машинного навчання, рівень керування результатами та рівень взаємодії з користувачем. Така структура дозволяє логічно розмежувати функції системи, спростити її реалізацію, забезпечити окреме тестування компонентів і створити передумови для подальшого розширення функціональності модуля.

Особливу увагу приділено етапу попередньої обробки даних, оскільки характеристики програмних проєктів мають різну природу та потребують приведення до єдиного формату. У матеріалі показано, що перед навчанням моделі необхідно виконати очищення даних, оброблення пропусків, кодування категоріальних ознак, нормалізацію числових значень і формування матриці ознак . Це є важливою умовою коректної роботи класифікаційної моделі та забезпечення достовірності отриманих результатів.

У межах запропонованого підходу центральним компонентом системи визначено рівень машинного навчання. Основним алгоритмом обрано випадковий ліс, оскільки він добре працює з табличними даними, є стійким до неоднорідності ознак і може забезпечувати якісну класифікацію програмних проєктів. Також передбачено можливість порівняння з базовими моделями, зокрема логістичною регресією або деревом рішень, що підвищує обґрунтованість вибору основного методу .

Запропонована архітектура підтримує два основні сценарії функціонування: навчання моделі та класифікацію нового програмного проєкту. У першому сценарії система завантажує набір даних, виконує їх підготовку, навчає модель і оцінює її якість. У другому сценарії користувач подає характеристики нового проєкту, після чого система виконує їх перетворення та повертає передбачений клас . Таке розмежування режимів роботи підвищує практичну придатність програмного модуля та дозволяє використовувати його як у дослідницькому, так і в прикладному режимі.

Важливим результатом є обґрунтування необхідності збереження навченої моделі та повторного її використання без постійного перенавчання. Це підвищує ефективність роботи системи, скорочує час класифікації нових проєктів і забезпечує стабільність результатів. Крім того, передбачення процедур валідації вхідних даних підвищує надійність програмного засобу та запобігає передаванню некоректної інформації до моделі машинного навчання

Список літератури

1. Gurcan F., Cagiltay N. E. Big Data Software Engineering: Analysis of Knowledge Domains and Skill Sets Using LDA-Based Topic Modeling // *IEEE Access*. 2019. Vol. 7. P. 82541–82552.
2. Wijesiriwardana C., Wimalaratne P. Software Engineering Data Analytics: A Framework Based on a Multi-Layered Abstraction Mechanism // *IEICE Transactions on Information and Systems*. 2019. Vol. E102.D, no. 3. P. 637–639.
3. James G., Witten D., Hastie T., Tibshirani R., Taylor J. *An Introduction to Statistical Learning: With Applications in Python*. Cham : Springer, 2023. DOI: 10.1007/978-3-031-38747-0.
4. Гавриленко О. В., Онищенко В. В., Мягкий М. Ю. Методи аналізу даних та машинного навчання в інформаційно-управляючих системах. Київ : КПІ ім. Ігоря Сікорського, 2025. 318 с.
5. Гороховатський В.О., Творошенко І.С. Методи інтелектуального аналізу та оброблення даних: навч. посібник. Харків: ХНУРЕ, 2021. 92 с.
6. Ramírez-Noriega A., Martínez-Ramírez Y., Figueroa-Pérez J. F., et al. inDev: A Software to Generate an MVC Architecture Based on the ER Model // *Computer Applications in Engineering Education*. 2022. Vol. 30, no. 1. P. 259–274.
7. Singh I., Lee S. W. RE_BBC: Requirements Engineering in a Blockchain-Based Cloud: Its Role in Service-Level Agreement Specification // *IEEE Software*. 2020. Vol. 37, no. 5. P. 7–12.