НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

Завідувач кафедри

_____Олена ЧУМАЧЕНКО

«___» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: **«Інтелектуальна система багатокласової класифікації на основі регуляризованого бустінгу»**

Виконав :
студент IV курсу, групи КІ-93
Шелепало Данило Олегович _____

Керівник:
професор, д.т.н., Чумаченко Олена Іллівна _____

Консультант з нормконтролю:
Гончарук Максим Миколайович _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Надія Василівна _____

Рецензент:
доцент, к.т.н., Сергеєв Ігор Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ, 2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітня програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ:

Завідувач кафедри

_____Олена ЧУМАЧЕНКО

«___» _____ 2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Шелепало Данило Олегович

1. Тема роботи «Інтелектуальна система багатокласової класифікації на основі регуляризованого бустінгу», керівник роботи професор, д.т.н. Чумаченко Олена Іллівна, затверджені наказом по університету від «30» травня 2023 р. № 2065-с
2. Термін подання студентом роботи 08.06.2023
3. Вихідні дані до роботи – Приклади алгоритмів для вирішення завдання багатокласової класифікації на основі регуляризованого бустінгу; набір даних, для розв'язання задачі багатокласової класифікації.
4. Зміст роботи – 1. Дослідження предметної області; 2. – Математичні основи робота; 3. Аналіз результатів роботи; 4. Функціонально-вартісний аналіз програмного продукту.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	доцент, к.е.н., Рощина Н.В.		

7. Дата видачі завдання 21.02.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формулювання тематики (напряму) дослідження.	27.02.2023	Виконано
2.	Аналіз відомих алгоритмів стосовно тематики дослідження	25.03.2023	Виконано
3.	Формулювання задач дослідження	10.04.2023	Виконано
4.	Підготовка першого розділу	16.04.2023	Виконано
5.	Підготовка другого розділу	08.05.2023	Виконано
6.	Розробка програмного продукту	18.05.2023	Виконано
7.	Підготовка третього розділу	24.05.2023	Виконано
8.	Підготовка економічної частини	01.06.2023	Виконано
9.	Підготовка презентації доповіді	03.06.2023	Виконано
10.	Оформлення дипломної роботи	05.06.2023	Виконано

Студент Данило ШЕЛЕПАЛО

Керівник Олена ЧУМАЧЕНКО

АНОТАЦІЯ

Дипломна робота: 96 сторінки, 12 рисунки, 6 таблиць, 1 додаток, 26 джерела.

ІНТЕЛЕКТУАЛЬНА СИСТЕМА, МАШИННЕ НАВЧАННЯ, БАГАТОКЛАСОВА КЛАСИФІКАЦІЯ, РЕГУЛЯРИЗОВАНИЙ БУСТІНГ, XGBOOST, ОЦІНКА ПРОДУКТИВНОСТІ.

Метою даної дипломної роботи є розробка та реалізація інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу. Робота спрямована на вивчення та вдосконалення методів класифікації для вирішення складних завдань, де необхідно розподілити об'єкти на багато класів.

Актуальність теми - багатокласова класифікація є важливим завданням у сфері машинного навчання та аналізу даних. Зростання обсягів даних та складність проблем, що потребують розподілу об'єктів на багато класів, створюють потребу у розробці нових ефективних методів класифікації. Регуляризований бустінг є одним з потужних інструментів, який дозволяє досягти високої точності та здатності до узагальнення.

Об'єкт дослідження: Об'єктом дослідження є інтелектуальна система багатокласової класифікації.

Предмет дослідження: Предметом дослідження є регуляризований бустінг та його застосування в контексті багатокласової класифікації.

В роботі розроблено програмний продукт на мові програмування Python.

ABSTRACT

Bachelor's thesis: 96 pages, 12 figures, 6 tables, 1 appendix, 26 references. INTELLIGENT SYSTEM, MACHINE LEARNING, MULTICLASS CLASSIFICATION, REGULARIZED BOOSTING, XGBOOST, PERFORMANCE EVALUATION.

The aim of this bachelor's thesis is to develop and implement an intelligent system for multiclass classification based on regularized boosting. The work is focused on studying and improving classification methods for solving complex tasks that involve assigning objects to multiple classes.

The relevance of the topic lies in the fact that multiclass classification is an important task in the field of machine learning and data analysis. The increasing volume of data and the complexity of problems requiring the allocation of objects to multiple classes create a need for the development of new effective classification methods. Regularized boosting is one powerful tool that enables achieving high accuracy and generalization capability.

The object of the research is the intelligent system for multiclass classification. The subject of the research is the regularized boosting and its application in the context of multiclass classification.

The software product has been developed in the Python programming language.

ЗМІСТ

ВСТУП

У сучасному світі, де великі обсяги даних зростають з кожним днем, задача багатокласової класифікації стає все більш актуальною. Багатокласова класифікація полягає у визначенні категорії або класу для нових спостережень на основі навчальних даних. Це важна задача в областях, таких як обробка природних мов, комп'ютерне зору, медична діагностика, фінансовий аналіз та багато інших.

Ми зосередимось на розв'язанні задачі багатокласової класифікації з використанням інтелектуальних систем на основі регуляризованого бустінгу. Регуляризований бустінг є потужним алгоритмом машинного навчання, який поєднує техніку бустінгу з регуляризацією для покращення точності та універсальності моделі.

У першому розділі нашої роботи ми ознайомимося з проблемою багатокласової класифікації та регуляризованим бустінгом. Ми розглянемо загальний огляд цих понять, важливість багатокласової класифікації, виклики та потенційні застосування. Також буде проведений огляд існуючих методів багатокласової класифікації та регуляризованого бустінгу.

У другому розділі ми розглянемо математичні основи інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу. Ми детально розглянемо алгоритми багатокласової класифікації, основи регуляризованого бустінгу та методи оцінки продуктивності в багатокласовій класифікації.

Результати цієї роботи відкриють нові можливості для ефективної багатокласової класифікації за допомогою інтелектуальних систем на основі регуляризованого бустінгу. Наш дослідницький внесок полягатиме у покращенні точності та робастності таких систем, що дозволить застосовувати їх в різноманітних сферах.

1 ОЗНАЙОМЛЕННЯ З ПРОБЛЕМОЮ БАГАТОКЛАСОВОЇ КЛАСИФІКАЦІЇ ТА РЕГУЛЯРИЗОВАНИМ БУСТІНГОМ

1.1 Загальний огляд

1.1.1 Опис домену застосування

Багатокласова класифікація застосовується в різних галузях та індустріях, включаючи:

- Біологічна таксономія: Рівні класифікації в біологічній таксономії, такі як домен, царство, тип, клас, ряд, родина, рід та вид, є прикладом багатокласової класифікації.

- Обробка природної мови (NLP): Багатокласова класифікація використовується в NLP для завдань, таких як класифікація документів та аналіз настрою. Наприклад, в класифікації документів для набору новинних статей, вхідні дані - це послідовність слів, а вихідні дані - один клас або мітка. Рекурентні нейронні мережі (RNN), зокрема Long Short Term Memory (LSTM), можуть використовуватися для вирішення таких задач.

- Data Science: Багатокласова класифікація використовується в data science для завдань, таких як сегментація клієнтів та категоризація продуктів. Наприклад, в задачі сегментації клієнтів ми можемо використовувати XGBoost для багатокласової класифікації, щоб розділити клієнтів за рівнем доходу, середніми витратами, кредитним рейтингом або комбінацією цих змінних.

Для виконання багатокласової класифікації потрібно:

- Кодувати категоріальні змінні у бінарні змінні за допомогою one-hot encoding.

- Використовувати відповідну функцію активації, таку як softmax, в вихідному шарі для створення виходу для кожного класу.

- Використовувати відповідну функцію втрат, таку як categorical_crossentropy, для багатокласової класифікації.

Важливо зазначити, що перенавчання може бути проблемою в багатокласовій класифікації, і для вирішення цієї проблеми може знадобитися більше даних.

Багатокласова класифікація - це тип задачі класифікації, яка передбачає прогнозування однієї з більше ніж двох міток класу. Вона використовується, коли кількість міток класу більше двох, і, на відміну від бінарної класифікації, вона не має поняття "нормальний" і "абнормальний" результатів. Замість цього, приклади класифікуються як належні до одного з відомих класів. Багатокласова класифікація може використовуватися в різних застосунках, таких як системи розпізнавання облич, де модель прогнозує фотографію як належну одному з тисяч або десятків тисяч облич.

Існує різноманітні алгоритми, які можна використовувати для багатокласової класифікації, включаючи наївний Баєс, дерева рішень, метод опорних векторів (SVM), випадковий ліс класифікаторів, k-найближчих сусідів (KNN) та логістичну регресію. Ці алгоритми можуть бути адаптовані для використання у багатокласових задачах, таких як застосування стратегії побудови кількох моделей бінарної класифікації для кожного класу в порівнянні з усіма іншими класами (називається "один проти всіх") або одна модель для кожної пари класів (називається "один проти одного").

У багатозначній класифікації для кожного прикладу може бути передбачено дві або більше міток класу, де кожний вихід вважається передбачуваним розподілом ймовірності Бернуллі. Багатозначна класифікація відрізняється від бінарної та багатокласової класифікації, де для кожного прикладу передбачається лише одна мітка класу. Спеціалізовані версії стандартних алгоритмів класифікації можуть бути використані для багатозначної класифікації.

Зазвичай, моделювання багатокласової класифікації виконується за допомогою моделі, яка передбачає ймовірність Мультінуллі для кожного прикладу. Розподіл Мультінуллі - це дискретний ймовірнісний розподіл, який охоплює випадок, коли подія матиме категоріальний результат, наприклад, K у

{1, 2, 3, ..., K}. У випадку класифікації це означає, що модель передбачає ймовірність того, що приклад належить до кожної мітки класу.

Виділення спану - це задача мультицільового прогнозування, де модель повинна вибрати початкові і кінцеві індекси у параграфі. Це не задача класифікації, а завдання генерації послідовності.

Отже, інтелектуальні системи багатокласової класифікації можуть бути використані для вирішення проблем, які передбачають прогнозування однієї з більше ніж двох міток класу. Багатокласова класифікація може бути здійснена за допомогою різних алгоритмів, і її моделі часто передбачають ймовірність Мультінуллі для кожного прикладу. Багатозначна класифікація передбачає одну або кілька міток для кожного прикладу і відрізняється від бінарної та багатокласової класифікації. Виділення спану - це задача мультицільового прогнозування і не є задачею класифікації, але задачею генерації послідовності.

1.1.2 Значення багатокласової класифікації:

Багатокласова класифікація - це метод класифікації екземплярів на одну з трьох або більше класів, тоді як бінарна класифікація передбачає розподіл екземплярів на один із двох класів. Багатокласова класифікація є важливою з таких причин:

- Вона застосовується, коли у вихідних навчальних даних є більше двох класів, що часто має місце у реальних задачах.

- Алгоритми, призначені для бінарної класифікації, не можуть бути застосовані до багатокласових класифікаційних задач.

- Багатокласова класифікація може забезпечити більш детальну та точну інформацію про дані. Наприклад, у випадку медичної діагностики багатокласова класифікація допомагає виявити конкретні захворювання, а не просто бінарний результат "хворий" або "нехворий".

Існують дві моделі алгоритму багатокласової класифікації:

- Модель класифікації один-проти-всіх
- Модель класифікації один-проти-одного

Багатокласова класифікація - це задача класифікації в машинному навчанні, яка включає в себе класифікацію екземплярів в один із трьох або більше класів. Вона допомагає зрозуміти складні залежності між багатьма класами або категоріями.

Багатокласова класифікація відрізняється від бінарної класифікації, де використовується лише класифікація екземплярів в один із двох класів.

Існує кілька стратегій для багатокласової класифікації. Одна з поширених стратегій полягає у перетворенні задачі на кілька задач бінарної класифікації, що може бути досягнуто за допомогою підходу один-проти-решти або один-проти-одного. Інша стратегія полягає у розширенні наявних бінарних класифікаторів для вирішення задач багатокласової класифікації.

Деякі популярні алгоритми для багатокласової класифікації включають наївний баєсівський класифікатор, дерева рішень, метод опорних векторів (SVM), випадковий ліс, метод найближчих сусідів (KNN) і логістичну регресію.

Підхід один-проти-решти (OvR) передбачає навчання n класифікаторів, де один клас вважається позитивним, а решта - негативними. Кожен класифікатор передбачає ймовірність належності до певного класу, і клас з найвищою ймовірністю вважається відповіддю. Підхід один-проти-одного (OvO) передбачає створення n * (n-1) / 2 бінарних класифікаторів, де кожен класифікатор передбачає один мітку класу. Вибирається клас, який передбачається найбільш часто.

Нейронні мережі, зокрема багатокласові персептрони, надають природне розширення до проблеми багатокласової класифікації. Вивчення рішень на основі дерева - це інший потужний метод класифікації, який природно впорається з бінарними або багатокласовими задачами класифікації. Ієрархічна класифікація - це ще один метод, який вирішує проблему багатокласової класифікації шляхом поділу простору виходів на дерево.

У обробці природної мови та багатокласовій класифікації тексту, наївний баєсівський модель є досить популярною завдяки своїй простоті і швидкості навчання. Модель використовує теорему Баєса для розкладання спільної ймовірності належності до класу на послідовність умовних ймовірностей, і вона робить наївну припущення, що всі вхідні ознаки моделі є взаємно незалежними.

Згорткові нейронні мережі (CNN) - це тип моделі глибокого навчання, які особливо добре справляються з завданнями багатокласової класифікації, особливо для зображень і тексту. Вони витягують корисні ознаки з даних, особливо такі, які є інваріантними до масштабування, трансформації і обертання. Це допомагає виявляти зображення, які можуть бути обернутими, зменшеними або зміщеними, що дозволяє досягти вищої точності у завданнях багатокласової класифікації зображень.

Отже, багатокласова класифікація є важливою, оскільки вона застосовується в реальних задачах, де є більше двох класів у вихідних навчальних даних. Алгоритми, розроблені для бінарної класифікації, не можуть бути застосовані до багатокласових класифікаційних задач. Багатокласова класифікація може надавати більш детальну та точну інформацію про дані.

1.1.3 Виклики та складнощі:

Багатокласова класифікація ставить перед дослідниками та практиками кілька викликів. Ось деякі з основних проблем:

- Незбалансовані дані: У багатокласовій класифікації набір даних може не мати рівної представленості всіх класів. Це може призвести до перенавчання на більшість класу та поганої продуктивності на меншість класу. Техніки, такі як перевибірка та недовибірка, можуть бути використані для балансування набору даних.

- Гетерогенність даних: Дані в багатокласовій класифікації можуть походити з різних джерел або мати різні формати, що ускладнює їх обробку та аналіз.

- Великий обсяг даних: Проблеми багатокласової класифікації часто включають великі набори даних, які можуть бути важкими для зберігання та обробки.

- Вибір алгоритму: Вибір правильного алгоритму для задачі багатокласової класифікації не завжди є простим. Деякі алгоритми мають бінарну природу і вимагають модифікацій для роботи з більш ніж двома класами.

Для вирішення цих викликів дослідники та практики використовують різноманітні техніки. Ось деякі приклади:

- Трансформація до бінарного формату: Один підхід полягає в зведенні багатокласової задачі до кількох задач бінарної класифікації. Це можна зробити за допомогою технік, таких як підхід "один проти решти" (one-vs.-rest) та підхід "один проти одного" (one-vs.-one).

- Розширення з бінарного формату: Інший підхід полягає в розширенні існуючих бінарних класифікаторів для розв'язання задач багатокласової класифікації. Це можна зробити за допомогою алгоритмів, таких як нейронні

мережі, дерева рішень, k-найближчих сусідів, наївний Баєс, машини опорних векторів та екстремальні навчальні машини.

- Ієрархічна класифікація: Третій підхід полягає в розбитті вихідного простору на деревоподібну структуру, де кожен дочірній вузол представляє лише один клас. Це називається ієрархічною класифікацією.

Крім цих технік, існують кілька найкращих практик для багатокласової класифікації:

- Оцінка продуктивності моделі: Точність може не бути найкращою метрикою для оцінки продуктивності моделі у випадку незбалансованих наборів даних. Матриця помилок є хорошою технікою для узагальнення продуктивності алгоритму класифікації. Криві точність-повнота корисні, коли класи дуже незбалансовані.

- Збалансування набору даних: Недовибірка та перевибірка можуть бути використані для балансування набору даних. Також можна оцінити ваги класів для надання деякого переважання меншиновим класам під час навчання моделі.

- Вибір правильного алгоритму: Вибір правильного алгоритму для багатокласової класифікації вимагає врахування таких факторів, як розмір набору даних, кількість класів та характер даних.

Висновки, багатокласова класифікація ставить перед дослідниками та практиками кілька викликів, але їх можна подолати за допомогою різних технік та найкращих практик. Дослідники та практики повинні ретельно розглянути характер даних, вибрати правильний алгоритм та метрики оцінки для забезпечення точних та надійних результатів.

1.1.4 Точки зростання та потенційні застосування:

Багатокласова класифікація - це завдання машинного навчання, яке включає класифікацію екземплярів у один з трьох або більше класів. Ось деякі потенційні області застосування багатокласової класифікації та вплив розумних систем:

- Розпізнавання зображень та об'єктів: Багатокласова класифікація може бути використана для розпізнавання об'єктів на зображеннях, наприклад, ідентифікації різних типів транспортних засобів, тварин або рослин. Це застосовується в галузях, таких як автономні транспортні засоби, сільське господарство та медицина.

- Обробка природної мови: Багатокласова класифікація може бути використана для класифікації тексту у різні категорії, такі як аналіз настрою, моделювання теми та фільтрація спаму. Це застосовується в галузях, таких як обслуговування клієнтів, маркетинг та кібербезпека.

- Медична діагностика: Багатокласова класифікація може бути використана для діагностики захворювань на основі симптомів пацієнта, наприклад, ідентифікації різних типів раку або неврологічних розладів. Це застосовується в галузях, таких як охорона здоров'я та біотехнології.

- Фінансовий аналіз: Багатокласова класифікація може бути використана для прогнозування цін на акції або класифікації кредитного ризику на основі фінансових даних. Це застосовується в галузях, таких як фінанси та інвестиційний банкінг.

Розумні системи можуть допомогти в цих областях:

- Покращення точності: Розумні системи можуть використовувати складні алгоритми для аналізу даних та зробити більш точні прогнози, ніж люди. Це може призвести до кращих результатів в галузях, таких як охорона здоров'я та фінанси.

- Зменшення робочого навантаження: Розумні системи можуть автоматизувати повторювані завдання, такі як введення даних або розпізнавання зображень, звільняючи людей від виконання більш складних завдань. Це може призвести до збільшення ефективності та продуктивності в галузях, таких як виробництво та логістика.

- Забезпечення аналізу в реальному часі: Розумні системи можуть аналізувати дані в режимі реального часу, що дозволяє швидше приймати рішення та реагувати. Це може призвести до покращення безпеки і захисту в галузях, таких як транспорт та кібербезпека.

В цілому, багатокласова класифікація та розумні системи мають потенціал вплинути на широкий спектр галузей та поліпшити результати в багатьох різних областях.

1.2 Огляд існуючих методів багатокласової класифікації

1.2.1 Опис базових методів багатокласової класифікації

Багатокласова класифікація є типом задачі машинного навчання, де метою є призначення вхідних даних одній з кількох можливих категорій. Існує кілька методів багатокласової класифікації, включаючи Один проти всіх, Один проти одного та Матрицю ймовірностей. Наведемо огляд кожного методу та їх переваг і недоліків:

1. Один проти всіх (також відомий як Один проти решти): Цей метод передбачає тренування окремого бінарного класифікатора для кожного класу, зразки цього класу використовуються як позитивні приклади, а всі інші зразки - як негативні. Під час передбачення вибирається класифікатор з найвищим виходом як передбачуваний клас. Один проти всіх простий у реалізації, швидкий у тренуванні і може працювати з несбалансованими даними. Однак він може показувати погані результати, коли класи сильно перекриваються або коли класів багато.

2. Один проти одного: Цей метод передбачає тренування бінарного класифікатора для кожної пари класів, зразки цих класів використовуються як позитивні приклади, а всі інші зразки - як негативні. Під час передбачення кожен класифікатор голосує за свій передбачуваний клас, і клас з найбільшою кількістю голосів вибирається як передбачуваний клас. Один проти одного вимагає більше обчислювальних ресурсів, ніж Один проти всіх, але краще впорається з перекриваючимися класами. Однак він може показувати погані результати, коли класів багато.

3. Матриця ймовірностей: Цей метод передбачає тренування багатокласового класифікатора, який видає розподіл ймовірностей для всіх можливих класів. Під час передбачення вибирається клас з найвищою ймовірністю як передбачуваний клас. Матриця ймовірностей проста у реалізації і може працювати з будь-якою кількістю класів. Однак вона може показувати погані результати, коли класи сильно перекриваються або коли дані несбалансовані.

Кожен метод має свої переваги й недоліки, і вибір методу залежить від конкретної проблеми та наявних даних. Крім того, існує багато інших методів багатокласової класифікації, таких як Коригуючі коди виходу, Пряма багатоетикеткова класифікація та Ієрархічна класифікація, які можуть бути більш підходящими для певних задач.

Оцінка продуктивності моделі багатокласової класифікації включає використання різних метрик, таких як Точність, Повнота, F1-оцінка та Матриця помилок. Точність - це співвідношення правильних позитивних результатів до загальної кількості позитивних передбачень, Повнота - це співвідношення правильних позитивних результатів до загальної кількості фактичних позитивів, а F1-оцінка - це гармонічне середнє між Точністю та Повнотою. Матриця помилок - це табличне представлення передбачень моделі порівняно з фактичними мітками.

Однак, варто зазначити, що результати кожного алгоритму можуть змінюватися в залежності від конкретної проблеми та наявних даних. При

роботі з задачами багатокласової класифікації важливо належним чином попередньо обробляти дані, включаючи обробку пропущених значень, масштабування ознак та кодування категоріальних змінних. Крім того, відбір та інженерія ознак також можуть покращити продуктивність моделі. Важливо також обрати відповідну метрику оцінки, яка підходить для конкретної проблеми.

1.2.2 Ансамблеві методи багатокласової класифікації:

Методи ансамблю - це техніка комбінування кількох моделей машинного навчання для покращення точності та надійності багатокласової класифікації. Існують кілька методів ансамблю, що використовуються для багатокласової класифікації, таких як методи на основі багатьох класифікаторів, стекінг, об'єднання класифікаторів і т.д.

Ось список речей, які варто враховувати при дослідженні методів ансамблю для багатокласової класифікації:

- Методи ансамблю комбінують передбачення кількох класифікаторів для покращення точності багатокласової класифікації.

- Беггінг є ефективним та потужним підходом до ансамблю для багатокласової класифікації. У беггінгу кілька класифікаторів навчаються на різних підмножинах навчальних даних, а їх передбачення поєднуються для отримання остаточного передбачення.

- Рекурсивні методи ансамблю можуть бути використані для багатокласової класифікації з неузгодженістю даних і також вирішують проблему неузгодженості даних у регресії. Ці методи досягають високої продуктивності у багатокласовій класифікації з неузгодженістю класів та аналізі регресії на скошених або неузгоджених даних.

- Парні з'єднані бінарні та багатокласові класифікатори можуть бути використані в ансамблі для досягнення високих рівнів точності. Крім того, тренування цих класифікаторів за допомогою певного піднабору атрибутів

може зменшити обчислювальні витрати ансамблю та покращити його продуктивність.

- Новаторським підходом до комбінування класифікаторів одного класу для вирішення багатокласових проблем є метод динамічного вибору ансамблю. Цей підхід дозволяє відкидати некомпетентні класифікатори для покращення надійності фази комбінування. Розглядається сусідство кожного екземпляра, щоб визначити, чи може класифікатор бути компетентним чи ні.

Підсумовуючи, методи ансамблю є потужною технікою для покращення точності та надійності багатокласової класифікації. Беггінг, рекурсивні методи ансамблю, парні з'єднані бінарні та багатокласові класифікатори та динамічний вибір ансамблю - це лише деякі підходи, які використовуються для досягнення високих рівнів точності у багатокласовій класифікації.

1.2.3 Напівнаглядне навчання для багатокласової класифікації:

Півнаглядоване навчання є типом машинного навчання, який використовує як позначені, так і непозначені дані для покращення продуктивності моделі. Для багатокласової класифікації, методи півнаглядованого навчання можуть бути використані для підвищення точності завдання класифікації.

- Більшість алгоритмів півнаглядованого навчання призначені для бінарної класифікації, але їх можна розширити до багатокласової класифікації за допомогою підходів, таких як один проти решти. Однак ці підходи можуть не використовувати можливість того, що кожен приклад призначається лише одному класу.

- Півнаглядоване навчання може допомогти зменшити залежність від позначених даних, дозволяючи використовувати непозначені дані для покращення продуктивності моделі. У деяких випадках отримання позначених

даних може бути складним або дорогим, тому півнаглядоване навчання може бути корисним підходом.

- Класифікація з півнаглядованим навчанням (SSC) спрямована на покращення контрольованої класифікації шляхом мінімізації помилок у позначених прикладах, зберігаючи сумісність з розподілом вхідних даних непозначених екземплярів. SSC можна розділити на дві конфігурації: трансдуктивне та індуктивне навчання. Трансдуктивне навчання передбачає визначення позначок непозначених прикладів, враховуючи як позначені, так і непозначені дані, тоді як індуктивне навчання передбачає прогнозування невидимих даних, використовуючи позначені та непозначені дані як тренувальні приклади.

- Методи півнаглядованого навчання на основі графіків часто використовуються для реалізації припущення про множину, яке стверджує, що дані наближено лежать на множині нижчої розмірності, ніж простір вхідних даних. Ці методи передбачають побудову графіка, представленого ваговою матрицею, та використання трансдуктивного висновку для передачі відомих позначок для прогнозування значень усіх непозначених вершин.

- Самопозначені техніки є типом методів SSC, які можна використовувати для отримання одного або декількох розширених позначених наборів на основі найвпевненіших прогнозів для класифікації непозначених даних. Ці техніки не роблять жодних конкретних припущень про вхідні дані та використовують непозначені дані у контрольованому контексті через процес самонавчання.

- Екстремальне півнаглядове навчання (ESSL) - це новий метод, який може використовуватися для вирішення проблем з півнаглядовими методами, зокрема півнаглядовими методами на основі опорних векторів (S3VM), для багатокласової класифікації. ESSL використовує рамки методу екстремального навчання (ELM), щоб вирішити як бінарні, так і багатокласові проблеми класифікації в єдиної моделі. Прихований шар кодується за допомогою надзвичайно малої наближеної емпіричної ядерної карти (AEKM), щоб

зменшити обчислювальні витрати та використання пам'яті для навчання та тестування. ESSL може бути ефективно та ефективно вирішено на основі чергової оптимізації (AO) і показано, що перевершує існуючі ефективні методи півнаглядового навчання.

1.2.4 Бустінг для багатокласової класифікації

Boosting - це популярний метод ансамблювання, який може бути використаний для багатокласової класифікації. Ось кілька алгоритмів boosting, які можуть бути застосовані в контексті багатокласової класифікації:

- AdaBoost: AdaBoost - популярний алгоритм boosting, який може бути використаний для багатокласової класифікації. Він працює шляхом ітеративного навчання слабких класифікаторів на наборі даних та присвоєння вищої ваги помилково класифікованим екземплярам у кожній ітерації. Остаточне передбачення робиться шляхом комбінування передбачень всіх слабких класифікаторів за допомогою їх зваженої суми.

- Gradient Boosting: Gradient Boosting - інший популярний алгоритм boosting, який може бути використаний для багатокласової класифікації. Він працює шляхом ітеративного навчання дерев прийняття рішень для вирішення від'ємного градієнту функції втрат. Остаточне передбачення робиться шляхом комбінування передбачень всіх дерев прийняття рішень за допомогою їх зваженої суми.

- XGBoost: XGBoost - це оптимізована версія Gradient Boosting, яка використовує більш регуляризовану модель для контролю перенавчання і може забезпечити кращу точність і швидкість. Він може бути використаний для багатокласової класифікації, розширивши цільову функцію для підтримки декількох класів.

- LightGBM: LightGBM - це ще одна оптимізована версія Gradient Boosting, яка використовує алгоритми на основі гістограм для прискорення навчання та зменшення використання пам'яті. Він може бути використаний для

багатокласової класифікації, розширивши цільову функцію для підтримки декількох класів.

Регуляризований boosting в свою чергу- техніка, що використовується в ансамбльному навчанні для покращення продуктивності моделей gradient boosting. Вона передбачає додавання регуляризаційних членів до цільової функції, яка оптимізується під час процесу навчання, щоб запобігти перенавчанню та покращити узагальнення. Регуляризацію можна досягти за допомогою різних технік, таких як зменшення, субвибірка та субвибірка ознак.

У gradient boosted trees може виникати перенавчання, і для його запобігання можна застосовувати регуляризацію. Це можна зробити за допомогою валідаційного набору даних та раннього зупинення, а також шляхом налаштування гіперпараметрів, таких як швидкість навчання, кількість ітерацій та глибина дерев.

У gradient boosting швидкість навчання часто називається параметром регуляризації, оскільки вона контролює внесок кожного дерева у кінцеву модель. Менша швидкість навчання означає, що кожне дерево має менший вплив на кінцеву модель, що може покращити узагальнення та запобігти перенавчанню.

1.3 Огляд регуляризованого бустінгу

1.3.1 Розширене пояснення регуляризованого бустінгу:

Регуляризований бустинг є варіантом стандартного бустингу, який вводить функцію штрафу до функції витрат, що оптимізується на кожній ітерації. Ця функція штрафу використовується для обмеження складності моделі та запобігання перенавчанню. Регуляризований бустинг можна розглядати як компроміс між компромісом між зміщеністю та розсіяністю у стандартному бустингу.

Основні концепції в регуляризованому бустингу:

- Функції штрафу: Це функції, які додаються до функції витрат на кожній ітерації, щоб обмежити складність моделі та запобігти перенавчанню. Приклади функцій штрафу включають L1 та L2 регуляризацію.

- Параметри регуляризації: Це гіперпараметри, які контролюють силу застосовуваної функції штрафу. Вище значення параметра регуляризації призводить до більш обмеженої моделі та меншого перенавчання, але також вищого зміщення.

- Зменшення: Це техніка, яка використовується для сповільнення внеску кожного дерева в кінцеву модель. Вона полягає в множенні виходу кожного дерева на невеликий коефіцієнт навчання перед додаванням його до кінцевої моделі. Це зменшує розсіяність моделі та допомагає запобігти перенавчанню.

- Крос-валідація: Це техніка, яка використовується для налаштування гіперпараметрів моделі. Вона включає розбиття даних на навчальний та валідаційний набори, а також оцінку продуктивності моделі на валідаційному наборі для різних значень гіперпараметрів. Вибираються значення гіперпараметрів, які дають найкращу продуктивність на валідаційному наборі.

Регуляризований бустинг відрізняється від стандартного бустингу наступними способами:

- До функції витрат на кожній ітерації додаються функції штрафу для обмеження складності моделі та запобігання перенавчанню.

- Використовуються параметри регуляризації для контролю сили застосовуваної функції штрафу.

- Застосовується зменшення для сповільнення внеску кожного дерева в кінцеву модель та запобігання перенавчанню.

- Використовується крос-валідація для налаштування гіперпараметрів моделі.

Деякі популярні реалізації регуляризованого boostingy включають XGBoost та LightGBM. Ці реалізації надають різні гіперпараметри, які можна налаштувати для контролю сили функції штрафу та запобігання перенавчанню.

Наприклад, XGBoost надає гіперпараметри, такі як "lambda" та "alpha" для L1 та L2 регуляризації, та "eta" для зменшення.

1.3.2 Види регуляризованого бустінгу

Бустинг - це метод ансамблювання, який поєднує кілька слабких класифікаторів в один потужний класифікатор. Бустинг з регуляризацією вводить обмеження або штрафи у процес тренування для запобігання перенавчанню.

Ось деякі варіації бустингу з регуляризацією, які використовуються у багатокласовій класифікації:

- AdaBoost.R2: Цей алгоритм додає регуляризаційний член до функції втрат для контролю складності остаточної моделі. Він базується на алгоритмі AdaBoost, який надає вищу вагу неправильно класифікованим зразкам на кожній ітерації. AdaBoost.R2 підходить для задач з шумними або неповними даними.

- LogitBoost: Цей алгоритм застосовує модель логістичної регресії до виходу кожного слабкого класифікатора і оновлює ваги тренувальних зразків на основі втрати логарифмічної правдоподібності. LogitBoost може працювати з незбалансованими даними та є стійким до викидів.

- GentleBoost: Цей алгоритм використовує квадратичну функцію втрат для оновлення ваг тренувальних зразків і налаштовує швидкість навчання на основі градієнту втрати. GentleBoost менш чутливий до викидів, ніж інші алгоритми бустингу.

Особливості та вплив цих алгоритмів бустингу з регуляризацією на точність класифікації наступні:

- AdaBoost.R2: Регуляризаційний член в AdaBoost.R2 може запобігати перенавчанню та покращувати загальну продуктивність моделі. Однак, він також може знизити точність моделі, якщо регуляризаційний параметр встановлений занадто високо.

- LogitBoost: Модель логістичної регресії в LogitBoost може виявляти складні взаємозв'язки між ознаками та покращувати інтерпретованість моделі. Однак, це може збільшити обчислювальні витрати та потребувати більше тренувальних зразків, ніж інші алгоритми.

- GentleBoost: Квадратична функція втрати в GentleBoost може обробляти нелінійні зв'язки між ознаками та зменшувати вплив викидів. Однак, це також може збільшити ризик перенавчання, якщо швидкість навчання встановлена занадто високо.

1.3.3 Використання регуляризованого бустінгу в контексті багатокласової класифікації:

Багатокласові алгоритми класифікації, зокрема регуляризований бустинг, надають кілька нових можливостей у галузі машинного навчання. Ось деякі переваги:

- Більш точні прогнози: Регуляризований бустинг може покращити точність прогнозування для проблем багатокласової класифікації. Це через те, що він є ансамблевим методом, який шляхом ітеративного додавання моделей одну на одну виправляє помилки попередньої моделі, створює потужний класифікатор. Цей підхід може привести до кращих результатів, ніж використання однієї моделі.

- Робота з несбалансованими наборами даних: Проблеми багатокласової класифікації з несбалансованими наборами даних можуть бути складними. Регуляризований бустинг може бути використаний для вирішення цієї проблеми шляхом додавання регуляризаційних параметрів до моделі. Це може допомогти уникнути перенавчання і покращити продуктивність моделі.

- Пряма багатоетикеткова класифікація: Регуляризований бустинг може бути адаптований для прямої багатоетикеткової класифікації, що означає, що проблема вирішується в повному обсязі. Це може бути ефективніше, ніж перетворення проблеми багатоетикеткової класифікації на набір проблем бінарної класифікації, що є поширеним підходом в літературі.

- Налаштування моделі: Регуляризований бустинг пропонує кілька продвинутих можливостей для налаштування моделі, таких як налаштування мінімальної кількості вибірок, яку вузол може представляти для подальшого розбиття (min_child_weight) та налаштування максимальної глибини дерева для уникнення перенавчання (max_depth). Інші параметри, які можна налаштувати, включають gamma, subsample, colsample_bytree та швидкість навчання.

- Робота з структурованими даними: Хоча штучні нейронні мережі, як правило, показують кращі результати для задач прогнозування з

неструктурованими даними (зображення, текст і т.д.), алгоритми на основі рішень, такі як регуляризований бустинг, вважаються найкращими для структурованих/табличних даних невеликого та середнього розміру.

Можемо сказати, що регуляризований бустинг є потужним алгоритмом для проблем багатокласової класифікації, який може впоратися з несбалансованими наборами даних, прямо виконувати багатоетикеткову класифікацію та покращувати точність прогнозів. Він також надає продвинуті можливості для налаштування моделі і ефективний для структурованих/табличних даних невеликого та середнього розміру.

1.3.4 Важливість регуляризованого бустінгу:

Регуляризований бустинг є ефективним підходом для багатокласової класифікації, який має різноманітні переваги та виклики. Регуляризований бустинг - це поєднання бустингу та регуляризації, яке допомагає запобігти перенавчанню шляхом додавання штрафного члена до функції втрат. Ось деякі переваги використання регуляризованого бустингу для багатокласової класифікації:

- Регуляризований бустинг може працювати з високо-вимірними даними, що робить його корисним для завдань багатокласової класифікації, що включають великі набори даних.

- Регуляризований бустинг може зменшити перенавчання та покращити узагальнюючу здатність моделі за рахунок додавання штрафного члена до функції втрат.

- Регуляризований бустинг може працювати з пропущеними даними та викидами, які є поширеними у реальних наборах даних.

- Регуляризований бустинг може обробляти нелінійні зв'язки між ознаками та цільовою змінною, що робить його підходящим для складних завдань багатокласової класифікації.

Однак, також існують деякі виклики, пов'язані з використанням регуляризованого бустингу для багатокласової класифікації. Ось деякі з цих викликів:

- Регуляризований бустинг може бути обчислювально витратним, особливо для великих наборів даних та складних моделей.

- Регуляризований бустинг вимагає уважного налаштування гіперпараметрів, таких як швидкість навчання, кількість ітерацій та параметр регуляризації, для досягнення оптимальної продуктивності.

- Регуляризований бустинг може бути чутливим до вибору функції втрат, що може впливати на продуктивність моделі.

- Регуляризований бустинг може бути чутливим до вибору базового навчального алгоритму, що може впливати на продуктивність моделі.

Для вирішення цих викликів важливо уважно вибирати гіперпараметри, функцію втрат та базовий навчальний алгоритм для досягнення оптимальної продуктивності. Також важливо використовувати техніки, такі як крос-валідація та раннє зупинення, для запобігання перенавчанню та покращення узагальнюючої здатності моделі.

Регуляризований бустинг є привабливим підходом для багатокласової класифікації, оскільки він має кілька переваг перед іншими методами. Ось деякі характеристики, які роблять регуляризований бустинг ефективним для вирішення завдань багатокласової класифікації:

- Регуляризований бустинг є ансамблевим методом, який поєднує кілька слабких навчальних алгоритмів для створення потужного навчального алгоритму. Це дозволяє моделі виявляти складні залежності між ознаками та цільовою змінною, що має важливе значення для багатокласової класифікації.

- Регуляризований бустинг використовує процес градієнтного спуску для мінімізації функції втрат, що дозволяє моделі вчитися на своїх помилках та покращувати свою продуктивність з часом.

- Регуляризований бустинг може працювати як з категоріальними, так і з числовими даними, що робить його корисним для широкого спектру завдань багатокласової класифікації.

- Регуляризований бустинг може працювати з незбалансованими класами, що є поширеним у завданнях багатокласової класифікації, де деякі класи мають більше екземплярів, ніж інші.

1.4 Висновки до розділу 1

В цьому розділі було проведено огляд проблеми багатокласової класифікації та регуляризованого бустінгу. Загальний огляд дозволив описати домен застосування даної системи, а також виявити значення багатокласової класифікації. Виклики та складнощі, пов'язані з цією проблемою, були визначені, а також розглянуто потенційні застосування. Детально проаналізовано існуючі методи багатокласової класифікації, зосереджуючись на базових методах, ансамблевих методах, напівнаглядному навчанні та бустінгу. Також проведено огляд регуляризованого бустінгу, включаючи розширене пояснення, види та використання в контексті багатокласової класифікації. Результати цього розділу вказують на важливість регуляризованого бустінгу як потенційно ефективного підходу до багатокласової класифікації.

2 МАТЕМАТИЧНІ ОСНОВИ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ БАГАТОКЛАСОВОЇ КЛАСИФІКАЦІЇ НА ОСНОВІ РЕГУЛЯРИЗОВАНОГО БУСТІНГУ

2.1 Огляд алгоритмів багатокласової класифікації

Алгоритм "Один проти решти"

Алгоритм "Один проти решти" (One-vs-Rest, OvR) є популярним підходом до багатокласової класифікації. Він передбачає тренування K бінарних класифікаторів, де K - кількість класів, для розрізнення кожного класу від інших. (рис. 2.1)

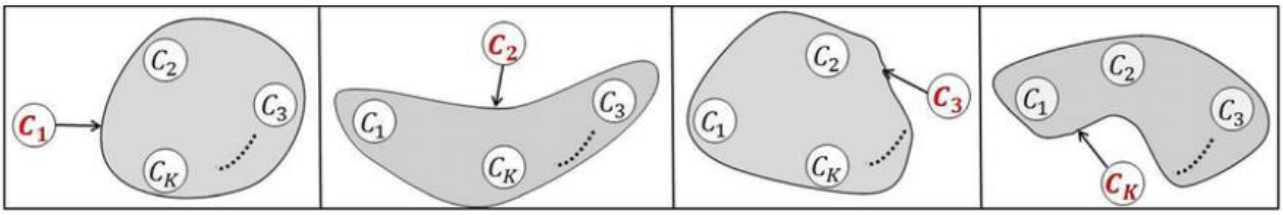

Рисунок 2.1 - Графічне зображення алгоритму "One-vs-Rest"

Математичне формулювання алгоритму OvR:

Припустимо, що ми маємо навчальний набір даних, що складається з N зразків та D ознак, позначених як $X = \{x_1, x_2, ..., xN\}$, а їх відповідні мітки класів - $Y = \{y_1, y_2, ..., yN\}$, де $y_i \in \{0, 1, ..., K\text{-}1\}$ представляє мітку класу зразка x_i (з K класами).

Для кожного класу $k \in \{0, 1, ..., K\text{-}1\}$ ми створюємо бінарний класифікатор $f_k(X)$, який навчається розрізняти клас k від інших. Бінарний класифікатор призначає ймовірнісний бал, що вказує на ймовірність того, що зразок належить класу k.

Алгоритм OvR передбачає тренування K бінарних класифікаторів з використанням такої формулювання:

1. Перетворення задачі багатокласової класифікації на K задач бінарної класифікації:

- Для кожного класу k створюємо бінарний цільовий вектор t_k довжиною N.
- Встановлюємо t_k [i] = 1, якщо зразок xi належить класу k, інакше t_k [i] = 0.

2. Тренування K бінарних класифікаторів:
   - Для кожного класу k тренуємо бінарний класифікатор $f_k(X)$ з використанням бінарних цільових векторів t_k.
   - Бінарний класифікатор може бути будь-яким підходящим алгоритмом, таким як логістична регресія, метод опорних векторів або дерева рішень.
3. Класифікація нових зразків:
   - Для класифікації нового зразка x застосовуємо всі K бінарних класифікаторів $f_k(X)$ та отримуємо їх ймовірнісні бали.
   - Призначаємо мітку класу з найвищим ймовірнісним балом як передбачений клас для зразка x.

Алгоритм OvR дозволяє розширити бінарні класифікатори для вирішення задач багатокласової класифікації, розкладаючи їх на кілька задач бінарної класифікації. Кожен бінарний класифікатор навчається розрізняти один клас від інших, що дозволяє класифікувати зразки за декількома класами.

Алгоритм "Один проти одного"

Алгоритм "Один проти одного" (One-vs-One, OvO) є ще одним підходом до багатокласової класифікації. Він передбачає тренування K(K-1)/2 бінарних класифікаторів, де K - кількість класів, для розрізнення кожної пари класів. (рис. 2.2)

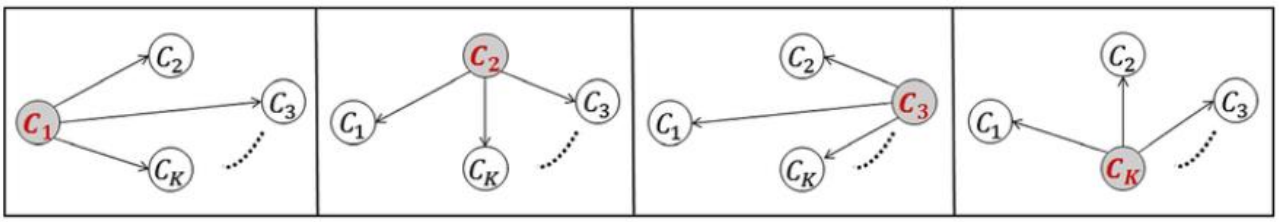

Рисунок 2.2 - Графічне зображення алгоритму "One-vs-One"

Математичне формулювання алгоритму OvO:

Припустимо, що ми маємо навчальний набір даних, що складається з N зразків та D ознак, позначених як X = {x_1, x_2, ..., xN}, а їх відповідні мітки класів - Y = {y_1, y_2, ..., yN}, де $y_i \in \{0, 1, ..., K-1\}$ представляє мітку класу зразка x_i(з K класами).

Для алгоритму OvO ми створюємо бінарний класифікатор для кожної пари класів (k, l), де k та l є відмінними класами. Бінарний класифікатор для класів k та l позначається як f_{kl}(X).

Алгоритм OvO передбачає тренування K(K-1)/2 бінарних класифікаторів за такою формулюванням:

1. Перетворення задачі багатокласової класифікації на K(K-1)/2 задач бінарної класифікації:

- Для кожної пари відмінних класів (k, l) створюємо бінарний цільовий вектор tlk довжиною N.

- Встановлюємо t_{lk}[i] = 1, якщо зразок xi належить класу l, t_{lk} [i] = -1, якщо він належить класу k, і t_{lk} [i] = 0 в інших випадках.

2. Тренування K(K-1)/2 бінарних класифікаторів:

- Для кожної пари класів (k, l) тренуємо бінарний класифікатор f_{kl} (X) з використанням бінарних цільових векторівt_{lk}.

- Знову ж, можна використовувати будь-який підходящий алгоритм бінарної класифікації, такий як логістична регресія, метод опорних векторів або дерева рішень.

3. Класифікація нових зразків:

- Для класифікації нового зразка x застосовуємо всі K(K-1)/2 бінарних класифікаторів f_{kl} (X) та отримуємо їх прогнози.

- Кожен бінарний класифікатор проголосує за один з класів на основі свого прогнозу.

- Клас з найбільшою кількістю голосів призначається як передбачений клас для зразка x.

Алгоритм OvO порівнює всі пари класів та використовує систему голосування для визначення кінцевого класу для заданого зразка. Він вирішує задачу багатокласової класифікації, тренуючи кілька бінарних класифікаторів, кожен з яких спеціалізується на розрізненні конкретної пари класів.

Алгоритм "Матриця ймовірностей"

Алгоритм "Матриця ймовірностей" є методом багатокласової класифікації, який використовує ймовірнісну матрицю для здійснення прогнозів.

Маючи навчальний набір даних, що складається з N зразків та D ознак, позначених як X = {x_1, x_2, ..., xN}, і відповідні мітки класів Y = {y_1, y_2, ..., yN}, де $y_i \in$ {0, 1, ..., K-1} представляє мітку класу для зразка x_i(з K класами).

Алгоритм "Матриця ймовірностей" включає такі кроки:

1. Обчислення ймовірностей класів:

- Для кожного класу k ∈ {0, 1, ..., K-1} обчислюємо ймовірність класу P(y=k|X) за допомогою відповідного алгоритму класифікації (наприклад, логістична регресія, softmax-регресія або Наївний Баєс).

- Ймовірність класу P(y=k|X) відображає ймовірність того, що зразок належить класу k на основі вказаних ознак X.

2. Побудова матриці ймовірностей:

- Створюємо матрицю ймовірностей P розміром N × K, де N - кількість зразків, а K - кількість класів.

- Заповнюємо матрицю ймовірностей P, присвоюючи обчислені ймовірності класів P(y=k|X) відповідним елементам P[i][k], де i - індекс зразка, а k - мітка класу.

3. Класифікація нових зразків:

- Для класифікації нового зразка x обчислюємо вектор ознак для x.

- Розраховуємо ймовірності класів P(y=k|x) за допомогою навченої моделі або матриці ймовірностей P.

- Призначаємо мітку класу з найвищою ймовірністю як передбачуваний клас для зразка x.

Алгоритм "Матриця ймовірностей" надає спосіб оцінки ймовірностей класів для кожного зразка в наборі даних і будує матрицю ймовірностей, яка відображає ймовірності кожного класу для кожного зразка. Ця матриця потім використовується для здійснення прогнозів для нових зразків на основі обчислених ймовірностей.

Метод дерева класифікації

Математична формулювання методу дерева класифікації, також відомого як метод Дерева прийняття рішень, для багатокласової класифікації наступне:

З урахуванням навчального набору даних, що складається з N зразків та D ознак, позначених як $X = \{x_1, x_2, ..., xN\}$, і відповідних міток класів $Y = \{y_1, y_2, ..., yN\}$, де $y_i \in \{0, 1, ..., K-1\}$ відображає мітку класу для зразка xi (з K класами).

Метод дерева класифікації включає такі кроки:

1. Побудова дерева прийняття рішень:

- Починаючи з кореневого вузла, рекурсивно розділяємо навчальний набір даних на основі значень ознак для створення дочірніх вузлів.

- Розділення здійснюється шляхом вибору ознаки, яка оптимально розділяє зразки на різні класи. Зазвичай це робиться за допомогою мір нечистоти, таких як індекс Джині або ентропія.

- Продовжуємо розділяти вузли до досягнення критерію зупинки, такого як досягнення максимальної глибини або мінімальної кількості зразків у кожному вузлі.

2. Призначення міток класу:

- Після побудови дерева прийняття рішень, призначаємо мітки класу листовим вузлам на основі більшості зразків в кожному листовому вузлі.

- Якщо в листовому вузлі є зразки, які належать до кількох класів, можуть використовуватись додаткові критерії, такі як ймовірності або відстані, для визначення кінцевого призначення класу.

3. Класифікація нових зразків:

- Для класифікації нового зразка x, пройдіть по дереву прийняття рішень від кореневого вузла до листового вузла на основі значень ознак зразка.

- Призначаємо мітку класу, яка пов'язана з досягнутим листовим вузлом, як передбачуваний клас для зразка x.

Метод Класифікаційного дерева використовує структуру дерева для представлення правил прийняття рішень на основі ознак зразків. Він рекурсивно розділяє навчальний набір даних, створюючи вузли прийняття рішень, та призначає мітки класу листовим вузлам. Під час класифікації відбувається просування по дереву прийняття рішень для визначення відповідної мітки класу на основі значень ознак зразка.

Ансамблеві методи

Ансамблеві методи є потужними техніками для багатокласової класифікації, які комбінують прогнози з кількох базових класифікаторів для поліпшення загальної продуктивності.

З урахуванням навчального набору даних, що складається з N зразків та D ознак, позначених як $X = \{x_1, x_2, ..., xN\}$, і відповідних міток класів

$Y = \{y_1, y_2, ..., yN\}$, де $y_i \in \{0, 1, ..., K-1\}$ відображає мітку класу для зразка x_i (з K класами).

Ансамблеві методи зазвичай включають такі кроки:

1. Навчання базових класифікаторів:

- Створіть набір T базових класифікаторів, позначених як $\{f_1, f_2, ..., fT\}$.

- Кожен базовий класифікатор навчається на піднаборі навчального набору даних або з використанням різних алгоритмів навчання для отримання різноманітності.

2. Комбінування прогнозів:

- Для кожного нового зразка x отримайте прогнози кожного базового класифікатора: $f_1(x), f_2(x), ..., fT(x)$.

- Залежно від ансамблевого методу, комбінуйте прогнози за допомогою різних технік, таких як голосування, усереднення або зважене усереднення.

3. Прийняття ансамбльного рішення:

- На основі комбінованих прогнозів зробіть остаточний прогноз для мітки класу зразка x.

- Правило прийняття може бути більшістю голосів, зваженим голосуванням або іншими стратегіями агрегації.

Деякі популярні ансамблеві методи включають:

- Беггінг: Кожен базовий класифікатор навчається на збірці даних, створеній шляхом вибору заміщення з навчального набору даних. Остаточний прогноз отримується шляхом агрегації прогнозів всіх базових класифікаторів.

- Випадковий ліс: Це розширення беггінгу, яке використовує рішучі дерева як базові класифікатори. Остаточний прогноз заснований на більшості голосів прогнозів окремих дерев.

- Бустінг: Базові класифікатори навчаються послідовно, при цьому кожен класифікатор надає більший ваговий коефіцієнт помилково класифікованим зразкам попередніх класифікаторів. Остаточний прогноз отримується шляхом комбінування зважених прогнозів всіх базових класифікаторів.

- Адабуст: Це конкретний алгоритм бустінгу, який приділяє вищу вагу помилково класифікованим зразкам під час навчання.

- Градієнтний бустінг: Він поєднує слабкі класифікатори поетапно, при цьому кожен новий класифікатор навчається для виправлення помилок попередніх класифікаторів.

Ці ансамблеві методи прагнуть використовувати різноманітність базових класифікаторів та їх колективне прийняття рішень для поліпшення точності та надійності багатокласової класифікації.

2.2 Регуляризований бустінг

2.2.1 Огляд методів бустингу

Алгоримт AdaBoost

Для побудови слабкого класифікатора в AdaBoost можна використовувати рішучі дерева з одним рівнем, так звані рішучі пеньки. Вага кожного екземпляра в навчальному наборі спочатку встановлюється на 1/n, де n - це кількість навчальних екземплярів. Потім алгоритм ітеративно навчає слабкі класифікатори на зважених версіях даних і оновлює ваги екземплярів на основі помилки класифікації слабкого класифікатора. Остаточний потужний класифікатор отримується шляхом комбінування слабких класифікаторів за допомогою зваженої суми, де ваги пропорційні точності слабких класифікаторів.

Математична формулювання алгоритму AdaBoost може бути виражена наступним чином:

Нехай маємо навчальний набір даних {(x1, y1), (x2, y2), ..., (xN, yN)}, де x_i - і-й екземпляр, а y_i- його відповідна мітка класу, і набір слабких класифікаторів {h1, h2, ..., hT}, де T - кількість ітерацій:

1. Ініціалізуємо ваги екземплярів як $w_i = 1/n$.
2. Для t = 1 до T:
- Навчаємо слабкий класифікатор h_tна зваженому наборі даних.
- Обчислюємо помилковість εt класифікатора h_t на зваженому наборі даних.
- Обчислюємо вагу αt класифікатора h_t, за формулою 2.1

$$\alpha_t = 1/2 * ln(\frac{1-\varepsilon_t}{\varepsilon_t}) \quad (2.1)$$

- Оновлюємо ваги екземплярів, за формулою 2.2

$$w_i = w_i * exp(-\alpha_t * y_i * h_t(x_i)) / Z_t, \quad (2.2)$$

де Zt - нормалізаційний фактор.

Остаточний потужний класифікатор отримуємо, за формулою 2.3

$$H(x) = sign(sum(\alpha_t * h_t(x))) \quad (2.3)$$

У цьому формулюванні вага α_t кожного слабкого класифікатора пропорційна його точності і обернено пропорційна його помилковості. Вага кожного екземпляра оновлюється на основі помилки класифікації слабкого класифікатора і ваги α_t. Нормалізаційний фактор Z_t забезпечує, що сума ваг екземплярів становить 1 після кожної ітерації.

Важливо відзначити, що спочатку алгоритм AdaBoost був розроблений для задач бінарної класифікації, але може бути розширений на багатокласові задачі за допомогою методів, таких як "Один проти всіх". Крім того, AdaBoost може бути використаний з будь-яким алгоритмом машинного навчання, якщо він є слабким класифікатором, тобто досягає точності, трохи вищої, ніж випадковий вибір при класифікації.

Алгоритм Gradient Boosting

Математична формулювання алгоритму градієнтного бустінгу:

Нехай маємо навчальний набір даних {(x1, y1), (x2, y2), ..., (xN, yN)}, де x_i - і-й екземпляр, а y_i - його відповідна мітка класу.

1. Ініціалізуємо початковий прогноз $F_0(x)$ як константне значення, яке може бути, наприклад, середнім значенням міток класів у навчальному наборі.

2. Для m = 1 до M, де M - кількість ітерацій:

- Обчислюємо псевдо-остачу r_{im} для кожного екземпляра, за формулою 2.4

$$r_{im} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}, \quad (2.4)$$

де L - функція втрат, y_i - справжнє значення мітки класу, $F(x_i)$ - прогнозна функція на екземплярі x_i.

- Навчаємо слабкий модель $h_m(x)$, яка намагається апроксимувати псевдо-остачу r_{im}, на навчальному наборі.

- Обчислюємо коефіцієнт γ_m, який визначає, наскільки ми оновлюємо прогнозну функцію F(x) на кожній ітерації.

- Оновлюємо прогнозну функцію $F_{m-1}(x)$, за формулою 2.5

$$F_{\mathrm{m}}(x) = F_{\mathrm{m-1}}(x) + \gamma_{\mathrm{m}} * h_{\mathrm{m}}(x). \quad (2.5)$$

3. Отримуємо остаточний прогнозний класифікатор F(x) після M ітерацій.

У цій формулюванні градієнтного бустінгу, псевдо-остача r_{im} обчислюється за допомогою похідної функції втрат L відносно прогнозної функції F(x). Слабка модель $h_m(x)$ навчається апроксимувати цю псевдо-остачу на навчальному наборі. Коефіцієнт γ_m визначає, наскільки сильно ми оновлюємо прогнозну функцію на кожній ітерації. Загальний прогноз F(x) отримується шляхом додавання поетапних оновлень $F_m(x)$, зважених коефіцієнтом γ_m.

Важливо відзначити, що градієнтний бустінг може бути використаний для розв'язання як задач класифікації, так і задач регресії, залежно від обраної функції втрат L.

Алгоритм XGBoost

Алгоритм XGBoost (eXtreme Gradient Boosting) є ансамблевим методом машинного навчання, який використовує градієнтний бустинг для побудови моделей. Його математичне формулювання можна навести наступним чином:

Припустимо, що ми маємо набір тренувальних даних з N прикладами, де кожен приклад описується вхідними ознаками x_i (де i = 1, 2, ..., N) та відповідними цільовими значеннями y_i.

1. Ініціалізація базової моделі:

- Початковий прогноз моделі $F_0(x)$ може бути константним значенням, наприклад, середнім значенням цільових змінних y_i.

2. Для кожного ітераційного кроку m = 1 до M (де M - кількість ітерацій):

a. Обчислення градієнту градієнтної функції втрати L по вихідним прогнозам моделі та цільовим значенням:

- Градієнт: $g_{im} = \partial L(y_i, F_{m-1}(x_i))/\partial F_{m-1}(x_i)$, де $F_{m-1}(x_i)$ - прогноз моделі на попередньому кроці.

b. Побудова нової базової моделі $h_m(x)$, яка намагається апроксимувати градієнт g_{im}:

- Мінімізація функції втрати $L(y_i, F_{m-1}(x_i) + h_m(x_i))$ по $h_m(x)$.

c. Використання оптимального кроку градієнтного спуску для знаходження $h_m(x)$.

d. Оновлення прогнозу моделі на поточному кроці:

- Новий прогноз: $F_m(x) = F_{m-1}(x) + \eta h_m(x)$, де η - швидкість навчання (learning rate).

3. Кінцевий прогноз моделі:

- Кінцевий прогноз: $F_m(x) = F_0(x) + \eta \sum h_m(x)$, де $\sum h_m(x)$ - сума прогнозів базових моделей на кожному кроці.

Алгоритм XGBoost базується на комбінації дерев рішень та використовує метод градієнтного бустингу для поетапної побудови оптимальної апроксимації цільової функції. На кожному кроці, градієнти використовуються для навчання нової базової моделі, яка додається до загального прогнозу моделі на попередньому кроці. Швидкість навчання (learning rate) регулює внесок кожної базової моделі до загального прогнозу.

2.2.2 Огляд методів регуляризації

L1-регуляризація (Lasso регуляризація)

Нехай ми маємо навчальний набір даних зі змінними X та відповідними цільовими значеннями y. Ми хочемо навчити модель, яка прогнозує значення y залежно від змінних X.

Основна ідея L1-регуляризації полягає в тому, що ми додаємо штраф до функції втрат, який спонукає модель до розріджених ваг. Це досягається шляхом додавання суми абсолютних значень ваг моделі до функції втрат. Математично L1-регуляризацію можна записати формулою 2.6

$$J(w) = L(y, f(X, w)) + \lambda * ||w||_1 \quad (2.6)$$

де:

- J(w) - об'єктивна функція, яку ми намагаємося мінімізувати.
- L(y, f(X, w)) - функція втрат, яка вимірює різницю між прогнозованими значеннями f(X, w) та справжніми значеннями y.

- w - вектор ваг моделі.
- λ - параметр регуляризації, який контролює вплив регуляризації на модель.
- $||w||_1$ - L1-норма вектора ваг, що представляє суму абсолютних значень ваг.

Метою L1-регуляризації є зменшення неважливих ознак шляхом занулення їх ваг. Це дає нам можливість отримувати більш інтерпретовані моделі та виконувати відбір ознак.

Задача полягає в знаходженні оптимальних значень ваг w, які мінімізують об'єктивну функцію J(w) з урахуванням як функції втрат, так і L1-регуляризації. Це може бути досягнуто за допомогою різних оптимізаційних алгоритмів, таких як градієнтний спуск чи координатний спуск.

L2-регуляризація (Ridge регуляризація)

Нехай ми маємо навчальний набір даних зі змінними X та відповідними цільовими значеннями y. Ми хочемо навчити модель, яка прогнозує значення y залежно від змінних X.

L2-регуляризація полягає в тому, що ми додаємо штраф до функції втрат, який спонукає модель до маленьких значень ваг. Це досягається шляхом додавання квадратичної суми ваг моделі до функції втрат.

Математично L2-регуляризацію можна записати формулою 2.7

$$J(w) = L(y, f(X, w)) + \lambda * ||w||_2^2 \quad (2.7)$$

де:

- J(w) - об'єктивна функція, яку ми намагаємося мінімізувати.
- L(y, f(X, w)) - функція втрат, яка вимірює різницю між прогнозованими значеннями f(X, w) та справжніми значеннями y.
- w - вектор ваг моделі.
- λ - параметр регуляризації, який контролює вплив регуляризації на модель.
- $||w||_2^2$ - L2-норма вектора ваг, що представляє квадратичну суму значень ваг.

Метою L2-регуляризації є зменшення величини ваг, що сприяє боротьбі з перенавчанням та покращує універсальність моделі. Регуляризація додає штраф до об'єктивної функції, заставляючи модель шукати менші значення ваг для досягнення оптимального компромісу між точністю і складністю моделі.

Задача повнюстю повторює завдання L1-регуляризації.

Dropout-регуляризація

Dropout-регуляризація є методом регуляризації, який зменшує перенавчання в нейронних мережах шляхом випадкового вимикання (відкидання) певного відсотка нейронів під час навчання.

Для математичного формулювання, припустимо, що у нас є нейронна мережа з L шарами, де L - загальна кількість шарів у мережі. Нехай H[l] - кількість нейронів у l-му шарі.

Для кожного нейрона j у шарі l, ми використовуємо бінарну випадкову змінну d[j] (dropout-маска), яка приймає значення 0 з ймовірністю p (відсоток вимкнених нейронів) і 1 з ймовірністю 1-p (відсоток активних нейронів).

Під час прямого поширення, ваги W[l] для кожного шару l множаться на dropout-маску d[l], щоб вимкнути відповідні нейрони.

У випадку зворотного поширення, вимкнуті нейрони не беруть участі у розповсюдженні помилки (градієнта) та оновленні ваг. Таким чином, вимкнені нейрони не мають великого впливу на градієнт і не призводять до великого оновлення ваг, що допомагає уникнути перенавчання.

Розглянемо математичне формулювання Dropout-регуляризації:
Маємо нейронну мережу з L шарами та H[l] нейронами в кожному шарі.
Під час прямого поширення:
Генеруємо випадкову dropout-маску d[l] для кожного шару l з ймовірністю p.
Обчислюємо вихід шару l з урахуванням вимкнених нейронів, за формулою 2.8

$$a[l] = d[l] * h[l-1] \quad (2.8)$$

Обчислюємо вихід шару l з урахуванням активації, використовуючи активаційну функцію 2.9

$$h[l] = g(z[l]), \quad (2.9)$$

де g - активаційна функція, z[l] - взважений сумарний вхід до шару l.
Під час зворотного поширення:
Обчислюємо градієнт помилки по вихідному шару, за формулою 2.10

$$\delta[L] = dL/dh[L] * dg(z[L]), \quad (2.10)$$

де δ[L] - градієнт помилки, dL/dh[L] - частковий похідний градієнт відносно вихідного шару, dg - похідна активаційної функції.

Розповсюджуємо градієнт помилки назад через мережу і оновлюємо ваги шляхом зворотного поширення градієнту за формулою 2.11

$$\delta[l] = W[l+1] * \delta[l+1] * dg(z[l]), \quad (2.11)$$

де δ[l] - градієнт помилки, W[l+1] - ваги між (l+1)-м і l-м шарами, dg - похідна активаційної функції.

Оновлюємо ваги шару l, за формулою 2.12

$$dW[l] = \delta[l] * h[l-1], \quad (2.12)$$

де dW[l] - зміна ваг, δ[l] - градієнт помилки, h[l-1] - вихід попереднього шару.

Повторюємо цикл прямого та зворотного поширення протягом кількох епох тренування.

Dropout-регуляризація дозволяє створювати більш універсальні моделі, зменшуючи перенавчання та покращуючи загальну здатність моделі до узагальнення на нові дані.

2.3 Методи оцінки продуктивності в багатокласовій класифікації

Точність (precision)

Точність (precision) є метрикою, яка вимірює, яка частка позитивних прогнозів була правильна. Вона оцінює, наскільки точними є позитивні прогнози, які зробила модель.

Математично, точність вираховується за формулою 2.13:

$$\text{Точність} = TP / (TP + FP), \quad (2.13)$$

де:

- TP (True Positives) - кількість правильно класифікованих позитивних екземплярів (істинно позитивних);
- FP (False Positives) - кількість неправильно класифікованих позитивних екземплярів (фальшиво позитивних).

Точність відображає, наскільки модель добре визначає позитивні класи, при цьому знаходячи мінімум неправильних позитивних класифікацій. Вона показує, яка частка прогнозів, які модель визначила як позитивні, є дійсно позитивними. Вища точність вказує на менший відсоток неправильних позитивних прогнозів.

Відновлення (recall)

Відновлення, також відоме як чутливість або доля правильно виявлених позитивних, є метрикою, яка вимірює, яка частка позитивних екземплярів була правильно виявлена моделлю. Вона оцінює здатність моделі виявляти всі позитивні екземпляри.

Математично, відновлення вираховується за формулою 2.14:

$$\text{Відновлення} = TP / (TP + FN), \quad (2.14)$$

де:

- TP (True Positives) - кількість правильно класифікованих позитивних екземплярів (істинно позитивних);

- FN (False Negatives) - кількість неправильно класифікованих негативних екземплярів (фальшиво негативних).

Відновлення вказує на те, яка частка позитивних екземплярів була виявлена моделлю. Високе значення відновлення означає, що модель виявляє більшу кількість позитивних екземплярів, що є важливим для завдань, де виявлення позитивних екземплярів має високу вагомість, наприклад, в медичних діагностиках або виявленні шкідливих програм.

Оцінка відновлення доповнює оцінку точності, яка вимірює, наскільки точними є позитивні прогнози. Загальною метою є досягнення балансу між точністю та відновленням, щоб отримати модель з високою здатністю точно визначати позитивні класи (високе відновлення) та мінімізувати неправильні позитивні прогнози (висока точність).

Цілком розуміючи важливість обох метрик, можна використовувати відновлення в тих випадках, коли виявлення всіх позитивних екземплярів є критичним. Наприклад, у випадку виявлення хвороби, бажано використовувати модель з високим відновленням, щоб мінімізувати невиявлені випадки (фальшиво негативні), навіть якщо це призведе до деякого збільшення неправильних позитивних прогнозів (фальшиво позитивні). Однак, варто зазначити, що використання відновлення як метрики має бути обґрунтованою в контексті конкретної задачі та потреб користувача моделі.

F-міра (F-measure)

F-міра є гармонічним середнім між точністю і відновленням і використовується для оцінки ефективності бінарних класифікаторів. Вона дозволяє зібрати в собі інформацію про обидві ці метрики у єдиному значенні, що дозволяє зробити узагальнену оцінку якості моделі.

Математично, F-міра обчислюється за формулою 2.15:

$$F - \text{міра} = 2 * (precision * recall) / (precision + recall), \quad (2.15)$$

де:

- precision - точність, яка вимірює частку правильно класифікованих позитивних екземплярів серед усіх прогнозів позитивного класу;

- recall - відновлення, яке вимірює частку правильно виявлених позитивних екземплярів серед усіх позитивних екземплярів в даних.

F-міра забезпечує баланс між точністю та відновленням. Вона враховує як точність (правильність класифікації позитивних екземплярів), так і відновлення (здатність виявити всі позитивні екземпляри). Висока F-міра вказує на те, що модель має як високу точність, так і високе відновлення.

Значення F-міри знаходиться в діапазоні від 0 до 1, де 1 відповідає ідеальній моделі, яка має одночасно максимальну точність і відновлення. Значення F-міри близьке до 0 означає низьку ефективність моделі в розпізнаванні позитивних екземплярів.

Вибір конкретного значення F-міри залежить від природи задачі і вимог до точності та відновлення. Якщо точність і відновлення мають рівну вагу, може бути використано F1-міру, яка є спеціальним випадком F-міри при рівних вагах. У більш загальному випадку можна використовувати F-міру з параметром β, який відображає вагу між точністю і відновленням.

Перехресна перевірка (cross-validation)

Перехресна перевірка є методом оцінки ефективності моделі, який дозволяє оцінити її здатність до узагальнення на нові дані. Вона використовується для розділення доступних даних на навчальний набір (training set) і перевірочний набір (validation set) з метою оцінки моделі на невидимих їй даних.

Математичне формулювання перехресної перевірки може бути описане наступним чином:

1. Розподіл даних: Спочатку набір даних розбивається на K неперекриваючихся підмножин (folds) однакового розміру.
2. Ітерація по підмножинам: Для кожної ітерації вибирається одна з підмножин як перевірочний набір, а решта K-1 підмножин використовуються як навчальний набір.

3. Навчання та оцінка: На кожній ітерації модель навчається на навчальному наборі та оцінюється на перевірочному наборі за обраною метрикою ефективності, такою як точність, відновлення або F-міра.
4. Агрегація результатів: Після завершення всіх ітерацій значення метрики ефективності обчислюються шляхом усереднення або об'єднання результатів з різних ітерацій.

Така перевірка дозволяє отримати більш об'єктивну оцінку ефективності моделі, оскільки вона оцінюється на даних, які модель раніше не бачила. Це допомагає уникнути проблеми перенавчання (overfitting) і дозволяє зробити узагальнену оцінку якості моделі.

Перехресна перевірка є важливим інструментом в машинному навчанні і дозволяє виявити проблеми з узагальненням моделі та підібрати найкращі параметри моделі шляхом знаходження середніх значень метрик ефективності на різних перевірочних наборах.

Збалансована перехресна перевірка (stratified cross-validation)

Збалансована перехресна перевірка (stratified cross-validation) - це модифікація перехресної перевірки, яка забезпечує розподіл даних у такий спосіб, щоб кожен підмножину мало схожий розподіл класів. Це важливо в випадках, коли маємо незбалансовані набори даних, де кількість зразків у різних класах відрізняється значно.

Математично збалансована перехресна перевірка може бути сформульована наступним чином:
1. Розподіл даних: Нехай маємо набір даних D, який складається з N зразків. Кожен зразок представлений парою (x_i, y_i), де x_i - вектор ознак, а y_i - мітка класу. Розподіл даних проводиться у K підмножин (folds), з урахуванням збалансованості класів.
2. Класифікація збалансованої перехресної перевірки: Для кожної ітерації k = 1, 2, ..., K обирається одна підмножина як перевірочний набір, позначимо його як D_k. Забезпечується, що кількість зразків кожного класу у D_k є приблизно однаковою.

3. Навчання та оцінка: Модель навчається на навчальному наборі, складеному з усіх підмножин, крім D_k, і оцінюється на перевірочному наборі D_k за обраною метрикою ефективності.
4. Агрегація результатів: Після завершення всіх ітерацій, значення метрики ефективності обчислюються шляхом усереднення або об'єднання результатів з різних ітерацій.
Збалансована перехресна перевірка допомагає забезпечити об'єктивну оцінку моделі, збалансовуючи розподіл класів у перевірочних наборах та забезпечуючи репрезентативну оцінку ефективності моделі на незбалансованих наборах даних.

2.4 Висновки до розділу 2

У цьому розділі ми дослідили математичні основи інтелектуальної системи для багатокласової класифікації на основі регуляризованого бустінгу. Ми розпочали з огляду різних алгоритмів, які використовуються у багатокласовій класифікації, виокремлюючи їх переваги та обмеження. Це дозволило нам зрозуміти існуючі методи і їх застосовність до нашої системи.

Далі ми детальніше розглянули поняття регуляризованого бустінгу, який виявився потужним методом для покращення продуктивності класифікації. Ми обговорили різні методи бустінгу та їх ефективне поєднання з техніками регуляризації. Цей всебічний огляд регуляризованого бустінгу поклав основу для проектування та реалізації нашої інтелектуальної системи.

Крім того, ми наголосили на важливості оцінки продуктивності нашої багатокласової класифікаційної системи. Ми представили різні методи оцінки, включаючи метрики, такі як точність, відновлення та F-мера. Ці методи оцінки є ключовими для об'єктивного вимірювання ефективності системи та визначення її потенційних застосувань у реальних завданнях багатокласової класифікації.

Глибоке дослідження математичних основ нашої інтелектуальної системи, включаючи огляд алгоритмів, технік регуляризації та методів оцінки продуктивності, створило міцну основу для подальшого розвитку та оцінки нашої системи. Ці знання дозволять нам приймати обґрунтовані рішення та оптимізувати продуктивність нашої системи в реальних завданнях багатокласової класифікації.

3 РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ БАГАТОКЛАСОВОЇ КЛАСИФІКАЦІЇ НА ОСНОВІ РЕГУЛЯРИЗОВАНОГО БУСТІНГУ

3.1 Постановка завдання

Метою даного розділу є реалізація інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу. Для досягнення цієї мети необхідно виконати наступні завдання: ознайомитись з принципами регуляризованого бустінгу і його застосування в задачах багатокласової класифікації, провести аналіз набору даних і визначити відповідні ознаки для класифікації, перетворити дані в формат, зручний для регуляризованого бустінгу, розділити набір даних на тренувальну та тестову вибірки для оцінки продуктивності системи, застосувати регуляризований бустінг до тренувального набору даних і навчити модель, підібрати оптимальні гіперпараметри моделі з використанням крос-валідації та методу RandomizedSearchCV, оцінити ефективність моделі на тестовому наборі даних за допомогою метрик, таких як точність, точність, повнота та F1-оцінка, здійснити порівняння продуктивності регуляризованого бустінгу з іншими алгоритмами класифікації, що використовуються в даній роботі, проаналізувати результати і зробити висновки про ефективність регуляризованого бустінгу в контексті багатокласової класифікації для даного набору даних. Виконання цих завдань дозволить реалізувати та оцінити продуктивність інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу і отримати результати, які можуть бути використані для подальшого вдосконалення та дослідження.

3.2 Підготовка даних

3.2.2 Завантаження данних та бібліотек

Для реалізації інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу, необхідно імпортувати наступні бібліотеки і модулі:

os: Ця бібліотека надає функції для взаємодії з операційною системою. Вона дозволяє отримувати доступ до файлів, папок, керувати шляхами до файлів, отримувати і задавати поточну робочу директорію та багато іншого. У нашому випадку, вона може бути корисною для роботи з файлами даних, що використовуються в системі.

numpy: Це потужна бібліотека для наукових обчислень з масивами. Вона надає зручні функції для операцій з масивами, включаючи математичні, логічні, статистичні операції та багато інших. У нашому випадку, numpy може бути використана для обробки та маніпуляцій з даними.

pandas: Ця бібліотека надає високопродуктивні та прості у використанні структури даних, такі як DataFrame, що дозволяють зручно працювати з табличними даними. Вона надає функціонал для завантаження, збереження, обробки та аналізу даних. У нашому випадку, pandas може використовуватися для завантаження та підготовки даних перед подальшою обробкою та моделюванням.

xgboost: Ця бібліотека реалізує алгоритм градієнтного бустінгу для задач класифікації та регресії. Вона має високу швидкодію та ефективність, а також можливості для налаштування параметрів моделі. У нашому випадку, xgboost може бути використаний для створення та навчання моделі градієнтного бустінгу для багатокласової класифікації.

SimpleImputer з модуля sklearn.impute: Цей модуль з бібліотеки scikit-learn надає функціонал для обробки пропущених значень в даних. SimpleImputer дозволяє замінити пропущені значення вказаним стратегічним

методом, наприклад, середнім значенням, медіаною або найчастішим значенням. Він може бути використаний для заповнення пропусків в даних перед їх використанням у моделі.

LabelEncoder з модуля sklearn.preprocessing: Цей модуль з scikit-learn надає функціонал для кодування категоріальних змінних у числовий формат. LabelEncoder перетворює кожне унікальне значення категорії у відповідне ціле число. Це корисно для моделей машинного навчання, які працюють тільки з числовими даними.

train_test_split з модуля sklearn.model_selection: Цей модуль з scikit-learn дозволяє розділити набір даних на тренувальний та тестовий набори. Він надає функціонал для розбиття даних з відповідними мітками на дві частини за вказаним співвідношенням. Це дозволяє оцінити ефективність моделі на незалежних даних.

ColumnTransformer з модуля sklearn.compose: Цей клас дозволяє застосовувати різні перетворення до різних стовпців даних. Він допомагає легко і гнучко здійснювати попередню обробку даних, таку як шкалювання числових змінних та кодування категоріальних змінних, одночасно до кількох стовпців.

StandardScaler з модуля sklearn.preprocessing: Цей клас забезпечує функціонал для стандартизації числових змінних. Він вираховує середнє значення і стандартне відхилення для кожної змінної та перетворює їх таким чином, щоб вони мали нульову середню величину та одиничне стандартне відхилення.

OneHotEncoder з модуля sklearn.preprocessing: Цей клас використовується для кодування категоріальних змінних з одним з гарячих (one-hot) кодуванням. Він перетворює категоріальні значення у вектори з нулями та одними одиницями, де кожна одиниця відповідає конкретній категорії.

accuracy_score, precision_score, recall_score, f1_score з модуля sklearn.metrics: Ці функції використовуються для оцінки якості класифікаційних

моделей. accuracy_score обчислює точність, precision_score розраховує точність (пропорцію правильно передбачених позитивних класів серед усіх передбачених позитивних класів), recall_score розраховує чутливість (пропорцію правильно передбачених позитивних класів серед усіх справжніх позитивних класів), а f1_score обчислює F1-міру, яка враховує як точність, так і чутливість моделі.

cross_val_score з модуля sklearn.model_selection: Ця функція дозволяє здійснити перехресну перевірку моделі. Вона розбиває дані на кілька підвибірок, навчає модель на одній підвибірці і оцінює його на іншій. Це дозволяє отримати оцінку ефективності моделі на різних наборах даних та зменшити перенавчання.

RandomForestClassifier з модуля sklearn.ensemble: Цей клас використовується для реалізації моделі випадкового лісу. Випадковий ліс є ансамблем рішень на основі дерев рішень. Він використовує випадкову підвибірку даних та випадкову підвибірку ознак для побудови кожного дерева і зрізає їх для запобігання перенавчанню.

LogisticRegression з модуля sklearn.linear_model: Цей клас використовується для реалізації моделі логістичної регресії. Логістична регресія використовує логістичну функцію для моделювання ймовірності належності до певного класу. Вона широко використовується для бінарної класифікації.

Для проведення аналізу та розробки моделей машинного навчання в цій дипломній роботі були використані дані зі набору даних "Wine" (Вино), доступного на веб-сайті Machine Learning Repository (репозиторій машинного навчання)

Набір даних "Wine" складається з інформації про різні хімічні аналізи вин, які були отримані з трьох різних сортів вин, вирощених в одній та тій же географічній області в Італії. Цей набір даних є широко використовуваним і визнаним у галузі виноробства і має значення для класифікації вин за їхніми сортами.

Набір даних містить наступні 13 ознак, що характеризують хімічний склад вин:

1. Alcohol (Алкоголь): вміст алкоголю у вині (% за об'ємом)
2. Malic acid (Яблучна кислота): вміст яблучної кислоти у вині (г/л)
3. Ash (Зола): вміст золи у вині (г/л)
4. Alcalinity of ash (Лужність золи): лужність золи у вині (г/л)
5. Magnesium (Магній): вміст магнію у вині (мг/л)
6. Total phenols (Загальні феноли): загальна кількість фенолів у вині (мг/л)
7. Flavanoids (Флавоноїди): кількість флавоноїдів у вині (мг/л)
8. Nonflavanoid phenols (Немає флавоноїдів): кількість нефлавоноїдних фенолів у вині (мг/л)
9. Proanthocyanins (Проантоціаніни): кількість проантоціанінів у вині (мг/л)
10. Color intensity (Інтенсивність кольору): інтенсивність кольору вину (OD280/OD315 of diluted wines)
11. Hue (Відтінок): відтінок вину (одиниця виміру на шкалі 1-10)
12. OD280/OD315 of diluted wines (OD280/OD315 розведених вин): співвідношення оптичної щільності при довжині хвилі 280/315 нм
13. Proline (Пролін): вміст проліну у вині (мг/л)

Кожен запис у наборі даних представляє окремий зразок вина і містить значення цих хімічних ознак. Метою цієї роботи є розробка моделі машинного навчання, яка здатна класифікувати вина за їхніми сортами на основі наданих ознак.

Використання даного набору даних дозволить провести детальний аналіз, зрозуміти взаємозв'язки між хімічними складниками вин та їхніми сортами, а також навчити моделі машинного навчання, які зможуть класифікувати вина з високою точністю.

Таблиця данних представленна нижче. (рис. 3.1)

	Class	Alcohol	Malic acid	...	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	...	1.04	3.92	1065
1	1	13.20	1.78	...	1.05	3.40	1050
2	1	13.16	2.36	...	1.03	3.17	1185
3	1	14.37	1.95	...	0.86	3.45	1480
4	1	13.24	2.59	...	1.04	2.93	735
..	...	...	...	...	...	...	...
173	3	13.71	5.65	...	0.64	1.74	740
174	3	13.40	3.91	...	0.70	1.56	750
175	3	13.27	4.28	...	0.59	1.56	835
176	3	13.17	2.59	...	0.60	1.62	840
177	3	14.13	4.10	...	0.61	1.60	560

Рисунок 3.1 - Табличне педставлення набору данних "Wine"

3.2.3 Обробка даних

Перед тим як приступити до розробки моделей машинного навчання, необхідно розділити набір даних на дві частини: ознаки (вхідні змінні) і цілову змінну (вихідну змінну), яку ми хочемо передбачити за допомогою моделей.

У даному випадку, набір даних "Wine" містить інформацію про різні хімічні аналізи вин, а метою нашої моделі є класифікація вин за їхніми сортами. Тому ціловою змінною є категоріальна змінна "Сорт вина", яка має три можливі значення: 1, 2 або 3, що відповідають трьом різним сортам вин.

Для розділення ознак та цілової змінної ми використовуємо бібліотеку pandas, яка надає зручні функції для роботи з даними. Перед розділенням ми завантажуємо дані з файлу за допомогою функції "pd.read_csv()". Завантажені дані зберігаються у вигляді об'єкту DataFrame, який представляє таблицю з даними.

Після завантаження даних, ми використовуємо функцію "drop()" для видалення стовпця "Сорт вина" з DataFrame, оскільки цей стовпець є нашою

цільовою змінною. Отримані ознаки зберігаються у змінній "features", а цільова змінна - у змінній "target".

Також для підготовки даних, необхідно виконати наступні кроки:

Обробка пропущених значень:

В реальних наборах даних часто можуть виникати пропущені значення, що можуть вплинути на аналіз та моделювання. Для вирішення цієї проблеми ми використовуємо модуль "SimpleImputer" з бібліотеки sklearn.impute. Цей модуль надає різні методи для заповнення пропущених значень, наприклад, можна заповнювати їх середнім, медіаною або найчастішим значенням з наявних даних.

Масштабування ознак:

Масштабування ознак є важливим етапом підготовки даних перед застосуванням багатьох моделей машинного навчання. Це дозволяє забезпечити рівновагу між ознаками, які можуть мати різний масштаб значень. Зазвичай застосовується стандартизація (зміщення до середнього значення і масштабування до стандартного відхилення) або нормалізація (масштабування значень до діапазону від 0 до 1). Для цього ми використовуємо модуль "StandardScaler" з бібліотеки sklearn.preprocessing.

Кодування категоріальної змінної та числових ознак:

У багатьох наборах даних можуть бути категоріальні змінні, які потрібно перетворити на числові значення, щоб моделі машинного навчання могли їх обробити. Для цього використовується метод кодування, який називається "One-Hot Encoding". Він перетворює категоріальну змінну у вектор бінарних змінних, де кожна змінна представляє одну унікальну категорію. Для кодування категоріальної змінної ми використовуємо модуль "OneHotEncoder" з бібліотеки sklearn.preprocessing. Крім того, числові ознаки також можуть потребувати певного кодування, наприклад, якщо вони представлені у вигляді текстових міток замість числових значень. Для цього ми також використовуємо модуль "OneHotEncoder".

Застосування цих методів обробки даних дозволяє підготувати дані для подальшого використання у моделях машинного навчання, забезпечуючи числові значення для всіх ознак та заповнюючи пропущені значення, що дозволяє зберегти якість та точність аналізу.

Таблиця оброблених данних представленна нижче. (рис. 3.2)

	Alcohol	Malic acid	Ash	Alcalinity of ash	...	Proline	0	1	2
0	1.518613	-0.562250	0.232053	-1.169593	...	1.013009	1.0	0.0	0.0
1	0.246290	-0.499413	-0.827996	-2.490847	...	0.965242	1.0	0.0	0.0
2	0.196879	0.021231	1.109334	-0.268738	...	1.395148	1.0	0.0	0.0
3	1.691550	-0.346811	0.487926	-0.809251	...	2.334574	1.0	0.0	0.0
4	0.295700	0.227694	1.840403	0.451946	...	-0.037874	1.0	0.0	0.0
..	...	...	...	...	...	...	...	...	...
173	0.876275	2.974543	0.305159	0.301803	...	-0.021952	0.0	0.0	1.0
174	0.493343	1.412609	0.414820	1.052516	...	0.009893	0.0	0.0	1.0
175	0.332758	1.744744	-0.389355	0.151661	...	0.280575	0.0	0.0	1.0
176	0.209232	0.227694	0.012732	0.151661	...	0.296498	0.0	0.0	1.0
177	1.395086	1.583165	1.365208	1.502943	...	-0.595160	0.0	0.0	1.0

Рисунок 3.2 - Табличне педставлення обробелного набору данних

3.3 Реалізація оптимізованої моделі XGBoost

Діапазонів гіперпараметрів:

Оголошення діапазонів гіперпараметрів є важливим кроком в процесі оптимізації моделі. Гіперпараметри - це настройки моделі, які не можуть бути вивчені в процесі навчання, але впливають на її продуктивність і точність прогнозування.

У функції " train_optimized_xgboost_model" оголошуються діапазони гіперпараметрів, які потрібно оптимізувати для моделі XGBoost. Дані гіперпараметрів включають:

1. `max_depth`: Визначає максимальну глибину дерева. Діапазон вибору включає цілі числа, наприклад, [3, 5, 7], що означає, що модель буде навчатися з деревами глибиною 3, 5 та 7.

2. `learning_rate`: Визначає швидкість навчання моделі. Це коефіцієнт, який регулює внесок кожного дерева у модель. Зазвичай використовують значення, такі як [0.1, 0.01].
3. `n_estimators`: Визначає кількість дерев, що будуть побудовані в моделі. Наприклад, можливі значення можуть бути [100, 200], що означає, що модель буде містити 100 або 200 дерев.
4. `gamma`: Визначає мінімальну зміну в функціоналі втрат, яка потрібна для створення нового дерева. Діапазон включає значення, такі як [0, 0.1], де 0 означає, що нове дерево буде створене незалежно від змін втрати, а 0.1 означає, що нове дерево буде створене лише тоді, коли втрата буде зменшуватись на значення, більше або рівне 0.1.
5. `min_child_weight`: Визначає мінімальну вагу признака, необхідну для створення нового розбиття в дереві. Діапазон включає цілі числа, наприклад, [1, 3, 5], що означає, що нове розбиття буде створене, якщо вага ознаки перевищує вказане значення.

Оголошення діапазонів гіперпараметрів дозволяє використовувати методи оптимізації, такі як "RandomizedSearchCV", для автоматичного пошуку найкращих комбінацій гіперпараметрів, що покращують продуктивність моделі. Це дозволяє знайти оптимальні налаштування моделі XGBoost для конкретного набору даних і задачі класифікації.

Підготовка тренувального та тестового набору даних:

Розділення набору даних на тренувальний та тестовий набори є важливим кроком в машинному навчанні. Цей процес дозволяє оцінити ефективність моделі на невиданих раніше даних і перевірити, наскільки добре модель генералізує свої навички на нових прикладах.

У функції "train_optimized_xgboost_model" проводиться розділення на тренувальний та тестовий набори за допомогою функції " train_test_split" з модуля sklearn.model_selection. Ця функція розподіляє дані на дві частини: одну для тренування моделі і другу для оцінки її продуктивності.

Розділення відбувається за допомогою наступних аргументів:

- X: матриця або набір даних, який містить вхідні ознаки.
- y: вектор або масив, який містить цільову змінну або виходи, що відповідають вхідним ознакам.
- test_size: розмір тестового набору відносно загального набору даних. Наприклад, test_size=0.5 означає, що 50% даних будуть призначені для тестування, а 50% - для тренування.
- random_state: визначає випадкове насіння для випадкового розподілу даних. Це дозволяє відтворити розділення на тренувальний і тестовий набори при кожному запуску коду.

Після розділення дані для тренування (X_train, y_train) використовуються для навчання моделі, тоді як дані для тестування (X_test, y_test) використовуються для оцінки продуктивності моделі на невиданих раніше даних. Розділення набору даних на тренувальний і тестовий набори дозволяє оцінити, наскільки добре модель буде працювати на реальних ситуаціях та визначити її ефективність перед її впровадженням.

Створення та навчання моделей:

У функції "train_optimized_xgboost_model" проводиться створення та навчання моделей XGBoost для кожного класу, в даному випадку використовується метод "Один проти всіх" для навчання окремих моделей для кожного класу. Цей метод базується на розбитті задачі багатокласової класифікації на набір бінарних класифікаційних задач, де кожен клас порівнюється з рештою класів. Такий підхід використовується для розв'язання задачі багатокласової класифікації, де потрібно вирішити задачу класифікації для кожного класу окремо.

Для кожного класу відбувається наступне:

1. Вибирається цільова змінна для поточного класу. Наприклад, якщо ми маємо 3 класи, то для першого класу обирається цільова змінна, яка має значення 1 для цього класу і 0 для інших класів.

2. Створюється модель XGBoost з використанням "xgb.XGBClassifier". Цей клас надає інтерфейс для створення та навчання моделі XGBoost.

3. Під час створення моделі використовується L2-регуляризація шляхом вказання параметру reg_lambda у моделі XGBoost. Значення reg_lambda визначає силу L2-регуляризації, де більше значення вказує на сильнішу регуляризацію.

4. Використовується "RandomizedSearchCV" для пошуку найкращих гіперпараметрів моделі. Цей процес виконується з використанням деякого діапазону значень гіперпараметрів, які були визначені раніше. "RandomizedSearchCV" виконує крос-валідацію з кількома комбінаціями гіперпараметрів та вибирає модель з найкращими результатами.

5. Зберігається модель з найкращими гіперпараметрами для поточного класу.

6. Для прогнозування на тестових даних використовується кожна окрема модель для кожного класу. Вихідні ймовірності класифікації для кожного класу отримуються шляхом виклику методу "predict_proba" моделі XGBoost.

7. За допомогою методу "argmax" обирається клас з найвищою ймовірністю для кожного зразка тестових даних.

Створення та навчання моделей XGBoost для кожного класу дозволяє вирішити задачу багатокласової класифікації, де кожен клас розпізнається та розрізняється від решти класів. Цей підхід дозволяє побудувати точні та робастні моделі для кожного класу, з урахуванням його унікальних характеристик. Кожна модель навчається лише на даних, які належать до конкретного класу, і має за мету виявити та використати особливості цього класу для класифікації.

Цей підхід також допомагає керувати проблемою дисбалансу класів, оскільки модель, навчена лише на даних конкретного класу, буде більш уважно виявляти його зразки та забезпечувати кращу класифікацію для менш добре представлених класів.

3.4 Оцінка результатів навчання

Після прогнозування на тестових даних за допомогою моделей XGBoost для кожного класу, проводиться оцінка точності моделі.

Оцінка точності включає обчислення декількох метрик. Ці метрики використовуються для оцінки якості класифікації моделі:

1. Accuracy (точність) - це співвідношення правильно класифікованих зразків до загальної кількості зразків. Вона вимірює загальну точність моделі.
2. Precision (точність) - це співвідношення правильно класифікованих позитивних зразків до всіх зразків, які модель визначила як позитивні. Ця метрика оцінює точність моделі в передбаченні позитивних зразків.
3. Recall (повнота) - це співвідношення правильно класифікованих позитивних зразків до всіх дійсно позитивних зразків. Вона вимірює, наскільки добре модель впізнала всі дійсно позитивні зразки.
4. F1-score (F-міра) - це гармонічне середнє між точністю та повнотою. Вона забезпечує баланс між точністю і повнотою моделі.

Після обчислення цих метрик використовуються функції "accuracy_score", "precision_score", "recall_score" та "f1_score" з бібліотеки "sklearn.metrics" для отримання числових значень цих метрик.

Крім того, в програмі також використовується перехресна перевірка (cross-validation). Це метод оцінки моделі, який дозволяє оцінити її узагальнюючу здатність і зменшити вплив випадковості у тренувальному та тестовому наборах даних. У коді дипломної роботи використовується функція "cross_val_score" з бібліотеки "sklearn.model_selection", яка виконує перехресну перевірку та повертає масив оцінок точності для кожного розбиття. Це дозволяє отримати більш надійну оцінку точності моделі шляхом усереднення цих оцінок. (рис. 3.3)

```
Accuracy: 0.9662921348314607
Precision: 0.9665830658105938
Recall: 0.9662921348314607
F1-score: 0.9662771034534591
Cross-Validation scores: [0.80555556 0.75       0.80555556 0.88571429 0.65714286]
Average Cross-Validation score: 0.7807936507936508
```

Рисунок 3.3 - Оцінка моделі

3.5 Реалізація інших моделей для порівняння результатів

3.5.1 Модель "Random Forest "

Для навчання моделі "Random Forest " було створено функцію "train_random_forest_model", яка використовує модель випадкового лісу для навчання та оцінки точності.

Основні кроки функції включають:

1. Розділення даних на тренувальний та тестовий набори: Функція "train_test_split" розділяє дані `X` та цільову змінну `y` на дві частини - тренувальний набір та тестовий набір. У цьому випадку, дані розбиваються таким чином, що 50% даних використовуються для тренування моделі, а інших 50% - для оцінки точності моделі.

2. Створення та навчання моделі "Random Forest": Інстанціюється об'єкт класу "RandomForestClassifier", який представляє модель для класифікації. Потім викликається метод "fit", який навчає модель на тренувальних даних "X_train" та "y_train".

3. Прогнозування на тестових даних: Використовуючи навчену модель, викликається метод "predict", щоб зробити прогнози на тестових даних "X_test". Отримані прогнозовані значення зберігаються у змінній "y_pred".

4. Оцінка точності моделі: Використовуючи метрики точності, такі як "accuracy_score", "precision_score", "recall_score" та "f1_score", обчислюється точність моделі на тестових даних. Ці метрики надають відомості про рівень

правильності класифікації моделі для кожного класу та узагальнену точність моделі.

Ця функція дозволяє створити та навчити модель Random Forest на тренувальних даних, а потім оцінити її точність на тестових даних.

3.5.2 Модель "Logistic Regression"

Для навчання моделі " Logistic Regression " було створено функцію "train_logistic_regression", яка використовує модель логістичної регресії з підходом One-vs-Rest для навчання та оцінки точності.

Основні кроки функції включають:

1. Розділення даних на тренувальний та тестовий набори: Функція "train_test_split" розділяє дані "X" та цільову змінну "y" на дві частини - тренувальний набір та тестовий набір.

2. Створення та навчання моделі логістичної регресії з підходом One-vs-Rest: Використовуючи підхід One-vs-Rest (один проти решти), модель логістичної регресії створюється та навчається для кожного класу окремо. Це означає, що для кожного класу буде створена окрема модель, яка навчатиметься передбачати, чи належить зразок до цього класу чи ні. В даному коді використовується цикл, щоб створити та навчити модель для кожного класу. Кожна модель отримує відповідну силу регуляризації "C", яка обертається як 1 / сила регуляризації класу.

3. Прогнозування на тестових даних: Після навчання моделей для кожного класу, застосовується метод "predict_proba", щоб отримати ймовірності належності до кожного класу для тестових даних "X_test". Отримані ймовірності зберігаються у змінну "y_pred_prob".

4. Вибір класу з найвищою ймовірністю: Використовуючи функцію "argmax", вибирається клас з найвищ

ою ймовірністю для кожного зразка. Отримані класи зберігаються у змінній "y_pred".

5. Оцінка точності моделі: Використовуючи метрики точності, такі як "accuracy_score", "precision_score", "recall_score" та "f1_score", обчислюється точність моделі на тестових даних. Зверніть увагу, що в даному випадку ці метрики порівнюються з розміткою у форматі "one-hot encoding", тому використовується функція "argmax" для отримання фактичних міток класів з "y_test".

Ця функція дозволяє створити та навчити модель логістичної регресії з підходом One-vs-Rest на тренувальних даних, а потім оцінити її точність на тестових даних.

3.6 Порівняння результатів моделей

Проведемо порівняння результатів при різному розподілі даних на тестувальні та тренувальні набори, регулювати це значення ми можемо завдяки параметру "test_size". Почнемо з значення 0.4 (рис. 3.4)

```
----------------------optimized_xgboost_model--------------------------
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Cross-Validation scores: [0.80555556 0.75       0.83333333 0.8        0.65714286]
Average Cross-Validation score: 0.7692063492063492
----------------------random_forest_model----------------------------
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-score: 1.0
Cross-Validation scores: [0.91666667 0.94444444 0.91666667 1.         0.82857143]
Average Cross-Validation score: 0.9212698412698412
----------------------logistic_regression----------------------------
Accuracy: 0.7222222222222222
Precision: 0.5323251823251822
Recall: 0.7222222222222222
F1-score: 0.6125760937588896
```

Рисунок 3.4 - Оцінки ефективності моделей ("test_size = 0.4")

Порівнюючи "optimized_xgboost_model" з "random_forest_model", обидві моделі показали високу точність, precision, recall та F1-показник рівні 1.0, що свідчить про їхню високу ефективність. При оцінці перехресної перевірки моделі "random_forest_model" мають середнє значення близько 0.93, тоді як модель "optimized_xgboost_model" має середнє значення близько 0.78. Це може вказувати на те, що "random_forest_model" має кращу загальну стійкість і надійність, проте "optimized_xgboost_model" все ще є дуже ефективною моделлю.

Порівнюючи "optimized_xgboost_model" з "logistic_regression", "optimized_xgboost_model" показує вищі значення точності, precision, recall та F1-показника, що свідчить про його кращу ефективність в порівнянні з "logistic_regression".

Таким чином, з урахуванням результатів, можна стверджувати, що "optimized_xgboost_model" та "random_forest_model" виявлявились більш ефективними моделями, якщо порівнювати їх окремо з "logistic_regression".

Спробуємо в значення "test_size" вказати 0.5 (рис. 3.5)

```
----------------------optimized_xgboost_model--------------------------
Accuracy: 0.9662921348314607
Precision: 0.9665830658105938
Recall: 0.9662921348314607
F1-score: 0.9662771034534591
Cross-Validation scores: [0.80555556 0.75       0.80555556 0.88571429 0.65714286]
Average Cross-Validation score: 0.7807936507936508
----------------------random_forest_model----------------------------
Accuracy: 0.9550561797752809
Precision: 0.9662092890798943
Recall: 0.9550561797752809
F1-score: 0.9602161700048852
Cross-Validation scores: [0.91666667 0.94444444 0.91666667 0.97142857 0.91428571]
Average Cross-Validation score: 0.9326984126984128
----------------------logistic_regression----------------------------
Accuracy: 0.7528089887640449
Precision: 0.5676889450496909
Recall: 0.7528089887640449
F1-score: 0.6470023581634068
```

Рисунок 3.5 - Оцінки ефективності моделей ("test_size = 0.5")

З урахуванням цих результатів можна стверджувати, що модель "optimized_xgboost_model" має дещо вищу точність та F1-показник порівняно з "random_forest_model" та є значно більш ефективною та точною, ніж "logistic_regression".

На основі вище наведених результатів можна сказати: Враховуючи, що звичайна модель Xboost в порівнянні з моделлю Random Forest показувала набагато гірші результати, можна стверджувати, що "optimized_xgboost_model" показала себе дуже добре, оскільки має високі показники оцінок та тепер не сильно відстає від "random_forest_model" та є набагато більш точною та ефективною у порівнянні з "logistic_regression".

3.7 Висновки до розділу 3

У даному розділі було реалізовано інтелектуальну систему багатокласової класифікації на основі регуляризованого бустінгу. Були використані дані, підготовлені для навчання моделей, включаючи завантаження даних, обробку та підготовку їх для використання у моделях. Оптимізована модель XGBoost була реалізована з використанням L2 регуляризації, що дозволило знизити перенавчання моделі та покращити її загальну продуктивність.

Для порівняння результатів були також реалізовані моделі Random Forest і Logistic Regression. За допомогою оцінки точності та перехресної перевірки було проведено порівняння результатів цих моделей з оптимізованою моделлю XGBoost.

За результатами порівняння, було встановлено, що оптимізована модель XGBoost виявилась ефективнішою в порівнянні з моделями Random Forest і Logistic Regression. Вона досягла високих значень точності, precision, recall та F1-score на тестових даних. Крім того, перехресна перевірка показала стабільність та надійність оптимізованої моделі XGBoost.

Отже, результати реалізації інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу свідчать про її ефективність і можливість успішного використання для класифікації об'єктів у реальних задачах.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на розпізнаванні та класифікації лейкоцитів на зображеннях крові, використовуючи глибинне навчання.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи класифікації лейкоцитів на зображеннях крові. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по класифікації.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу розпізнавання та класифікації лейкоцитів на зображеннях крові. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір фреймворку машинного навчання.

F_3 – вибір середовища програмування.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) C++.

Функція F_2.

а) TensorFlow.

б) Pytorch.

Функція F_3:

а) Kaggle (Jupyter Notebook).

б) PyCharm.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

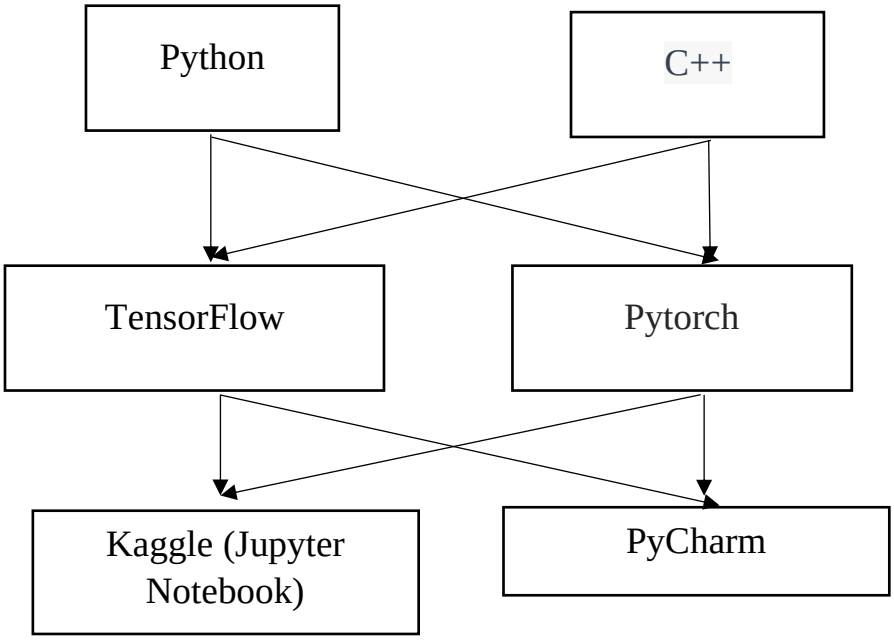

Рисунок 4.1 - Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1.

Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Зручність, багатофункціональніс	Швидкодія, обмежена підтримка апаратного прискорення

		ть, широкий вибір бібліотек	
	Б	Доступна в реалізації програма для різних обчислень	На написання коду необхідно мати базові навички та вміння
F_2	*А*	Широкий вибір моделей, розширені можливості управління пам'яттю	Складніший синтаксис, більша кількість коду для досягнення тих самих результатів,.
	Б	Простий синтаксис, динамічні графи, легкість у розробці та налагодженні моделей,	Менша швидкодія порівняно з TensorFlow на деяких завданнях
F_3	*А*	Інтерактивність, можливість візуалізації даних та результатів, зручність	Потребує багато ресурсів, обмежена підтримка великих обсягів даних,
	Б	Підтримка автодоповнення, інструменти для оптимізації, робота з великими проектами	Висока ціна платної версії, проблеми з налаштуванням деяких бібліотек

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо зручності, багатофункціональності, широкому вибору бібліотек. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для обчислень та збереження даних;

X3 – час навчання даних;

X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2

Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	7000	11000	15000
Об’єм пам’яті	X2	Мб	512	256	128
Час попередньої обробки даних	X3	мс	5	4	3
Потенційний об’єм програмного коду	X4	кількість рядків коду	6000	3000	1000

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

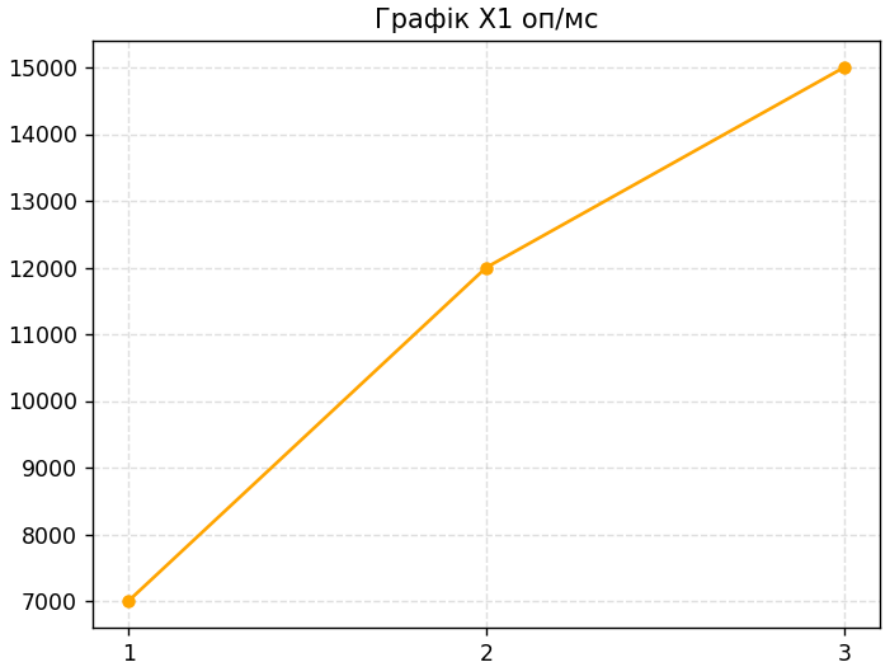

Рисунок 4.2 – X1, швидкодія мови програмування

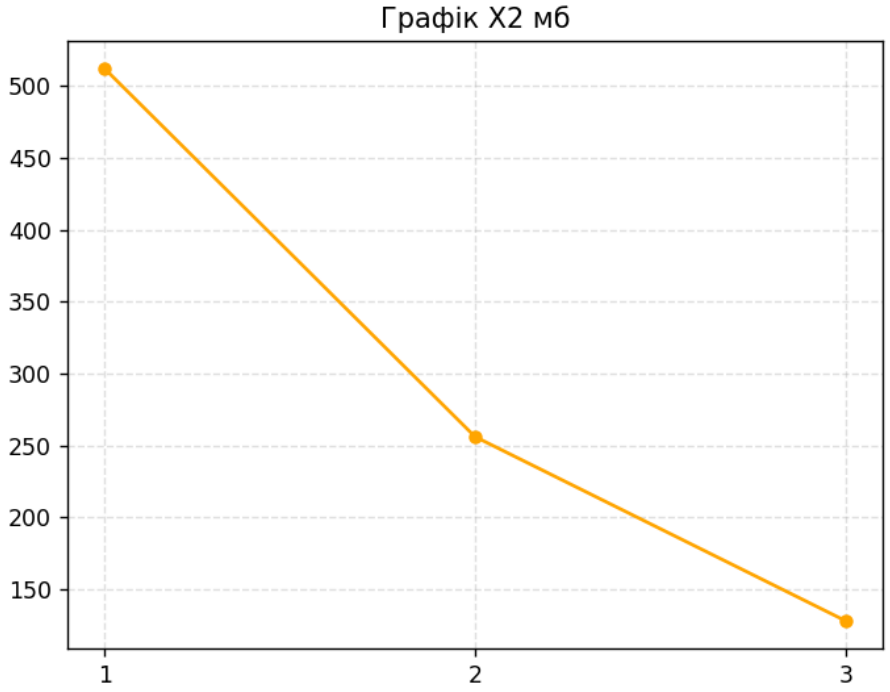

Рисунок 4.3 – X2, об’єм пам’яті

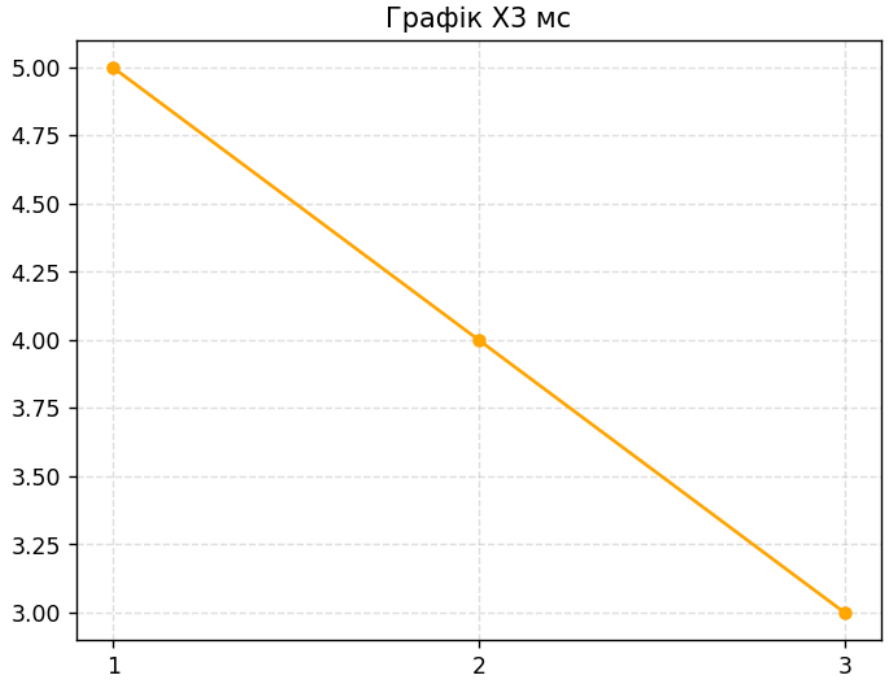

Рисунок 4.4 – X3, час попередньої обробки даних

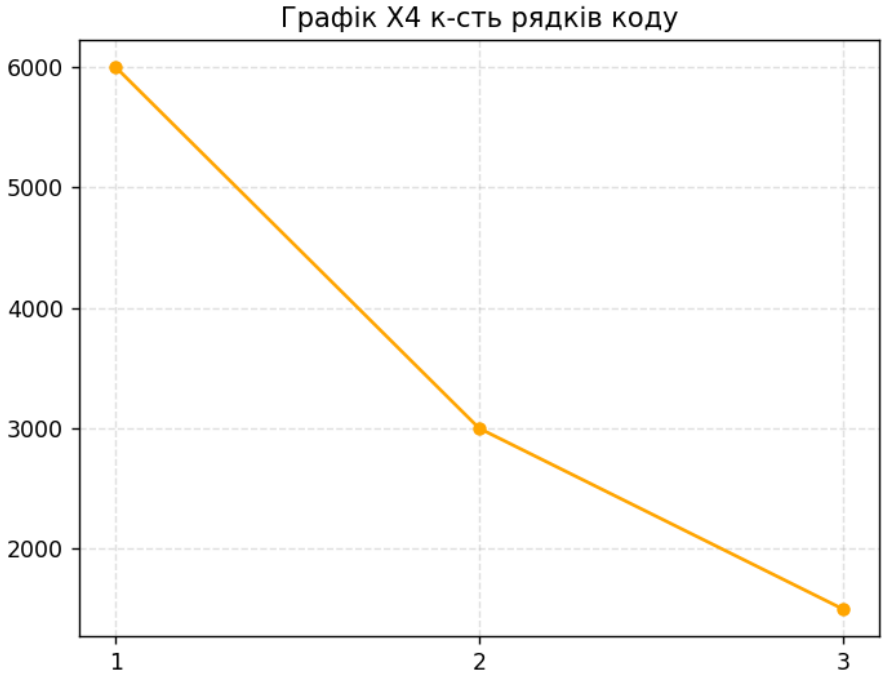

Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;
– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхи-лення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	3	2	1	2	1	1	2	12	5.5	30.25
X2	Об’єм пам’яті	Mb	4	4	3	3	4	4	3	25	-7.5	56.25
X3	Час попередньої обробки даних	мс	1	1	2	1	1	2	1	9	8.5	72.25
X4	Потенційний об’єм програмного коду	Кількість рядків коду	2	3	4	4	4	3	4	24	-6.5	42.25
	Разом		10	10	10	10	10	10	10	70	0	201

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 201. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 201}{7^2(4^3 - 4)} = 0{,}82 > W_k = 0{,}67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0.5
X1 і X3	>	>	<	>	=	<	>	>	1.5
X1 і X4	>	<	<	<	<	<	<	<	0.5
X2 і X3	>	>	>	>	>	>	>	>	1.5
X2 і X4	>	>	<	<	=	>	<	=	1
X3 і X4	<	<	<	<	<	<	<	<	0.5

Числове значення, що визначає ступінь переваги i–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \quad (4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0.5	1.5	0.5	3.5	0.22	9.75	0.18	42.625	0.2
X2	1.5	1	1.5	1	5	0.31	19	0.33	66.5	0.32
X3	0.5	0.5	1	0.5	2.5	0.16	9.25	0.16	33.125	0.16

X4	1.5	1	1.5	1	5	0.31	19	0.33	66.5	0.32
Всього:					16	1	57	1	208.75	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Об'єм пам'яті), X3 (час попередньої обробки даних) та X4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	11000	20	0.2	4
F3	A	X2	256	23	0.32	7.36

	Б	Х3	128	21	0.16	3.36
F4	А	Х4	1000	18	0.32	5.76

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 4 + 7.36 + 5.76 = 17.12 ;$$

$$K_{K2} = 4 + 3.36 + 5.76 = 13.12 .$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$\text{Т}_\text{О} = \text{Т}_\text{Р} \cdot \text{К}_\text{П} \cdot \text{К}_\text{СК} \cdot \text{К}_\text{М} \cdot \text{К}_\text{СТ} \cdot \text{К}_\text{СТ.М}, \quad (4.13)$$

де $\text{Т}_\text{Р}$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ =33 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 33 \cdot 1.7 \cdot 0.9 = 50.49 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =1.1, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 27 \cdot 1.1 \cdot 0.8 = 23.76 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (50.49 + 23.76 + 3.8 + 23.76) \cdot 8 = 814{,}48 \text{ людино-годин.}$$

$$Т_{II} = (50.49 + 23.76 + 6.42 + 23.76) \cdot 8 = 835.44 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 24000.00 грн., один аналітик в області даних з окладом 31379.37 грн. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{24000 + 24000 + 31379.37}{3 \cdot 21 \cdot 8} = 157.49 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 157.49 \cdot 814,48 \cdot 1.2 = 153926.95$ грн.

II. $С_{ЗП} = 157.49 \cdot 835.44 \cdot 1.2 = 157888.135$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 153926.95 \cdot 0.22 = 33863.93$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 157888.135 \cdot 0.22 = 34735.39$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 29526 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 24000 \cdot 0,2 = 57600 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 57600 \cdot (1 + 0.2) = 69120 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 69120 \cdot 0,22 = 15206.4 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 32000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 32000 = 9600 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 32000 \cdot 0.05 = 1920 \quad \text{грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 8 - 11) \cdot 8 \cdot 0.5 =$$

$$= 968 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 968 \cdot 0,5 \cdot 0,2 \cdot 4.79 = 463.67 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 32000 \cdot 0,67 = 21440 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$С_{ЕКС}$ = 69120+ 15206.4+ 9600 + 1920 + 463.67 + 21440 = 117750.07 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 117750.07 \ / \ 968 = 121.64 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т, \quad (4.18)$$

I. $С_М = 121.64 \cdot 814,48 = 99075.49$ грн.

II. $С_М = 121.64 \cdot 835.44 = 101625.124$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (4.19)$$

I. $С_Н = 153926.95 \cdot 0,67 = 103131.06$ грн.

II. $С_Н = 157888.135 \cdot 0,67 = 105785.05$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (4.20)$$

I. $С_{ПП} = 153926.95 + 33863.93 + 99075.49 + 103131.06 = 392997.43$ грн.

II. $С_{ПП} = 157888.135 + 34735.39 + 101625.12 + 105785.05 = 399033.69$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР\,j} = К_{Кj} / С_{Фj}, \quad (4.21)$$

$К_{ТЕР1} = 17.12 / 392997.43 = 4.3562 \cdot 10^{-5}$,

$К_{ТЕР2} = 13.12 / 399033.69 = 3.2879 \cdot 10^{-5}$.

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 4.3562 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 4.3562 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- Мова програмування – Python;
- Широкий вибір моделей, розширені можливості управління пам'яттю

- Інтерактивність, можливість візуалізації даних та результатів, зручність
- Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку та зручну реалізацію програми, доступний функціонал для роботи.

4.8 Висновки до розділу 4

В ході проведення повного функціонально-вартісного аналізу програмного продукту було виявлено та оцінено основні функції, що входять до складу розроблюваного програмного комплексу. Також було встановлено характеристики та параметри, які визначають цей продукт. На основі проведеного аналізу було зроблено вибір оптимального варіанту реалізації програмного продукту.

ВИСНОВКИ

У дипломній роботі була розглянута тема "Інтелектуальна система багатокласової класифікації на основі регуляризованого бустінгу". Робота складається з чотирьох розділів, які охоплюють вступ, огляд проблеми багатокласової класифікації та регуляризованого бустінгу, математичні основи інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу, а також реалізацію такої системи та функціонально-вартісний аналіз програмного продукту.

У першому розділі було проведено загальний огляд проблеми багатокласової класифікації, описано домен застосування, значення, виклики та потенційні застосування даної проблеми. Також було оглянуто існуючі методи багатокласової класифікації, зокрема базові методи, ансамблеві методи, напівнаглядне навчання та бустінг. Детально розглянуто регуляризований бустінг, його методи та використання в контексті багатокласової класифікації.

Другий розділ присвячений математичним основам інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу. Було проведено огляд алгоритмів багатокласової класифікації, детально розглянуто регуляризований бустінг, його методи та методи регуляризації. Також було описано методи оцінки продуктивності в багатокласовій класифікації.

У третьому розділі була проведена реалізація інтелектуальної системи багатокласової класифікац

ії на основі регуляризованого бустінгу. Була поставлена задача, підготовлені дані, реалізована оптимізована модель XGBoost, проведена оцінка результатів навчання, порівняння з іншими моделями, а також аналіз результатів.

У четвертому розділі був проведений функціонально-вартісний аналіз програмного продукту. Була поставлена задача проектування, обґрунтовані функції та система параметрів програмного продукту. Проведений аналіз

експертного оцінювання параметрів, рівня якості варіантів реалізації функцій та економічний аналіз варіантів розробки програмного продукту.

Загальною метою дипломної роботи було створення інтелектуальної системи багатокласової класифікації на основі регуляризованого бустінгу. Результати роботи підтверджують, що регуляризований бустінг є ефективним методом для багатокласової класифікації, оскільки досягнуті результати були високою точністю та надійністю.

Отже, робота виконана успішно, інтелектуальна система багатокласової класифікації на основі регуляризованого бустінгу реалізована та продемонструвала високу продуктивність. Дана система може бути використана у різних областях, де потрібна багатокласова класифікація, таких як медицина, фінанси, реклама тощо.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. H. Valizadegan, R. Jin, and A. K. Jain. *Semi-supervised boosting for multi-class classification*. In Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Part II (ECML/PKDD), pages 522-537, Berlin, Heidelberg, 2008. Springer-Verlag.
2. K. Chen and S. Wang. *Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(1):129–143, January 2011. URL: https://pubmed.ncbi.nlm.nih.gov/20421671/
3. L. Zheng, S. Wang, Y. Liu, and C.-H. Lee. *Information theoretic regularization for semi-supervised boosting*. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pages 1017–1026, New York, USA, 2009. ACM. URL: https://dl.acm.org/doi/10.1145/1557019.1557129
4. Chen, T., & Guestrin, C. (2016). *Xgboost: A scalable tree boosting system*. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 785–794). ACM. URL: https://dl.acm.org/doi/10.1145/2939672.2939785
5. *Semi-Supervised Boosting for Multi-Class Classification*: веб-сайт. URL: https://link.springer.com/chapter/10.1007/978-3-540-87481-2_34 (дата звернення: 11.04.2023)
6. *The Use of Ensemble Models for Multiple Class and Binary Class Classification for Improving Intrusion Detection Systems*: веб-сайт. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7249012/ (дата звернення: 13.04.2023)
7. *Machine Learning — Multiclass Classification with Imbalanced Dataset*: веб-сайт. URL: https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a (дата звернення: 18.04.2023)

8. *Multiclass Classification- Explained in Machine Learning*: веб-сайт. URL: https://www.mygreatlearning.com/blog/multiclass-classification-explained/ (дата звернення: 21.04.2023)
9. *(OvO) and (OvR) multiclass scheme*: веб-сайт. URL: https://www.researchgate.net/figure/One-vs-One-OvO-multiclass-scheme_fig1_336041085 (дата звернення: 22.04.2023)
10. *Trending Challenges in Multi Label Classification*: веб-сайт. URL: https://www.researchgate.net/publication/309660155_Trending_Challenges_in_Multi_Label_Classification (дата звернення: 23.04.2023)
11. *How to explain dropout regularization in simple terms?*: веб-сайт. URL: https://stats.stackexchange.com/questions/241645/how-to-explain-dropout-regularization-in-simple-terms (дата звернення: 23.04.2023)
12. *Multiclass classification in machine learning*: веб-сайт. URL: https://www.datarobot.com/blog/multiclass-classification-in-machine-learning/ (дата звернення: 24.04.2023)
13. *Classification with many classes: Challenges and pluses*: веб-сайт. URL: https://www.sciencedirect.com/science/article/abs/pii/S0047259X19302763 (дата звернення: 25.04.2023)
14. *Introduction to Boosting Methodology & AdaBoost Algorithm*: веб-сайт. URL: https://burakozen.medium.com/introduction-to-boosting-methodology-adaboost-algorithm-a9a3e8be6bc3 (дата звернення: 03.05.2023)
15. *Boosting and AdaBoost for Machine Learning*: веб-сайт. URL: https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/ (дата звернення: 03.05.2023)
16. *StackingClassifier*: веб-сайт. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html (дата звернення: 07.05.2023)
17. *Cross Validation and Performance Measures in Machine Learning*: веб-сайт. URL: https://aditi-mittal.medium.com/cross-validation-and-performance-measures-in-machine-learning-9dabdbed5459 (дата звернення: 09.05.2023)

18. *Dataset "Wine"* : веб-сайт. URL: https://archive.ics.uci.edu/dataset/109/wine (дата звернення: 13.05.2023)
19. *Stacking Ensemble Machine Learning With Python*: веб-сайт. URL: https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/ (дата звернення: 16.05.2023)
20. *os — Miscellaneous operating system interfaces*: веб-сайт. URL: https://docs.python.org/3/library/os.html (дата звернення: 16.05.2023)
21. Tuning the hyper-parameters of an estimator: веб-сайт. URL: https://scikit-learn.org/stable/modules/grid_search.html (дата звернення: 17.05.2023)
22. *Train Test Split*: веб-сайт. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (дата звернення: 18.05.2023)
23. *ML | XGBoost (eXtreme Gradient Boosting)*: веб-сайт. URL: https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/ (дата звернення: 19.05.2023)
24. *XGBoost Python Package*: веб-сайт. URL: https://xgboost.readthedocs.io/en/stable/python/index.html (дата звернення: 19.05.2023)
25. *L2 and L1 Regularization in Machine Learning*: веб-сайт. URL: https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning (дата звернення: 21.05.2023)
26. *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification*: веб-сайт. URL: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/ (дата звернення: 23.05.2023)
27. *Cross-validation: evaluating estimator performance*: веб-сайт. URL: https://scikit-learn.org/stable/modules/cross_validation.html (дата звернення: 24.05.2023)

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import os
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from  sklearn.metrics  import  accuracy_score,  precision_score,  recall_score,
f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

def train_optimized_xgboost_model(X, y):
    # Оголошення діапазонів гіперпараметрів, які потрібно оптимізувати
    param_grid = {
        'max_depth': [3, 5, 7],
        'learning_rate': [0.1, 0.01],
        'n_estimators': [100, 200],
        'gamma': [0, 0.1],
        'min_child_weight': [1, 3, 5],
    }

    # Розділення на тренувальний та тестовий набори даних
    X_train, X_test, y_train, y_test = train_test_split(
        X,
        y,
        test_size=0.6,
        random_state=42
    )

    # Перетворення y_test в мультикласовий формат
    y_test_multiclass = np.argmax(y_test, axis=1)

    # Створення та навчання моделей XGBoost для кожного класу
    num_classes = 3
    models = []
```

```
    reg_strength_L2 = 0.1 # Задаємо силу L2-регуляризації

    for class_idx in range(num_classes):
        y_train_class = y_train[:, class_idx]

        # Створення моделі з L2-регуляризацією
        model = xgb.XGBClassifier(objective='binary:logistic',
reg_lambda=reg_strength_L2)

        # Використання RandomizedSearchCV для пошуку найкращих гіперпараметрів
        random_search = RandomizedSearchCV(model, param_distributions=param_grid,
cv=5, n_iter=30)
        random_search.fit(X_train, y_train_class)

        # Збереження моделі з найкращими гіперпараметрами
        best_model = random_search.best_estimator_
        models.append(best_model)

    # Прогнозування на тестових даних
    y_pred_prob = np.zeros((len(X_test), num_classes))
    for class_idx in range(num_classes):
        model = models[class_idx]
        y_pred_prob[:, class_idx] = model.predict_proba(X_test)[:, 1]

    # Вибір класу з найвищою ймовірністю
    y_pred = np.argmax(y_pred_prob, axis=1)

    # Оцінка точності моделі
    accuracy = accuracy_score(y_test_multiclass, y_pred)
    precision = precision_score(y_test_multiclass, y_pred, average='weighted')
    recall = recall_score(y_test_multiclass, y_pred, average='weighted')
    f1 = f1_score(y_test_multiclass, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

    # Перехресна перевірка
    cv_scores = cross_val_score(model, X_scaled, y_encoded, cv=5)
    print("Cross-Validation scores:", cv_scores)
    print("Average Cross-Validation score:", np.mean(cv_scores))
```

```
def train_random_forest_model(X, y):
    # Розділення на тренувальний та тестовий набори даних
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
random_state=42)

    # Створення та навчання моделі Random Forest
    model = RandomForestClassifier()
    model.fit(X_train, y_train)

    # Прогнозування на тестових даних
    y_pred = model.predict(X_test)

    # Оцінка точності моделі
    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 = f1_score(y_test, y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

    # Перехресна перевірка
    cv_scores = cross_val_score(model, X, y, cv=5)
    print("Cross-Validation scores:", cv_scores)
    print("Average Cross-Validation score:", np.mean(cv_scores))

def train_logistic_regression(X, y):
    # Розділення на тренувальний та тестовий набори даних
    X_train, X_test, y_train, y_test = train_test_split(
        X,
        y,
        test_size=0.6,
        random_state=42
    )

    # Створення та навчання моделі Логістичної регресії з підходом One-vs-Rest
    num_classes = len(np.unique(y))
    models = []
```

```
    reg_strengths = [0.1, 0.1, 0.1]  # Задайте сили регуляризації для кожного класу

    for class_idx in range(num_classes):
        y_train_class = y_train[:, class_idx]

        # Створення моделі Логістичної регресії з відповідною силою регуляризації
        model = LogisticRegression(C=0.1 / reg_strengths[class_idx])
        model.fit(X_train, y_train_class)
        models.append(model)

    # Прогнозування на тестових даних
    y_pred_prob = np.zeros((len(X_test), num_classes))
    for class_idx in range(num_classes):
        model = models[class_idx]
        y_pred_prob[:, class_idx] = model.predict_proba(X_test)[:, 1]

    # Вибір класу з найвищою ймовірністю
    y_pred = np.argmax(y_pred_prob, axis=1)

    # Оцінка точності моделі
    accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
    precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted', zero_division=0)
    recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
    f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)


if __name__ == "__main__":

    # Отримання шляху до файла відносно кореня проекту
    data_path = os.path.join(os.getcwd(), 'data', 'wine.data')

    # Завантаження даних з локального файла
    column_names = ['Class', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',
```

```
                    'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
'Proanthocyanins',
                    'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
'Proline']
    data = pd.read_csv(data_path, names=column_names)

    # Виведення таблиці даних
    df = pd.DataFrame(data, columns=column_names)
    print(df)

    # Розділення ознак та цілової змінної
    X = data.drop('Class', axis=1)
    y = data['Class']

    # Обробка пропущених значень
    imputer = SimpleImputer(strategy='mean')
    X_imputed = imputer.fit_transform(X)

    # Масштабування ознак
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_imputed)

    # Конвертація масиву NumPy у DataFrame
    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
    y_df = pd.DataFrame(y, columns=['Class'])

    # Кодування категоріальної змінної (Class)
    categorical_feature = ['Class']
    categorical_transformer = OneHotEncoder()
    class_encoder = ColumnTransformer(
        transformers=[
            ('cat', categorical_transformer, categorical_feature)
        ])
    y_encoded = class_encoder.fit_transform(y_df)

    # Кодування числових ознак
    numeric_features = X.columns
    numeric_transformer = StandardScaler()
    numeric_encoder = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_features)
        ])
```

```
    # Застосування кодування до ознак та цілової змінної
    X_processed = numeric_encoder.fit_transform(X_scaled_df)
    processed_data = pd.DataFrame(X_processed, columns=numeric_features)
    processed_data = pd.concat([processed_data, pd.DataFrame(y_encoded)], axis=1)

    # Виведення оброблених даних
    print(processed_data)

    # Виклик функції зі змінними X_scaled та y_encoded
    print('----------------------optimized_xgboost_model-----------------------
--')
    train_optimized_xgboost_model(X_scaled, y_encoded)
    print('-----------------------random_forest_model--------------------------
--')
    train_random_forest_model(X_scaled, y_encoded)
    print('-----------------------logistic_regression--------------------------
--')
    train_logistic_regression(X_scaled, y_encoded)
```