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: «**Напівкерований граничний бустинг**»

Виконав:
студент IV курсу, групи КІ-91
Вітковський Данило Олександрович _____

Керівник:
д.т.н., професор Синєглазов В.М. _____

Консультант з нормконтролю:
фахівець 1-ї категорії Гончарук М.М. _____

Консультант з економічного розділу:
доцент, к.е.н., Рощина Н.В. _____

Рецензент:
ст. викладач каф. АКІК НАУ, к.т.н.
Долгоруков С.О. _____

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

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Олена ЧУМАЧЕНКО

«___»_____2023 р.

ЗАВДАННЯ

на дипломну роботу студента

Вітковського Данила Олександровича

1. Тема роботи «Напівкерований граничний бустинг», керівник роботи **Синєглазов Віктор Михайлович**, доктор технічних наук, професор, затверджені наказом по університету від «30» травня 2023 р. № 2065-с

2. Термін подання студентом роботи 07.06.2023 року

3. Вихідні дані до роботи – двовимірні набори даних для бінарної класифікації різного просторового розташування: «банан», «зворотній банан», «концентричні кола», «два місяці» класичні, щільні та нещільні; набір даних медичної діагностики для класифікації пацієнтів з інфекційним ендокардитом.

4. Зміст роботи – 1. Штучний інтелект та перспективи його розвитку; 2. Напівкерований граничний бустинг; 3. Побудова модифікованого алгоритму напівкерованого граничного бустингу; 4. Використання запропонованого алгоритму в медичній діагностиці; 5. 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	Рощина Н.В.		
Нормоконтроль	Гончарук М.М.		

7. Дата видачі завдання _____

Календарний план

з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення літератури за темою роботи.	25.04.2023	Виконано
2.	Підготовка першого розділу.	30.04.2023	Виконано
3.	Проведення дослідження за темою «напівкерований граничний бустинг».	04.05.2023	Виконано
4.	Підготовка другого розділу.	12.05.2023	Виконано
5.	Розробка програмного продукту.	17.05.2023	Виконано
6.	Підготовка презентації доповіді.	25.05.2023	Виконано
7.	Підготовка третього розділу.	28.05.2023	Виконано
8.	Підготовка економічної частини.	30.05.2023	Виконано
9.	Оформлення дипломної роботи.	05.06.2023	Виконано

Студент **Данило ВІТКОВСЬКИЙ**

Керівник **Віктор СИНЄГЛАЗОВ**

РЕФЕРАТ

Дипломна робота: 92 сторінки, 33 рисунки, 13 таблиць, 1 додаток, 17 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, КЛАСИФІКАЦІЯ, БІНАРНА КЛАСИФІКАЦІЯ, БУСТИНГ, НАПІВКЕРОВАНЕ МАШИННЕ НАВЧАННЯ, НАПІВКЕРОВАНИЙ ГРАНИЧНИЙ БУСТИНГ.

Об’єкт дослідження – напівкерований граничний бустинг.

Під час створення систем підтримки прийняття рішень, нейронних мереж та інших систем машинного навчання, виникає проблема збору та правильної класифікації тренувальних даних. Залежно від типу даних, процес класифікації може бути складним, повільним або вартісним. Наприклад, класифікація медичних знімків вимагає глибокої медичної експертизи і значних ресурсів часу. Існує ризик помилкової класифікації, що вносить шум у тренувальні дані.

З метою вирішення цих викликів, були розроблені алгоритми напів-керованого навчання, що потребують меншого обсягу відмічених даних та здатні використовувати невідмічені дані. Одним з таких алгоритмів є напівкерований граничний бустинг, який дозволяє покращувати точність моделі бінарної класифікації через ітеративне навчання, використовуючи відмічені та невідмічені дані. Це дозволяє зекономити на попередній класифікації навчальних даних, не втрачаючи при цьому якості моделі.

Мета роботи – розробити покращення існуючих підходів до реалізації алгоритму напівкерованого граничного бустингу.

Практичне значення роботи полягає в отриманні високоефективного алгоритму для класифікації даних при невеликій кількості міток.

ABSTRACT

Bachelor thesis: 92 pages, 33 figures, 13 tables, 1 appendix, 17 sources.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, CLASSIFICATION, BINARY CLASSIFICATION, BOOSTING, SEMI-SUPERVISED MACHINE LEARNING, SEMI-SUPERVISED MARGIN BOOSTING.

The object of research is an algorithm of semi-supervised MarginBoost.

When creating decision support systems, neural networks, and other machine learning systems, the problem of collecting and correctly classifying training data arises. Depending on the type of data, the classification process can be complex, slow, or costly. For example, classifying medical images requires in-depth medical expertise and significant time resources. There is a risk of misclassification, which introduces noise into the training data.

In order to address these challenges, semi-supervised learning algorithms have been developed that require less labeled data and are able to utilize unlabeled data. One of these algorithms is Semi-Supervised MarginBoost, which improves the accuracy of a binary classification model through iterative learning using both labeled and unlabeled data. This allows to save on pre-classification of training data without losing the model performance.

The aim of the work is to develop an improvement of existing approaches to the implementation of the semi-supervised margin boosting algorithm.

The practical significance of this work is to obtain a highly efficient algorithm for data classification with a small number of labels.

ЗМІСТ

ВСТУП

При розробці систем підтримки прийняття рішень, нейронних мереж, й інших рішень штучного інтелекту постає питання збору тренувальних даних, на основі яких буде відбуватися навчання моделей. Тренувальні дані мають бути попередньо класифіковані людиною, до того ж правильно. Для деяких видів даних таке завдання є складним, повільним або високовартісним процесом. Наприклад, класифікація медичних знімків потребує професійних медичних навичок, знань і досвіду у певній сфері, набуття яких початково потребує років навчання. Вартість часу такого професіонала, відповідно, висока. До того ж, усе ще існує ймовірність неправильної класифікації, що додає шуму у тренувальні дані загалом.

Для вирішення цих проблем, було створено алгоритми напів-керованого навчання, що потребують набагато меншу кількість відмічених даних, бо здатні навчатися з використанням, у тому числі, не відмічених даних. Це дозволяє отримати результати, порівняні за якістю із алгоритмами керованого навчання, при цьому, зекономивши на попередній класифікації навчальних даних.

Одним із таких алгоритмів напівкерованого навчання є алгоритм Semi-Supervised MarginBoost. Його суть полягає у ітеративному покращенні точності моделі бінарної класифікації (яку можливо розширити до багатокласової класифікації), використовуючи як позначені, так і не позначені дані. Модель попередньо навчається на множині відмічених даних, потім використовується для класифікації частини невідмічених даних. Цей процес повторюється ітеративно, кожний раз покращуючи якість класифікації навчальної вибірки та якість моделі.

ПЕРЕЛІК ПОЗНАЧЕНЬ

i, N	Індекс і кількість прикладів у тренувальній вибірці
t, T	Індекс і кількість ітерацій алгоритму
m, M	Індекс і кількість гіпотез (якщо скінченна)
$(\boldsymbol{x}, y)$	Точка тренувальної вибірки і її значення
$\mathcal{X}, \mathcal{Y}$	Множина вхідних даних та позначень $\{\pm 1\}$ відповідно
S, L, U	Множина усіх даних, лише відмічених та лише невідмічених відповідно
g_t	Лінійна комбінація базових гіпотез після t ітерацій
$\boldsymbol{w} = (w_1, \dots, w_N)$	Вектор розподілу ваг точок тренувальної вибірки такий, що $w_i \geq 0, \sum_{i=1}^{N} w_i = 1$
$\mathcal{H}, h$	Множина базових гіпотез та її елемент
ε	Значення зваженої помилки гіпотези h
$\mathbb{I}(\cdot)$	Індикаторна функція: $\mathbb{I}(\text{так}) = 1, \mathbb{I}(\text{ні}) = 0$
т. т. т., к.	«тоді й тільки тоді, коли»

РОЗДІЛ 1. ШТУЧНИЙ ІНТЕЛЕКТ ТА ПЕРСПЕКТИВИ ЙОГО РОЗВИТКУ

1.1. Визначення штучного інтелекту та його ознак

Штучний інтелект (ШІ, англ. Artificial intelligence, AI) – це галузь комп’ютерних наук, яка займається створенням алгоритмів та систем, що здатні виконувати завдання, які зазвичай потребують людського інтелекту. Системи штучного інтелекту створені для навчання на досвіді, розпізнавання закономірностей та прийняття рішень на основі вхідних даних.

Основні ознаки штучного інтелекту включають:

- Здатність до навчання: системи ШІ можуть отримувати нові знання і навички із даних, або взаємодії з оточенням.
- Адаптація: системи ШІ здатні адаптувати свою поведінку до унікальних змін у своєму оточенні, навіть якщо раніше не стикалися із подібними умовами.
- Самостійність: системи ШІ можуть працювати без постійного нагляду зі сторони людини та стороннього втручання.

Штучний інтелект знаходить застосування в різних сферах, таких як медицина, освіта, фінанси, енергетика, виробництво, реклама та маркетинг, наукові дослідження, розваги тощо. Завдяки своїм когнітивним здібностям та навчанню, AI-системи здатні значно покращити ефективність та продуктивність різних галузей, автоматизувати рутинні та складні процеси, підтримувати прийняття рішень, розв'язувати складні проблеми.

Основні перспективи розвитку штучного інтелекту включають:

- Збільшення обчислювальної потужності: Продовження розвитку апаратного забезпечення, зокрема графічних процесорів (GPU), спеціалізованих інтегральних схем (ASIC) та квантових комп'ютерів, дозволить AI-системам

опрацьовувати великі обсяги даних та виконувати розрахунки швидше та ефективніше.

- Вдосконалення алгоритмів: Науковці продовжують вивчати нові методи машинного навчання та оптимізації алгоритмів, які дозволять системам ШІ ставати більш точними, надійними та адаптивними до різних ситуацій та завдань.
- Збільшення доступності даних: З ростом відкритих наборів даних та інтернету речей системи ШІ матимуть доступ до більшої кількості даних, які можуть використовуватися для навчання та вдосконалення.
- Удосконалення інтерфейсів людина-машини: Розвиток штучного інтелекту допоможе створити більш ефективні, інтуїтивні та ергономічні інтерфейси для взаємодії між людьми та комп'ютерами. Це включає розвиток нових методів взаємодії, таких як жестове керування, мовленнєве керування та інтерфейси, засновані на віртуальній або доповненій реальності.
- Етичні та регулятивні аспекти: З розвитком AI з'являються нові етичні, правові та регулятивні виклики, пов'язані з безпекою, приватністю, відповідальністю та впливом на робочий ринок. У майбутньому відбудеться формування норм та стандартів, які будуть регулювати використання ШІ та його вплив на суспільство.

1.2. Штучні нейронні мережі та їх класифікація

Штучні нейронні мережі (ШНМ, англ. Artificial neural networks, ANN) – це математичні моделі, які імітують структуру та функціонування біологічних нейронних мереж. Вони є основним компонентом багатьох систем штучного інтелекту та машинного навчання.

ШНМ можна класифікувати за архітектурою, способом навчання та застосування. Основні типи архітектур ШНМ:

- Одношаровий персептрон: Найпростіша ШНМ, яка складається з одного шару вузлів, або нейронів, з'єднаних з вхідними та вихідними вузлами. Вони можуть використовуватись для розв'язання простих задач класифікації та регресії.
- Багатошаровий персептрон: Має більше одного шару нейронів між вхідним та вихідним шарами. Можуть розв'язувати набагато складніші задачі, ніж одношарові персептрони. У нейронних мережах прямого поширення зв'язки організовані таким чином, що кожний нейрон даного рівня сприймає інформацію тільки від деякої непорожньої множини нейронів рівнем нижче. Назва мереж вказує на те, що у них є виділений напрямок поширення сигналів, що рухаються із входу, через один або більше прихованих шарів до виходу.
- Конкурентні нейронні мережі (англ. Competitive Neural Network) – це вид нейронних мереж, в яких нейрони в одному шарі взаємодіють між собою, змагаючись за можливість активуватися. Така взаємодія допомагає мережі виявляти та згруповувати властивості або закономірності вхідних даних. В мережах такого типу затухаючий ітераційний процес відбувається в межах одного шару нейронів, а ітераційна формула завжди має властивість зниження рівня вихідного сигналу, що викликає затухання слабких вихідних сигналів до рівня, нижчого від порога чутливості. Таким чином реалізується стратегія

«переможець забирає все», що зупиняє ітераційний процес у випадку перемоги одного нейрона або кластера.

- Рекурентні нейронні мережі (англ. Recurrent Neural Network, RNN) – тип ШНМ, який має зворотні зв’язки між нейронами, що дозволяє мережі обробляти послідовності даних та враховувати контекст інформації. RNN особливо корисні для завдань, пов’язаних з обробкою природної мови та часовими рядами.
- Згорткові нейронні мережі (ЗНМ, англ. Convolutional Neural Network, CNN) – архітектура ШНМ, що спеціалізується на обробці та аналізі зображень. Такі мережі використовують операції згортки та пулінгу для виявлення локальних особливостей у зображенні.

Рисунок 1.1. Структура ЗНМ.

ЗНМ використовують різновид багатошарових персептронів, розроблений так, щоб вимагати використання мінімального обсягу попередньої обробки. Згорткові мережі взяли за основу схему з’єднання нейронів зорової кори тварин. Окремі нейрони реагують на стимули лише в обмеженій області зорового поля, відомій як *рецептивне поле*. Рецептивні поля різних нейронів частково покривають усе зорове поле.

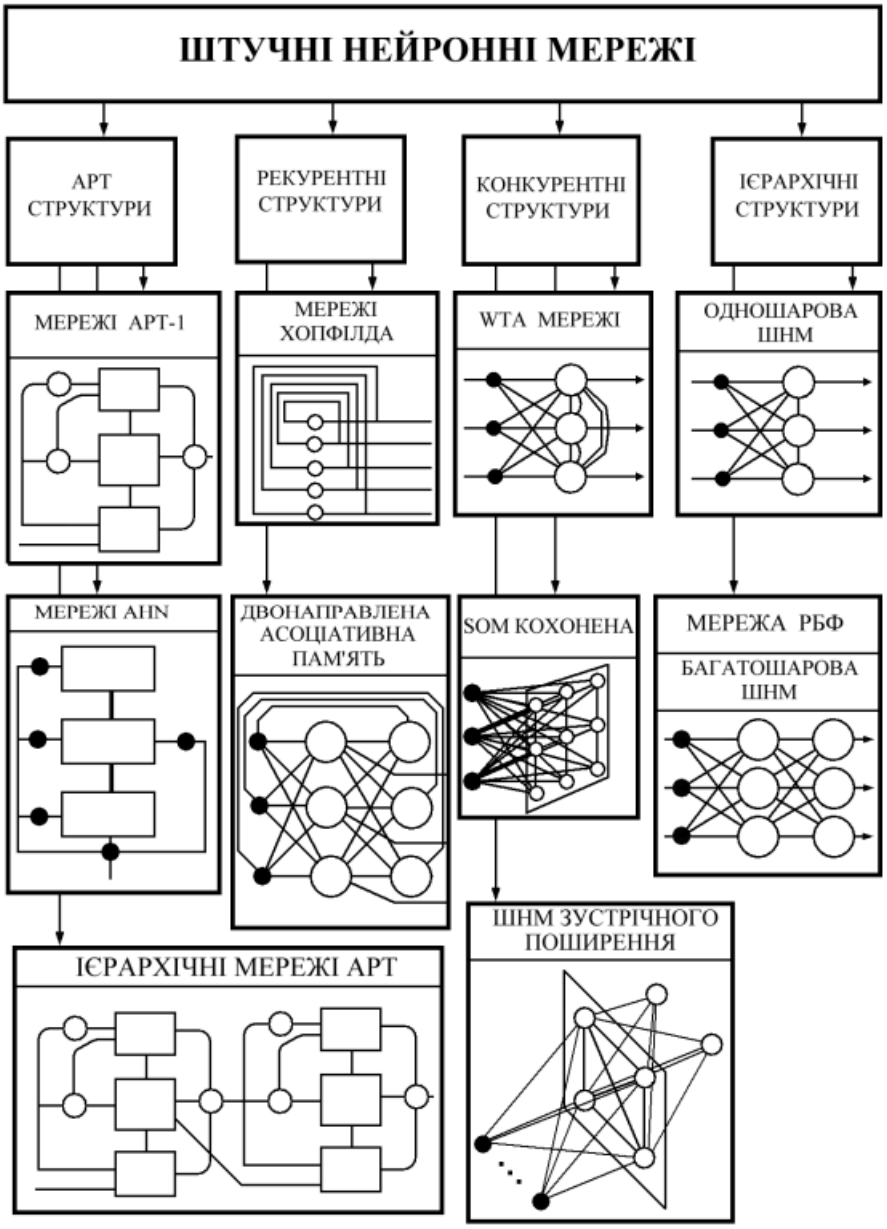

Рисунок 1.2. Архітектури ШНМ.

1.3. Методи машинного навчання та їх класифікація

Машинне навчання (МН, англ. Machine learning, ML) – це підгалузь штучного інтелекту, яка забезпечує здатність комп'ютерних систем навчатися та розвиватися без прямого програмування. Методи машинного навчання можна класифікувати за типом навчання та задачами, які вони вирішують. Основні типи машинного навчання:

- Навчання з учителем (англ. Supervised Learning): Методи, які використовують набір вхідних даних з відповідними мітками для навчання моделі передбачення або класифікації. Завдання можуть бути регресійними (передбачення неперервних значень) або класифікаційними (передбачення категорій).
- Навчання без учителя (англ. Unsupervised Learning): Методи, які використовують нерозмічені дані для виявлення закономірностей, структур та відносин між змінними. Завдання можуть включати кластеризацію (групування даних за схожістю), зменшення розмірності (виявлення значимих шаблонів або особливостей даних), аномалій або відхилень (ідентифікація аномальних або нетипових даних).
- Напівкероване навчання (англ. Semi-supervised Learning, SSL): Методи, які поєднують елементи навчання з учителем та без учителя, використовуючи малий обсяг розмічених даних та значний обсяг нерозмічених даних. Ці методи можуть використовуватися для підвищення точності та надійності моделей, зменшення потреби в розмітці даних або адаптації моделей до нових даних.
- Навчання з підкріпленням (англ. Reinforcement Learning): Методи, які базуються на процесі взаємодії агента з середовищем та отримання винагород за виконані дії, щоб максимізувати загальну винагороду або досягти певної мети. Агент навчається з часом, оцінюючи кожну дію та коригуючи свою

стратегію відповідно до отриманих винагород. Завдання можуть включати оптимізацію контролю, навігацію, графічні завдання тощо.

- Активне навчання (англ. Active Learning): Методи, які надають моделям можливість запитувати розмічені дані, які вони вважають найбільш інформативними або корисними для навчання. Це може зменшити кількість потрібних розмічених даних, прискорити процес навчання та покращити результати.

Ці категорії методів машинного навчання не є взаємовиключними, і деякі алгоритми або підходи можуть поєднувати аспекти з декількох категорій.

1.4. Необхідність застосування напівкерованого навчання

Напівкероване навчання має декілька переваг порівняно з традиційними методами навчання з учителем та без учителя. Ось деякі причини, чому напівкероване навчання важливе:

- Економія часу та ресурсів: розмітка даних може бути тривалою та дорогою процедурою. Напівкероване навчання дозволяє моделям використовувати невеликі розмічені набори даних разом з великими обсягами нерозмічених даних, що допомагає покращити точність прогнозування без необхідності розмічати весь набір даних.
- Використання реальних даних: у реальному світі дані часто мають неповну або відсутню розмітку. Напівкероване навчання дозволяє моделям використовувати ці реальні дані для покращення своєї роботи, навіть якщо розмітка відсутня або неповна.

- Покращення точності моделі: напівкероване навчання може поліпшити точність моделі, поєднуючи переваги навчання з учителем (використання розмічених даних для прогнозування) та без учителя (виявлення закономірностей у нерозмічених даних).
- Адаптація до нових даних: напівкероване навчання дозволяє моделям адаптуватися до нових даних, використовуючи накопичений досвід з розмічених та нерозмічених даних. Це забезпечує більшу гнучкість та стабільність роботи моделі.
- Зменшення упереджень: традиційні методи навчання з учителем можуть страждати від упереджень, пов'язаних з неповною або неточною розміткою. Напівкероване навчання забезпечує меншу залежність від розмічених даних та зменшує ризик упереджень.
- Надійність моделі: Напівкероване навчання може забезпечити захищеність моделі від шуму та відхилень у даних. Моделі, навчені за допомогою напівкерованого навчання, можуть краще справлятися з неточностями у розмічених даних, оскільки вони також вивчають структуру нерозмічених даних.
- Доменне знання: У деяких випадках, експерти можуть не мати достатньо часу або ресурсів для розмітки великих наборів даних. Напівкероване навчання може використовувати доменне знання експертів, забезпечуючи зв'язок між розміченими та нерозміченими даними, що покращує якість моделі.
- Виявлення нових шаблонів: Напівкероване навчання може допомогти виявити нові шаблони та закономірності у даних, які можуть бути непомітними при використанні тільки навчання з учителем або без учителя. Це може розширити можливості моделі та дозволити їй вирішувати більш складні та нетипові задачі.
- Перенос навчання (англ. Transfer Learning): Напівкероване навчання дозволяє використовувати знання, набуте під час навчання на одному наборі даних, для поліпшення роботи моделі на інших, споріднених задачах або наборах даних.

Це може прискорити процес навчання та зменшити потребу в додаткових розмічених даних.

У цілому, напівкероване навчання є важливим інструментом у галузі штучного інтелекту та машинного навчання, оскільки воно дозволяє створювати більш точні, ефективні та гнучкі моделі, які можуть адаптуватися до реального світу і вирішувати складні задачі з меншими витратами часу та ресурсів.

1.5. Постановка завдання навів-керованого навчання

Дано навчальну вибірку N прикладів, M з яких – невідмічені:

$S_L = \{(\boldsymbol{x}_1, \mathrm{y}_1), \dots, (\boldsymbol{x}_M, \mathrm{y}_\mathrm{M})\}$ – множина відмічених прикладів, $S_U = \{\boldsymbol{x}_{M+1}, \dots, \boldsymbol{x}_N\}$ – множина невідмічених прикладів.

Завдання: Навчити модель класифікувати нові вхідні дані з достатньо високою точністю, використовуючи ансамблевий алгоритм напівкерованого навчання «Напівкерований граничний бустинг» (Semi-supervised MarginBoost).

Вибіркою слугують дані про здоров’я людей що класифікуються за наявністю у людини такого захворювання серця, як «інфекційний ендокардит».

Критерієм для оптимізації є квадрат помилки навченої моделі на повністю розміченій валідаційній вибірці (ε^2).

РОЗДІЛ 2. НАПІВКЕРОВАНИЙ ГРАНИЧНИЙ БУСТИНГ

2.1. Напівкероване машинне навчання та його класифікація

Напівкероване навчання, як було згадано у підрозділі 1.3, є гібридним підходом до машинного навчання, в якому використовуються як розмічені, так і нерозмічені дані для навчання моделі. Такі методи дозволяють досягти кращих результатів, ніж лише навчання з учителем, або без учителя, особливо коли розмічені дані обмежені, або дорогі в отриманні.

Напівкеровані методи машинного навчання можна класифікувати за різними критеріями. Одним з основних критеріїв є тип взаємодії між розміченими та нерозміченими даними в процесі навчання. Згідно з цим критерієм, напівкеровані методи можна розділити на три основні групи:

1. Самонавчання (англ. Self-training): В цьому підході модель навчається на розмічених даних і використовує отримані знання для класифікації нерозмічених даних. Найбільш впевнені передбачення з нерозмічених даних додаються до навчальної вибірки, і процес повторюється, поки не буде досягнуто заданої кількості ітерацій або покращення.

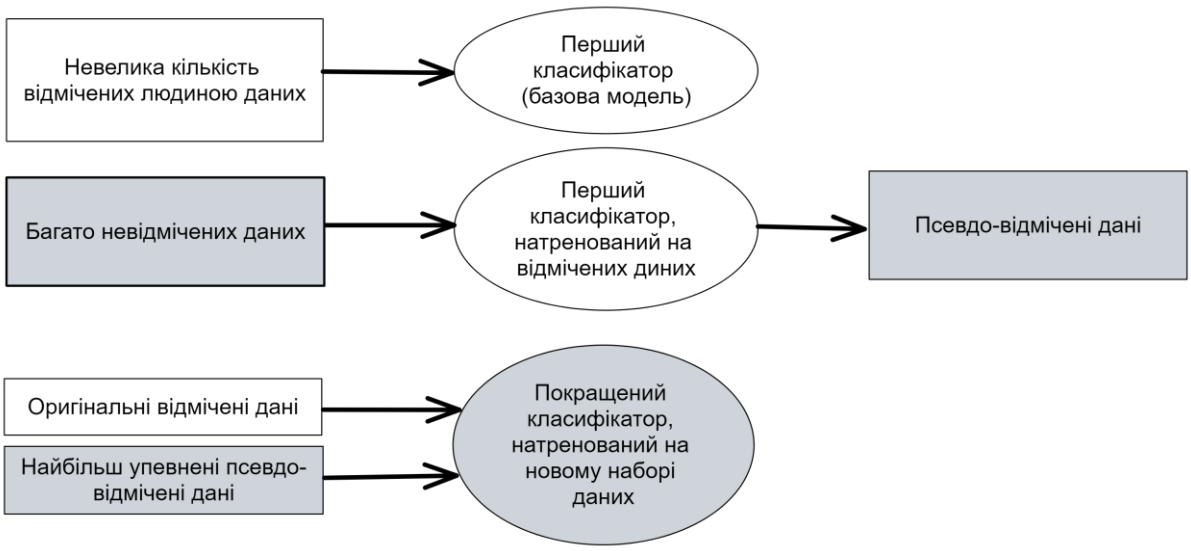

Рисунок 2.1. Самонавчання

Береться невелика кількість відмічених даних, на якій навчається початкова модель, використовуючи методи навчання з учителем.

Далі, застосовується процес, відомий як *псевдо-відмічення* – коли береться частково навчена модель та використовується, щоб зробити передбачення щодо решти даних, що досі невідмічені. Згенеровані таким чином позначення називаються «псевдо», оскільки створені на основі обмеженої кількості відмічених даних (розподіл відмічених даних може бути нерівномірним).

Після цього, обираються найбільш упевнені передбачення, наприклад більше 80%, та додаються до тренувальної вибірки для нової моделі.

Процес повторюється кілька ітерацій, кожний раз із усе більшою кількістю псевдо-відмічених даних.

2. Кооперативне навчання (англ. Co-training): В цьому підході використовуються дві або більше моделей, які навчаються одночасно на розмічених даних. Кожна модель передбачає мітки для нерозмічених даних, і найбільш впевнені передбачення використовуються для додавання до навчальної вибірки іншої моделі. Цей процес ітеративно повторюється, що дозволяє моделям навчатися одна від одної та зменшувати помилки.

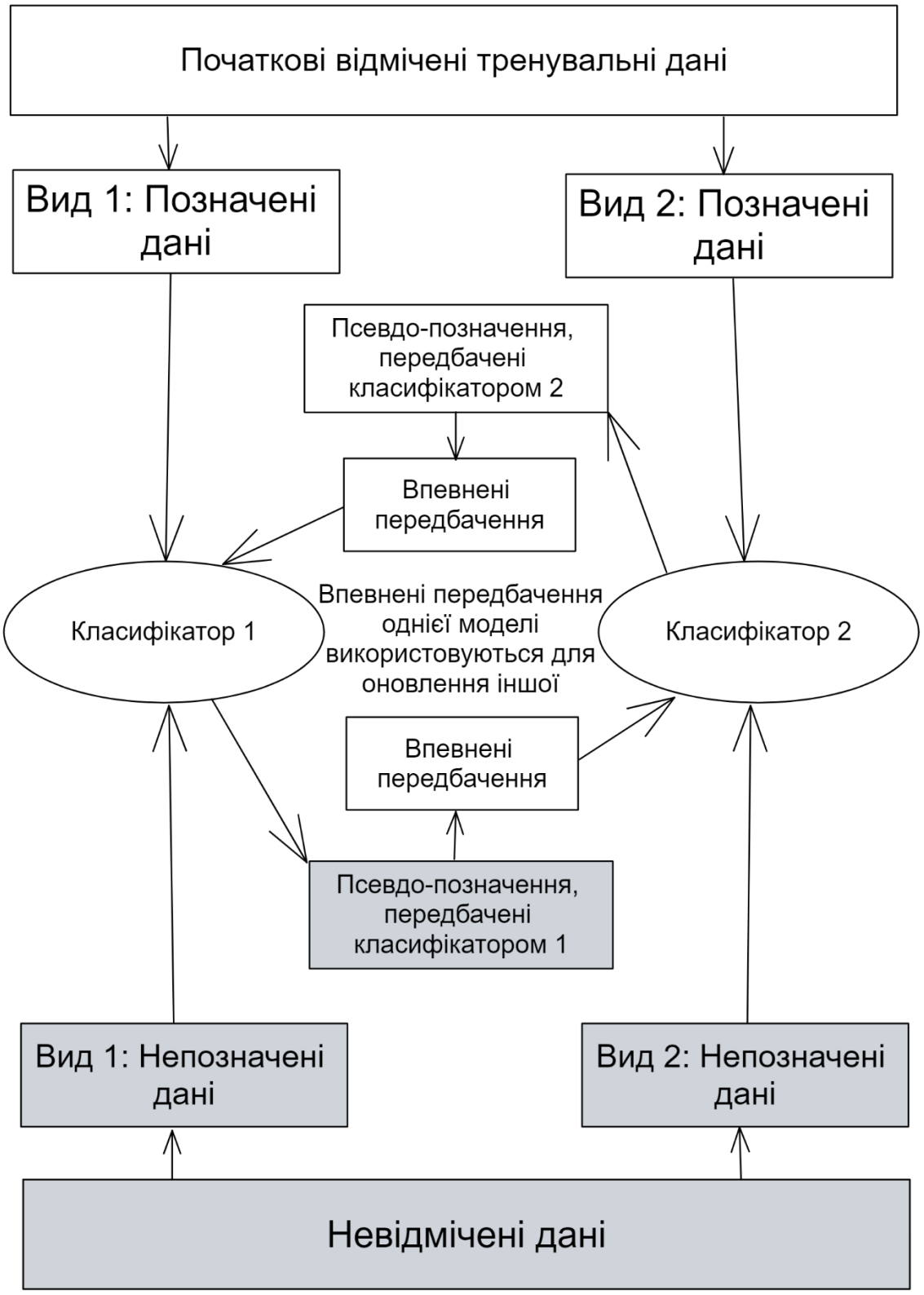

Рисунок 2.2. Кооперативне навчання

Будучи похідним від самонавчання і його покращеною версією, цей підхід, на відміну від звичайного, тренує декілька окремих класифікатори, заснованих на кількох окремих підвибірках даних.

Оригінальне дослідження кооперативного навчання [1] стверджує, що цей підхід може бути успішно використаним для, наприклад, задач класифікації наповнення веб-сторінок. Опис кожної веб-сторінки може бути розділений на дві частини: перша зі словами, що виникають на сторінці, а друга – з ключовими словами у посиланні, що ведуть до неї.

- Спочатку тренується окрема модель для кожного вигляду даних за допомогою малої кількості відмічених даних.
- Далі додається більший пул невідмічених даних для отримання псевдоміток.
- Класифікатори сумісно навчають один одного, використовуючи найбільш впевнені псевдо-позначення. Якщо перший класифікатор упевнено передбачає справжню мітку для прикладу, у той час, коли другий робить помилку, тоді дані з упевненими псевдомітками, присвоєними першим класифікатором, оновлюють другий класифікатор, і навпаки.
- Останній крок включає комбінування передбачень із двох оновлених класифікаторів, щоб отримати один результат.

3. Графові методи (англ. Graph-based methods): Ці методи представляють дані у вигляді графу, де вершини відповідають об’єктам, а ребра відображають схожість між ними. Розмічені вершини використовуються як база для розповсюдження міток на нерозмічені вершини. Процес розповсюдження може здійснюватися через ітеративні алгоритми, такі як Label propagation, або через глобальні оптимізаційні методи, які враховують структуру графа, такі як Label spreading або Harmonic function. Графові методи надають важливу інформацію про структуру даних і можуть використовуватися для адаптації моделі до нових даних або доменів.

Кожен із цих підходів має свої переваги та недоліки, і вибір конкретного методу залежить від характеристик задачі, доступності розмічених та нерозмічених даних та комп’ютерних ресурсів.

2.2. Огляд методів напівкерованого граничного бустингу

2.2.1. Огляд складових при навчанні з учителем

Бустинг – це сімейство ансамблевих мета-алгоритмів машинного навчання, що перетворюють «слабких учнів» на «сильних».

У даному контексті, «слабким учнем» називається модель, що вміє надавати правильну відповідь лише трохи краще за випадкове вгадування. Для випадку бінарної класифікації – вгадує правильний клас з імовірністю більше 50%. Тоді «сильним учнем» є добре натренована модель машинного навчання, що прогнозує правильну відповідь із достатнім рівнем достовірності, набагато кращим за випадкове вгадування.

Бустинг об’єднує слабкі учні в ансамблі, де кожна така модель має вплив на остаточну відповідь.

Граничний бустинг (англ. MarginBoost) це алгоритм, чия мета – максимізувати відступ (margin) між точками тренувальних даних та межею прийняття рішень, що розділяє два класи. Цей алгоритм також можливо узагальнити до багатокласового випадку, використовуючи підхід «один проти всіх» – навчання кількох моделей (по моделі для кожного класу) де кожна модель навчена відрізняти даний клас від решти, але надалі буде розглядатися лише бінарний випадок.

В основі MarginBoost лежить класичний алгоритм AdaBoost [15], результатом роботи якого є набір слабких учнів, що об’єднуються у зважену суму виходів кожного учня, отримуючи таким чином сильного учня.

При навчанні класифікаторів, у кожного навчального прикладу є певна вага w_i^t. Ця вага відповідає тому, наскільки важливо щоб слабкий класифікатор правильно класифікував конкретний приклад: чим більше вага, тим більше уваги

буде приділено даному прикладу при навчанні слабкого учня. На початку навчання усі приклади мають однакому «важливість». Із кожною ітерацією ті приклади, що були класифіковані неправильно будуть збільшувати вагу, а ті, що були класифіковані правильно – відповідно зменшувати вагу. Таким чином, кожний наступний слабкий учень буде навчатися, враховуючи помилки попередніх слабких учнів.

Результатом навчання слабкого класифікатора є деяка гіпотеза $h_t: \mathcal{X} \to \mathcal{Y}$, де t – номер ітерації, $\mathcal{X}$ – простір вхідних даних, а $\mathcal{Y}$ – простір відповідей, що у випадку бінарної класифікації дорівнює $\{-1,1\}$.

В залежності від розміру помилки ε_t слабкого учня, йому надається відповідна вага α_t. Помилка обчислюється, як сума ваг неправильно класифікованих прикладів. Чим краще даний слабкий учень класифікує дані, тим більша у нього вага. Ця вага враховується при роботі алгоритму: кожний голос враховується із відповідною вагою, і який клас набрав більше умовних одиниць, той клас і буде відповіддю алгоритму. В результаті отримуємо лінійну комбінацію гіпотез базових класифікаторів $g: \mathcal{X} \to \mathcal{Y}$.

Алгоритм 1. AdaBoost

1) **Вхід**: $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$ – множина відмічених тренувальних даних; T – кількість ітерацій/базових класифікаторів
2) **Ініціалізуємо**: $\forall i = \overline{1, N}, w_i^0 = \frac{1}{N}$ –нормалізовані ваги тренувальних прикладів
3) **Повторюємо для** $t = \overline{1, T}$:
    а) Тренуємо слабкого учня (алгоритм 2), отримуємо гіпотезу $h_t: \mathcal{X} \to \mathcal{Y} = \{\pm 1\}$
    б) Обчислюємо зважену помилку:

$$\varepsilon_t = \sum_{i=1}^{N} w_i^t \cdot \mathbb{I}(h_t(\boldsymbol{x}_\mathrm{i}) \neq y), \tag{2.1}$$

де N – кількість тренувальних прикладів, w_i^t – вага (важливість правильної класифікації) тренувального прикладу i на ітерації t, h_t – гіпотеза нового слабкого класифікатора, $\mathbb{I}(\cdot)$ – індикаторна функція.

в) **Якщо** $\varepsilon_t \in \{0,1\}$, **то** $T = 1$, $g(\boldsymbol{x}) = \begin{cases} h_t(\boldsymbol{x}), \varepsilon_t = 0 \\ -h_t(\boldsymbol{x}), \varepsilon_t = 1 \end{cases} =$

$= (1 - 2\varepsilon_t)h_t(\boldsymbol{x})$.

г) Обчислюємо вагу даного базового класифікатора:

$$\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t} \tag{2.2}$$

д) Оновлюємо ваги прикладів:

$$w_i^{t+1} = \frac{w_i^t \, exp\big(\alpha_t(2 \cdot \mathbb{I}(\mathrm{h_t}(\boldsymbol{x}_i) \neq \mathrm{y}) - 1)\big)}{\sum_{k=1}^{N} w_k^t \, exp\big(\alpha_t(2 \cdot \mathbb{I}(\mathrm{h_t}(\boldsymbol{x}_k) \neq \mathrm{y}) - 1)\big)}, \tag{2.3}$$

де α_t – вага (кількість впливу) базового класифікатора h_t, $exp(\cdot)$ –функція експоненти $exp(x) = e^x$, $\mathbb{I}(\cdot)$ – індикаторна функція.

4) **Вихід**: лінійна комбінація гіпотез:

$$\mathrm{g}(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t \, h_t(\boldsymbol{x}) \tag{2.4}$$

Щоб отримати результат класифікації: $sgn\big(g(\boldsymbol{x})\big)$.

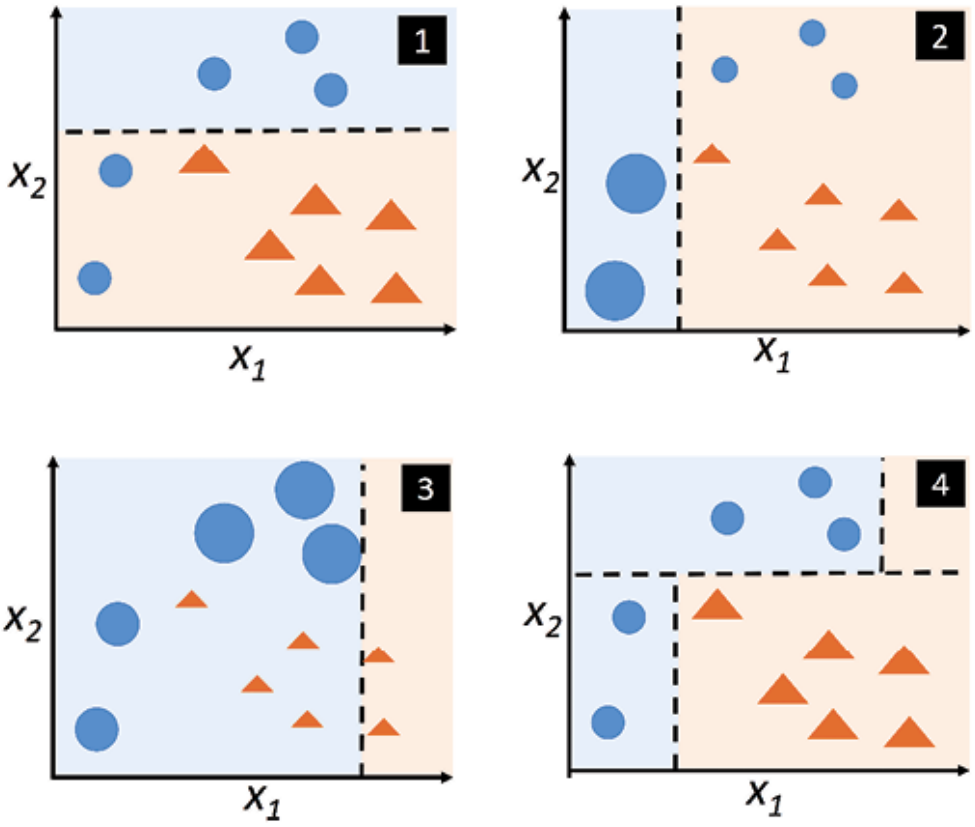

Рисунок 2.3. Слабкі учні (1-3) та їх комбінація (4). Розмір мітки відповідає вазі прикладу.

Базовими класифікаторами часто виступають дерева рішень з одним вузлом рішень, так звані «ростки рішень». Але у якості базового класифікатора також можна використовувати перцептрон, машину опорних векторів тощо. Однак, якщо базовим класифікатором використовувати сильного учня, то існує великий ризик перенавчання.

Дерево рішень – це алгоритм, що будує деревовидну структуру, що складається з вузлів рішень та листків. Вузли рішень оцінюють одну ознаку точки даних, перевіряючи, чи є вона більше певного порогу (для булевих ознак «ні» $= 0$, «так» $= 1$, поріг $= 0.5$). Якщо так, то курсор іде у праву гілку, інакше – у ліву (можливо й навпаки, залежить від реалізації). У кінці гілки знаходяться листки – вузли без дочірніх вузлів. Вони представляють собою рішення дерева.

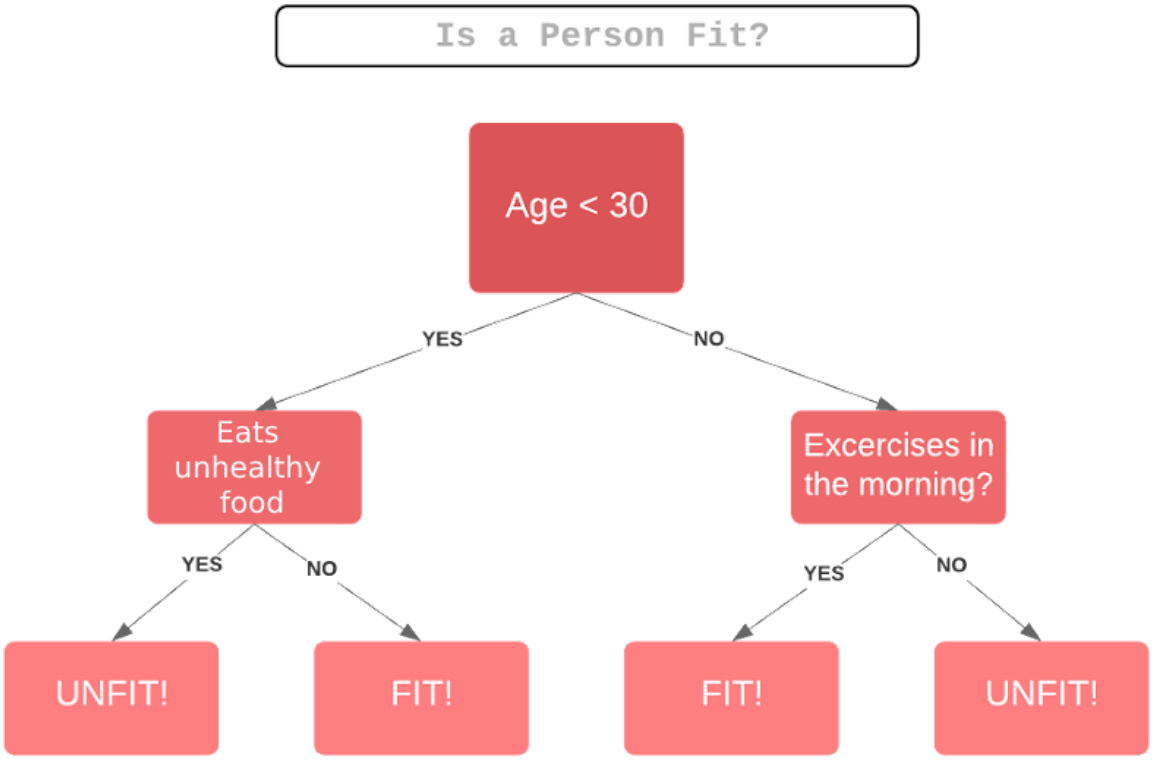

Рисунок 2.4. Приклад дерева рішень.

Розглянемо алгоритм навчання ростку рішень:

Алгоритм 2. Росток рішень з урахуванням ваг прикладів.

1) **Вхід**: $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$ – множина відмічених тренувальних даних; $\boldsymbol{w} = \{\boldsymbol{w_1}, \dots, \boldsymbol{w_N}\}$ – ваги прикладів.
2) Будуємо нову вибірку S потужністю N так, що кожний з прикладів потрапляє у нову вибірку з імовірністю w_i.
3) **Повторюємо для** $\boldsymbol{m} = \overline{\boldsymbol{1, M}}$, де M – кількість ознак:
    а) Якщо ознака x_i^m категоріальна – перетворюємо її у числову. Подальші кроки виконуємо для числової ознаки
    б) Залишаємо лише унікальні входження пар (x_i^m, y_i)
    в) Сортуємо вибірку за ознакою x_i^m
    г) Будуємо множину чисел порогів-кандидатів для вузла рішень з середніх арифметичних значень сусідніх x^m:

$$\theta_k = \frac{x_k^m + x_{k+1}^m}{2}, k = \overline{1, N-1} \quad (2.5)$$

д) Рахуємо кількість правильно та неправильно класифікованих прикладів відповідно до класу (якщо $x_i^m > \theta_k$, відносимо приклад до класу 1, інакше – до класу 2):

$\alpha_k^{\text{correct}}$ – кількість прикладів, правильно віднесених до класу 1 при значенні порогу θ_k;

$\alpha_k^{\text{incorrect}}$ – кількість прикладів, неправильно віднесених до класу 1 при значенні порогу θ_k;

β_k^{correct} – кількість прикладів, правильно віднесених до класу 2 при значенні порогу θ_k;

$\beta_k^{\text{incorrect}}$ – кількість прикладів, неправильно віднесених до класу 2 при значенні порогу θ_k.

е) Для кожного значення порогу-кандидату обчислюємо коефіцієнт нечистоти – величина, що відповідає тому, наскільки добре вузол рішення розділяє два класи. Одним з варіантів критерію може бути індекс Джині (англ. Gini index):

$$I_k^\alpha = 1 - \left(\frac{\alpha_k^{\text{correct}}}{N_\alpha}\right)^2 - \left(\frac{\alpha_k^{\text{incorrect}}}{N_\alpha}\right)^2; \quad (2.6)$$

$$I_k^\beta = 1 - \left(\frac{\beta_k^{\text{correct}}}{N_\beta}\right)^2 - \left(\frac{\beta_k^{\text{incorrect}}}{N_\beta}\right)^2; \quad (2.7)$$

$$I_k = \frac{N_\alpha}{N_\alpha + N_\beta} I_k^\alpha + \frac{N_\beta}{N_\alpha + N_\beta} I_k^\beta, \quad (2.8)$$

де $N_\alpha = \alpha_k^{\text{correct}} + \alpha_k^{\text{incorrect}}$ – кількість прикладів, віднесених до класу 1, $N_\beta = \beta_k^{\text{correct}} + \beta_k^{\text{incorrect}}$ – кількість прикладів, віднесених до класу 2.

ж) Серед отриманих значень порогу, обираємо те, якому відповідає найменше значення коефіцієнту нечистоти.

4) **Вихід**: класифікатор для ознаки із найменшим значенням індексу нечистоти.

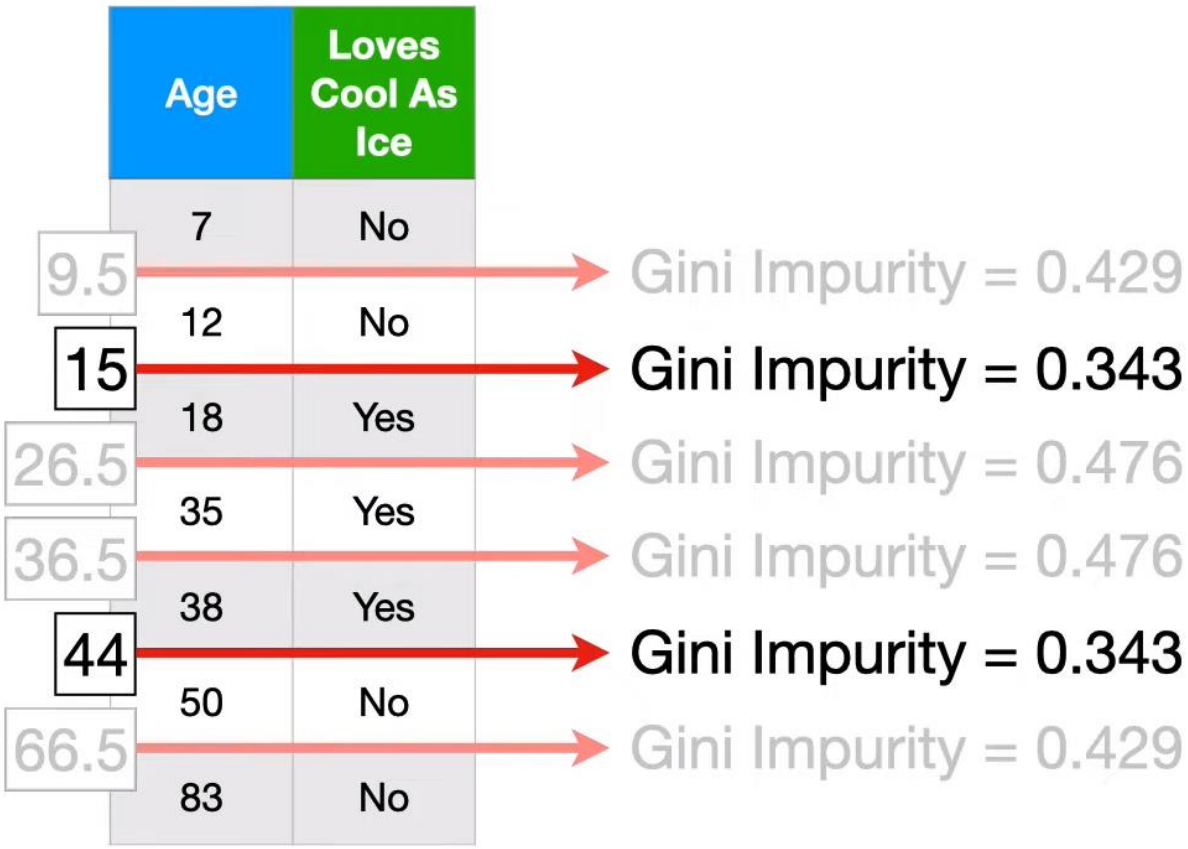

Рисунок 2.5. Обчислення значення нечистоти для числових ознак відносно порогу-кандидату.

Щоб узагальнити AdaBoost до MarginBoost, потрібно щоб базові класифікатори із більшим значенням відступу мали більший вплив ніж ті, що з меншим значенням відступу.

Оскільки межа прийняття рішень може мати складну форму, і деякі ознаки можуть бути не числовими, для обчислення характеристики відступу можливо використовувати не лише мінімальне значення відстані від точки даних до межі, а й певні характеристики

Одним з можливих визначень критерію для відступу для гіпотези є наступне:

$$\gamma_t = \sum_{i=1}^{N} w_i^t y_i h_t(\boldsymbol{x}_i), \tag{2.9}$$

де w_i^t – вага прикладу i на ітерації t (важливість його правильної класифікації), y_i – справжня мітка прикладу i, $h_t(\boldsymbol{x}_i)$ – результат класифікації прикладу i слабким учнем t.

За цим визначенням, відступ є додатнім, якщо більшість прикладів класифіковані правильно, і від'ємним, якщо більшість прикладів класифікована неправильно. Оскільки ваги прикладів нормовані, то $\gamma_t \in [-1,1]$.

Також, у випадку базового класифікатора, такого як ростку рішень, для конкретної гіпотези розрахувати відступ є тривіальною задачею:

$$\gamma_t = \min_{i=\overline{1,N}} \left| x_i^k - \theta_k^t \right|, \tag{2.10}$$

де k – індекс ознаки, за якою іде розділення, θ_k^t – поріг класифікації ростку t за ознакою k.

Тоді маємо наступний алгоритм MarginBoost, запропонований у [3] під назвою AdaBoost_ν^*, де ν – це параметр точності для максимізації відступу:

Алгоритм 3.MarginBoost у вигляді AdaBoost_ν^*.

1) **Вхід:** $S = \langle (\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N) \rangle$, кількість ітерацій T, бажана точність ν
2) **Ініціалізуємо:** $w_i^1 = \frac{1}{N}, \forall i = \overline{1, N}$
3) **Виконуємо для** $t = \overline{1, T}$:
   a) Тренуємо слабкий класифікатор на $\{S, \boldsymbol{w}^t\}$ і отримуємо гіпотезу $h_t: \mathcal{X} \to [-1,1]$;

б) Обчислити значення межі γ_t для h_t: $\gamma_t = \sum_{i=1}^{N} w_i^t y_i h_t(\boldsymbol{x}_i)$, де w_i^t – вага прикладу i на ітерації t (важливість його правильної класифікації), y_i – справжня мітка прикладу i;

в) **Якщо** $|\gamma_t| = 1$, **тоді** $\alpha_1 = \text{sgn}(\gamma_t), h_1 = h_t, \text{T} = 1$; **виходимо з циклу**

г) $\gamma_t^{min} = \underset{r=\overline{1,t}}{min}\gamma_r$; $\rho_t = \gamma_t^{min} - \nu$;

д) Встановлюємо значення ваги слабкого учня

$$\alpha_t = \frac{1}{2}\left(\ln\frac{1+\gamma_t}{1-\gamma_t} + \ln\frac{1+\rho_t}{1-\rho_t}\right) \tag{2.11}$$

е) Оновлюємо ваги: $w_i^{t+1} = \frac{w_i^t \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{\sum_{n=1}^{N} w_n^t \exp(-\alpha_t y_n h_t(\boldsymbol{x}_n))}$

4) **Вихід:** $\text{g}(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$. Щоб отримати клас, обчислюємо $sgn\big(g(\boldsymbol{x})\big)$.

Як видно, у формулі (2.11) для ваги класифікатора з'являється новий доданок $\ln\frac{1+\rho_t}{1-\rho_t}$, що відповідає зв'язку між величиною відступу та важливістю даного слабкого учня: $\rho_t \in [-1,1]$, чим ближче значення ρ_t до 1, тим більша вага слова даного слабкого учня.

Однак, більшість робіт, присвячених напівкерованому граничному бустингу, покладаються на інший підхід до реалізації MarginBoost, а саме на ідею градієнтного спуску функціоналу оцінки відступу у просторі лінійних комбінацій гіпотез базових класифікаторів [4].

Припустимо, що приклади $(\boldsymbol{x}, y)$ утворені випадковим чином, відповідно до випадкового розподілу $\mathcal{D}$ на просторі $\mathcal{X} \times \mathcal{Y}$, де $\mathcal{X}$ – простір вимірів, а $\mathcal{Y} = \{\pm 1\}$ – простір міток. Метою алгоритму є отримання класифікатору на основі зваженої лінійної комбінації базових класифікаторів $sgn\big(g(\boldsymbol{x})\big)$:

$$g(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}),$$

де $h_t: \mathcal{X} \to \mathcal{Y}$ – базовий класифікатор із деякого фіксованого простору класифікаторів $\mathcal{H}$, $\alpha_t \in \mathbb{R}$. Відступ (margin) прикладу $(\boldsymbol{x}, y)$, відповідно до класифікатора $sgn\big(g(\boldsymbol{x})\big)$ визначається як $y \cdot g(\boldsymbol{x})$.

При тому, що $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$ – вибірка прикладів, утворених відповідно до розподілу $\mathcal{D}$, метою є побудувати лінійну комбінацію класифікаторів виду, описаного вище, таку, що ймовірність неправильної класифікації випадкового прикладу $P_{\mathcal{D}}\big(sgn\big(g(\boldsymbol{x})\big) \neq y\big)$ є малою. Оскільки розподіл $\mathcal{D}$ невідомий і дано лише тренувальну вибірку S, використовується підхід знаходження лінійної комбінації класифікаторів, що мінімізує вибіркове середнє деякої функції оцінки відступу. Тобто, для тренувальної вибірки S шукаємо g таке, що функціонал $C: Lin(\mathcal{H}) \to \mathbb{R}$

$$C(g) = \frac{1}{N} \sum_{i=1}^{N} c\big(\rho(g(\boldsymbol{x}_i), y_i)\big) \tag{2.12}$$

є мінімальним для деякої відповідної функції оцінки $c: \mathbb{R} \to \mathbb{R}$, де $\rho(x, y) = xy$.

Одним зі способів побудувати лінійну комбінацію класифікаторів, що оптимізує (2.12) є градієнтний спуск у просторі функцій.

На рівні абстракцій, можна розглядати базові гіпотези $h_t \in \mathcal{H}$ та їхні лінійні комбінації $g_t \in Lin(\mathcal{H})$ як елементи простору внутрішніх добутків. Внутрішній добуток функцій F та G визначається як:

$$\langle F, G \rangle = \frac{1}{N} \sum_{i=1}^{N} F(x_i) G(x_i)\,, \forall F, G \in Lin(\mathcal{H}) \tag{2.13}$$

Тепер, припустимо, що маємо функцію $g \in Lin(\mathcal{H})$ і нам потрібно знайти нову функцію $h \in \mathcal{H}$ щоб додати її до g, аби зменшити оцінку $C(g + \epsilon h)$ для деякого малого числа ϵ. У термінах простору функцій, потрібно знайти такий «напрямок» h, що $C(g + \epsilon h)$ зменшується якомога швидше. Бажаним напрямком є від'ємна функціональна похідна від C у точці g, $-\nabla C(g)(\boldsymbol{x})$, яку можна наближено виразити як:

$$\nabla C(g)(\boldsymbol{x}) = \begin{cases} \dfrac{y_i c'\left(y_i g(\boldsymbol{x}_i)\right)}{N}, \text{якщо } \boldsymbol{x} = \boldsymbol{x}_i, \\ 0, \text{інакше} \end{cases} \tag{2.14}$$

де $c'(z)$ – похідна функції оцінки відступу відносно z. Оскільки існує обмеження на те, що функція $h \in \mathcal{H}$, у загальному випадку неможна обрати $h = -\nabla C(g)$, бо це не гарантує $g \in \mathrm{Lin}(\mathcal{H})$. Тому обираємо h таким чином, що максимізує $-\langle \nabla C(g), h \rangle$. Це можна пояснити тим, що, до першого порядку ϵ,

$$C(g + \epsilon h) \approx C(g) + \epsilon \langle \nabla C(g), h \rangle, \tag{2.15}$$

і отже найбільше зменшення оцінки для h виникне при максимізації $-\langle \nabla C(g), h \rangle$.

Умовою зупинки алгоритму є $-\langle \nabla C(g_t), h_{t+1} \rangle \leq 0$, тобто коли слабкий класифікатор повертає базову гіпотезу h_{t+1}, яка більше не вказує у напрямку схилу функціоналу оцінки $C(g)$. Отже, алгоритм зупиняється коли, до першого порядку, крок у просторі функцій у напрямку базової гіпотези, що повернув слабкий учень, збільшив би значення оцінки, не зважаючи на довжину кроку.

2.2.2. Огляд підходів до напівкерованого граничного бустингу

Вперше ідею напівкерованого граничного бустингу (англ. Semi-supervised MarginBoost, SSMBoost) було запропоновано та реалізовано у [2]. Ідея алгоритму полягає у застосуванні процедури градієнтного спуску до функціоналу оцінки відступу у просторі лінійних комбінацій гіпотез (як описано вище у 2.2.1) та використанні псевдоміток у вигляді наближеної оцінки. Максимізація $-\langle \nabla C(g_t), h_{t+1} \rangle$ еквівалентна максимізації нової величини $J_t^S = J_t^L + J_t^U$.

$$C(g_t) = \sum_{i \in L} c\big(\rho_L(g_t(\boldsymbol{x}_i), y_i)\big) + \sum_{i \in U} c\left(\rho_U\big(g_t(\boldsymbol{x}_i)\big)\right), \tag{2.16}$$

$$J_t^L = \sum_{\mathrm{i} \in \mathrm{L}} \mathrm{w_i^t y}_i h_t(\boldsymbol{x}_i), \tag{2.17}$$

$$J_t^U = \sum_{\mathrm{i} \in \mathrm{L}} \mathrm{w_i^t \hat{y}}_i h_t(\boldsymbol{x}_i), \tag{2.18}$$

$$\hat{y}_i = \frac{\partial \rho_U\big(g_t(\boldsymbol{x}_i)\big)}{\partial g_t(\boldsymbol{x}_i)}, \tag{2.19}$$

$$w_i^t = \begin{cases} \dfrac{c'\big(\rho_L(g_t(\boldsymbol{x}_i), y_i)\big)}{|w_{t-1}|_1} \text{ якщо } i \in L \\ \dfrac{c'\left(\rho_U\big(g_t(\boldsymbol{x}_i)\big)\right)}{|w_{t-1}|_1} \text{ якщо } i \in U \end{cases} \tag{2.20}$$

де $\rho_U(\cdot)$ – функція відступу для непозначеного прикладу, $\rho_L(\cdot,\cdot)$ – функція відступу для відміченого прикладу.

Розглядалося два види функцій відступу для невідмічених даних:

$$\rho_U^g\big(g_t(\boldsymbol{x})\big) = \frac{g_t(\boldsymbol{x})+1}{2} g_t(\boldsymbol{x}) = g_t(\boldsymbol{x})^2;$$
$$\frac{\partial \rho_U^g\big(g_t(\boldsymbol{x})\big)}{\partial g_t(\boldsymbol{x})} = 2g_t(\boldsymbol{x}), \tag{2.21}$$

що випливає з математичного сподівання відступу $\mathbb{E}_y \rho_L(g_t(\boldsymbol{x}), y)$.

Іншим способом визначення розширення до відступу є пряме використання максимуму апостеріорної оцінки справжнього відступу. Ця оцінка залежить від знаку виходу класифікатора:

$$\rho_U^s\big(g_t(\boldsymbol{x})\big) = g_t(\boldsymbol{x})\, sgn\big(g_t(\boldsymbol{x})\big) = |g_t(\boldsymbol{x})|;$$
$$\frac{\partial \rho_U^s\big(g_t(\boldsymbol{x})\big)}{\partial g_t(\boldsymbol{x})} = sgn\big(g_t(\boldsymbol{x})\big) \tag{2.22}$$

У якості базового класифікатора пропонувалося використовувати моделі сумішей (тут – Гаусівські моделі сумішей, тобто використовуючи нормальний випадковий розподіл), оскільки вони пристосовані до роботи з невідміченими даними. Вони відносяться до алгоритмів кластеризації і є алгоритмом навчання без учителя. Суть моделі суміші полягає у тому, що простір даних моделюється як суміш декількох випадкових розподілів одного виду у різних пропорціях з різними параметрами, і приналежність точки даних кластеру визначається через те, ймовірність приналежності точки до якого класу є найбільшою.

Щільність розподілу суміші визначається як:

$$f(\boldsymbol{x}; \boldsymbol{\Phi}) = \sum_{k=1}^{K} p_k f_k(\boldsymbol{x}; \theta_k), \tag{2.23}$$

де θ_k – параметр розподілу кластеру k, p_k – пропорція суміші, $p_k > 0$, $K = K_1 + K_2$ – кількість кластерів, при тому, що перші K_1 кластерів належать класу 1, а останні K_2 – класу 2.

Для максимізації логарифмічної правдоподібності (критерію якості кластеризації) використовується EM (Expectation Maximization) алгоритм.

У висновках до роботи було зазначено, що $c(x) = exp(-x) = e^{-x}$. Порівнювалася якість роботи AdaBoost при навчанні на лише відміченій частині вибірки, SSMBoost з використанням ρ_U^s та ρ_U^g, а також окремо алгоритм для базових класифікаторів. Відсоток невідмічених даних становив 0%, 50%, 75%, 90% та 95%. Кількість ітерацій T=100, без спеціальної обробки перенавчання. Перезапускали 100 разів для випадкових початкових рішень.

У випадку 95% відсутніх міток (що є найбільш реалістичною ситуацією) найменшу середню помилку мав SSMB з ρ_U^s.

У іншій роботі [9] пропонується покращення даного алгоритму через поступове використання малої кількості «сильних» невідмічених даних.

Стверджується, що невідмічені дані не завжди допомагають підчас напівкерованого граничного бустингу. Точніше кажучи, нема гарантій, що додавання U, вибірки невідмічених даних, до тренувальної вибірки $S = L \cup U$ призводить до покращення якості класифікації. Таким чином, якщо існує можливість дізнатися про рівень «впевненості» при класифікації U, то можливо було би включати у тренувальну вибірку лише найбільш інформативні дані. Така ідея була використана у SemiBoost [12], де автори використовували попарну подібність щоб керувати вибором підвибірки U на кожній ітерації і призначенні міток. У [9] пропонується використовувати поступове відмічення, аніж порційне.

У якості алгоритму передбачення міток невідмічених даних використовується кластеризація за правилом k найближчих сусідів – обирається k найближчих сусідів до точки даних і результуючим класом є той, сусідів з якого більше. Число k зазвичай невелике, і якщо $k = 1$, то відповідно, точка даних відноситься до найближчого класу.

Порівнюючи із тим, як працюють SemiBoost та поточний варіант SSMB, вони по-різному обирають «сильні» приклади серед невідмічених даних та по-різному визначають псевдомітки. У SemiBoost постійно обираються 10% серед усіх невідмічених даних, тоді як у SSMBoost поступово обирається 10% невідмічених прикладів серед доступних на даний момент. SemiBoost визначає псевдомітки, спираючись на матрицю подібності, у той час як SSMB визначає їх, за допомогою кластеризації (тут – k найближчих сусідів). Отже, спираючись на те, що було розглянуто вище, пропонується наступна покращена версія SSMB:

Алгоритм 4. Покращений SSMB за [9]

1) **Ініціалізуємо:** $\mathrm{g}_0(\boldsymbol{x}) = 0$, $w_i^0 = \frac{1}{N_L+N_U}, i = \overline{1, N_L + N_U}$
2) **Для усіх** $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$ обчислити матрицю подібності:

$$\mathcal{S}(\mathrm{i},\mathrm{j}) = exp\left(-\frac{\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2}{\sigma^2}\right), \tag{2.24}$$

де σ – параметр розміру, $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$

3) **Для усіх** $\boldsymbol{x} \in U$ обчислити передбачену мітку, використовуючи правило k найближчих сусідів.
4) **Виконуємо для** $t = \overline{1, T}$:
    a) Виконуючи процес відбору, створюємо нову тренувальну вибірку S_t із доступних невідмічених даних U на додачу до L. У процесі відбору:

i. Обираємо частину сильних невідмічених даних з U (скажімо, 10%; U_{10}), відповідно до рівня впевненості, заснованого на матриці подібності $\mathcal{S}$

ii. Прибираємо обрані дані з множини невідмічених:

$$U \leftarrow U \backslash U_{10}; N_U \leftarrow |U|$$

б) $m = 10$ разів повторити наступні кроки:

i. Вивчити градієнтний напрямок h_t для S_t, максимізуючи $J_t^S = J_t^L + J_t^U$

ii. **Якщо** $J_t^S \leq 0$ **вийти, повернувши** $g_t(\boldsymbol{x})$

iii. $g_{t+1}(\boldsymbol{x}) = g_t(\boldsymbol{x}) + \alpha_t h_t(\boldsymbol{x})$

iv. **Оновити** w_i^{t+1} за формулою (2.20)

5) **Вихід:** $g_t(\boldsymbol{x})$. Клас = $sgn\big(g_t(\boldsymbol{x})\big)$

Під час експериментів, дані були поділені на три частини у пропорції 20 частин відмічених тренувальних даних, 10 частин відмічених тестувальних даних та 70 частин невідмічених даних. Процедури тренування та тестування повторено 10 разів і результати усереднені. Кількість слабких класифікаторів для обох SSMB у проміжку від 10 до 50 з кроком 5. $c(x) = exp(-x)$; $\alpha_t = \frac{1}{4} ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$; $\varepsilon_t = \sum_i w_i^t \, \mathbb{I}\big(y_i \neq g(\boldsymbol{x}_i)\big)$, $\mathbb{I}$ – індикаторна функція. У якості слабких класифікаторів для обох SSMB використовується дерево рішень.

За результатами експериментів було видно, що покращений SSMB мав помітно меншу помилку, порівняно із попередником та методами керованого навчання AdaBoost і MarginBoost.

Зазначено, що для даних з високою просторовою розмірністю алгоритм показав більшу перевагу при порівнянні з оригінальним SSMB, аніж для даних з малою розмірністю.

Також, існують роботи [8, 10, 11], що пропонують впровадження регуляризації – техніки, що допомагає уникати недонавчання, перенавчання та в цілому покращує якість роботи навченого алгоритму – його здатність узагальнюватися до нових даних, підвищуючи стійкість до шуму й викидів у тренувальних даних шляхом додавання штрафу до критерію оптимізації.

РОЗДІЛ 3. ПОБУДОВА МОДИФІКОВАНОГО АЛГОРИТМУ НАПІВКЕРОВАНОГО ГРАНИЧНОГО БУСТИНГУ

3.1. Аналіз та недоліки відомих підходів до напівкерованого граничного бустингу

Серед підходів, розглянутих вище, маємо оригінальний SSMBoost [2], покращений SSMBoost [9].

Основою алгоритмів є мінімізація функціоналу $C: Lin(\mathcal{H}) \to \mathbb{R}$ оцінки, шляхом градієнтного спуску у просторі лінійних комбінацій гіпотез базових класифікаторів $Lin(\mathcal{H})$. Це не є чистий градієнтний спуск, бо у якості напрямку h_{t+1} не береться $-\nabla C(g_t)$, адже тоді не гарантовано, що результуюча гіпотеза $g_{t+1} \in Lin(\mathcal{H})$. Натомість, нова гіпотеза h_{t+1} обирається таким чином, щоб максимізувати внутрішній добуток $-\langle \nabla C(g_t), h_{t+1} \rangle$, як пояснено формулою (2.15).

Важливо, також, обрати правильну довжину кроку градієнтного спуску. Від неї залежить, наскільки добре та як швидко буде мінімізуватися функціонал оцінки. Якщо довжина кроку обрана неправильно, алгоритм або не зійдеться, або розбіжиться, або буде повільно йти до оптимуму. В ідеалі, варто мати таку довжину кроку, що адаптується до ситуації. У випадку бустингу, довжина кроку адаптується до помилки базової гіпотези – чим вона менша, тим більший крок. Значення помилки залежить від ваги окремих прикладів на даній ітерації, тому її називають зваженою.

Будь-яка осмислена функція оцінки відступу $c: \mathbb{R} \to \mathbb{R}$ є монотонно спадною, отже $-c'\big(\rho_L(g_t(\boldsymbol{x}_i), y_i)\big)$ та $-c'\left(\rho_U\big(g_t(\boldsymbol{x}_i)\big)\right)$ будуть завжди додатними. Поділивши усі значення на $\left(-\sum_{i \in I_L} c'\big(\rho_L(g_t(\boldsymbol{x}_i), y_i)\big) - -\sum_{i \in I_U} c'\left(\rho_U\big(g_t(\boldsymbol{x}_i)\big)\right)\right)$ видно, що знаходження такого h_{t+1}, що максимізує

$-\langle \nabla C(g_t), h_{t+1} \rangle$ еквівалентне знаходженню такого h_{t+1}, що мінімізує зважену помилку $\varepsilon_t = \sum_{i:h_{t+1}(x_i) \neq y_i} w_i^t$, де w_i^t –нормалізована вага прикладу i,

$$w_i^t = \begin{cases} \dfrac{c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right)}{\sum_{i \in I_L} c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right) + \sum_{i \in I_U} c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)} & \text{якщо } i \in L \\ \dfrac{c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)}{\sum_{i \in I_L} c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right) + \sum_{i \in I_U} c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)} & \text{якщо } i \in U \end{cases}$$

Вага отриманої гіпотези базового класифікатора, що є її коефіцієнтом у вихідній лінійній комбінації, і, відповідно, довжиною кроку в алгоритму градієнтного спуску, обчислюється наступним чином:

$$\alpha_t = \frac{1}{4} ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right),$$

де ε_t – зважена помилка базової гіпотези.

Недоліком такого значення довжини кроку є те, що коефіцієнт при логарифмі сталий. Таким чином, крок може виявитися завеликим та/або недостатньо адаптивним.

У другому алгоритмі SSMBoost, що є покращенням оригінального, представленим у [9], пропонується використовувати лише частину невідмічених даних, оскільки не усі з них можуть бути корисними для навчання.

На початку роботи алгоритму, невідміченим даним присвоюється псевдомітка, використовуючи правило k найближчих сусідів, за яким нова точка відноситься до того класу, до якого належить більшість її k найближчих сусідів. Так, якщо k=1, точка відноситься до найближчого класу, якщо k=N – до класу з найбільшою кількістю прикладів. Тому, значення k є одним з важливих гіперпараметрів покращеного алгоритму SSMBoost.

Разом із псевдомітками, обчислюється рівень впевненості у їх значеннях. Кожний десятий крок до множини відмічених прикладів відносять частину (10%) невідмічених даних із найбільш впевненими псевдомітками. Таким чином, у навчанні класифікатора беруть участь лише «сильні» невідмічені дані.

Загалом, представлені алгоритми є агностичними стосовно вибору виду базових класифікаторів, але їх вибір безпосередньо впливає на загальну якість роботи алгоритму.

У оригінальній роботі по напівкерованому граничному бустингу [2] пропонується використовувати моделі сумішей, що пристосовані до роботи із невідміченими даними. Але, моделі сумішей – це алгоритм некерованого навчання, алгоритм кластеризації, тобто при навчанні він не бере до уваги значення міток класів. Розподіл кластерів на класи робиться вже після навчання слабкого учня.

У роботі [9] в якості слабких учнів використовуються дерева рішень. Їхня глибина не зазначається, але, для уникнення перенавчання, вона має бути невеликою. Зазвичай використовуються так звані «ростки рішень» - дерева рішень з глибиною 1.

Недоліком такого вибору є те, що ростки рішень беруть до уваги лише одну ознаку одночасно, а дерева рішень в цілому – будують межі рішень у вигляді комбінації меж, перпендикулярних до осей ознак, що може бути недостатньо гнучким варіантом для деяких задач і потребувати додаткової інженерії ознак.

Після навчання, алгоритм може мати накопичену помилку і кінцевий стан може виявитися гірше, відповідно до значення функціоналу оцінки, аніж деякі з попередніх.

3.2. Модифікований алгоритм напівкерованого граничного бустингу

Новий алгоритм має вирішувати проблеми обмеженості простих класифікаторів та нестабільності довжини кроку класифікатора.

Пропоную наступні покращення:

1. Для уникнення обмеженості простих класифікаторів, одним з можливих варіантів розв’язання є – додати можливість обирати найкращий простий класифікатор із доступних на даній ітерації, використовуючи деяку палітру простих класифікаторів.

На кожній ітерації у навчанні бере участь декілька видів класифікаторів, наприклад – росток рішень, лінійна опорно-векторна машина та перцептрон із одним нейроном. Для кожного з кандидатів обчислюється зважена помилка і обирається той класифікатор, чия помилка виявилася найменшою. Таким чином, з’являється можливість побудувати межу прийняття рішень більш складної форми аніж комбінація меж, перпендикулярних до осей ознак без потреби у додатковій інженерії однак (тим не менш, вона все ще може мати місце).

2. Для подолання можливої нестабільності у довжині кроку пропоную реалізувати алгоритм його ітеративного зменшення у випадку коли нове значення функціоналу оцінки виявляється більше за попереднє.

У випадку погіршення значення оцінки пропонується зменшити довжину кроку α_t у 2 рази (конфігурацію значення можливо винести як гіперпараметр), окрім тих кроків, на яких до відмічених даних додаються сильні невідмічені дані, оскільки на спостерігається загальне погіршення значення оцінки незалежно від довжини кроку.

Оскільки в такій конфігурації алгоритм може застрягти в деякому локальному мінімумі, можливим є варіант пропускати деякі ітерації зменшення

кроку з певною імовірністю. Також, оскільки градієнтний спуск сильно залежить від початкової точки, має сенс виконувати кілька запусків алгоритму з різних початкових точок.

3. Оскільки кінцеве значення функціоналу оцінки може бути гірше ніж на деяких із попередніх ітерацій, має сенс знайти ітерацію, на якій оцінка була мінімальна та відкинути з переліку базових гіпотез усі наступні.

Після застосування запропонованих покращень, отримуємо наступний алгоритм:

Алгоритм 5. SSMBoost із застосуванням запропонованих покращень.

1) **Вхід:**

$\mathcal{X}_L$ – множина точок даних, що відповідають відміченим прикладам;

$\mathcal{X}_U$ – множина точок даних, що відповідають невідміченим прикладам;

$\mathcal{Y}_L$– множина міток відмічених прикладів;

$S = L \cup U$ – тренувальна вибірка, де $L = \{(\boldsymbol{x}, y)|\boldsymbol{x} \in \mathcal{X}_L, y \in \mathcal{Y}_L\}$ – множина відмічених даних, $U = \{(\boldsymbol{x})|\boldsymbol{x} \in \mathcal{X}_U\}$ – множина невідмічених даних;

k – кількість сусідніх прикладів для алгоритму «k найближчих сусідів», наприклад 5;

T – кількість ітерацій, вона ж – максимальна кількість слабких учнів;

β – початкове значення коефіцієнту ваги класифікатора, наприклад 0.25;

$p \in [0; 1]$ – частка невідмічених даних, що додається до набору відмічених, наприклад 0.1;

n – кожний n-й крок до набору відмічених даних S_L додається частка найбільш впевнених невідмічених даних з набору S_U, наприклад 10;

m – максимальна кількість ітерацій-спроб покращення оцінки, наприклад 15;

$\theta > 1$ – коефіцієнт зменшення довжини кроку, наприклад 2;

П – Палітра базових класифікаторів, наприклад: росток рішень, лінійна опорно-векторна машина та перцептрон з одним нейроном;

2) **Ініціалізація:**

$w_i^0 := \frac{1}{|L|+|U|}$ – початкові ваги кожного з тренувальних прикладів;

$\mathcal{Y}_U$ – множина псевдоміток невідмічених даних, відповідно до правила k найближчих сусідів

P – впевненості у правильності міток, відповідно до правила k найближчих сусідів;

$C(g_0) := +\infty$ – початкове значення функціоналу оцінки;

3) **Повторюємо для** $t = \overline{1,T}$:

а) Якщо t кратне n: обираємо з набору S_U частку p найбільш упевнених невідмічених даних та їхні псевдомітки з $\mathcal{Y}_U$ відповідно до рівнів впевненості P, додаємо їх до множини L та видаляємо з U, $\mathcal{Y}_U$ та P відповідно;

б) Навчаємо слабкі класифікатори-кандидати з палітри класифікаторів П: $\left\{h_j^t \middle| j = \overline{1,|\Pi|}\right\}$ на відміченій частині вибірки;

в) Обчислюємо для кожного отриманого слабкого класифікатора зважену помилку та обираємо найкращу гіпотезу, яка буде напрямком кроку градієнтного спуску:

$$h_t(\boldsymbol{x}) := arg\min_{h_j^t}\left\{\varepsilon_j^t = \sum_{i\in I_L} w_i^t \mathbb{I}\left(h_j^t(\boldsymbol{x_i}) \neq y_i\right)\right\}, \qquad (3.1)$$

де $\mathbb{I}(\cdot)$ – індикаторна функція, I_L – множина індексів, що відповідають відміченим прикладам;

г) Обчислюємо вагу гіпотези – довжину кроку градієнтного спуску:

$$\alpha_t \coloneqq \beta\, ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right), \tag{3.2}$$

де ε_t – помилка базової гіпотези;

д) Обчислюємо значення характеристик $J_S^t = J_L^t + J_U^t$:

$$J_L^t = \sum_{i \in I_L} w_i^t y_i h_t(\boldsymbol{x}_i), \tag{3.3}$$

$$J_U^t = \sum_{i \in I_U} w_i^t \frac{\partial \rho_U\big(g_t(\boldsymbol{x}_i)\big)}{\partial g_t(\boldsymbol{x}_i)} h_t(\boldsymbol{x}_i), \tag{3.4}$$

де $\rho_U\big(g_t(\boldsymbol{x})\big) = g_t(\boldsymbol{x})^2$, I_L – множина індексів, що відповідають відміченим прикладам, I_U – множина індексів, що відповідають невідміченим прикладам,

$$g_t(\boldsymbol{x}) = \sum_{r=1}^{t} \alpha_r h_r(\boldsymbol{x}), \tag{3.5}$$

е) **Якщо** $J_S^t \leq 0$, то $T \coloneqq t-1$, виходимо з циклу;

ж) Обчислюємо значення функціоналу оцінки:

$$\mathrm{C}(g_t) = \sum_{i \in I_L} c\big(\rho_L(g_t(\boldsymbol{x}_i), y_i)\big) + \sum_{\mathrm{i} \in \mathrm{I_U}} c\left(\rho_U\big(g_t(\boldsymbol{x}_i)\big)\right), \tag{3.6}$$

де $c(x) = exp(-x)$, $\rho_L(x,y) = xy$, $\rho_L(x) = x^2$;

з) **Повторюємо для** $j = \overline{1,m}$:

i. **Якщо** $\mathrm{C}(g_t) < \mathrm{C}(g_{t-1})$ або t кратне n, то виходимо з циклу (з), де n – максимальна кількість ітерацій покращення оцінки;

ii. Зменшуємо поточну довжину кроку в θ разів:

$$\alpha_t := \frac{\alpha_t}{\theta}, \tag{3.7}$$

и) Оновлюємо ваги прикладів:

$$w_i^t = \begin{cases} \dfrac{c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right)}{\sum_{i\in I_L} c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right) + \sum_{i\in I_U} c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)} & \text{якщо } i \in L \\ \dfrac{c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)}{\sum_{i\in I_L} c'\left(\rho_L(g_t(\boldsymbol{x}_i), y_i)\right) + \sum_{i\in I_U} c'\left(\rho_U\left(g_t(\boldsymbol{x}_i)\right)\right)} & \text{якщо } i \in U \end{cases}, \tag{3.8}$$

де $c'(x) = -exp(-x)$, I_L – множина індексів, що відповідають відміченим прикладам, I_U – множина індексів, що відповідають невідміченим прикладам;

4) $T := arg\,\min_T\left(C(g_T)\right)$;

5) **Вихід:** $g_T(\boldsymbol{x}) = \sum_{\boldsymbol{t=1}}^{\boldsymbol{T}} \boldsymbol{\alpha_t h_t}(\boldsymbol{x})$. Для отримання результату класифікації, обчислюється $sgn\left(g_T(\boldsymbol{x})\right)$.

3.3. Результати роботи розробленого алгоритму на навчальній та валідаційній вибірці

3.3.1. Підготовка навчальної та валідаційної вибірок

Для дослідження роботи алгоритму використовуватимемо штучно згенеровані вибірки даних. На якість розділення даних впливають їх просторове взаємне розташування (пересікання, лінійна роздільність) і кількість невідмічених даних.

Було підготовлено кілька видів наборів даних за просторовим розташуванням. Кожний вид просторового розміщення представлений у таких співвідношеннях розмічених даних до нерозмічених: 100%, 50%, 10% й 1%. Вибірки представлені у розмірах 1000 та 10 000 елементів. До валідаційної вибірки потрапляє 10% від загальної кількості даних, взяті з нерозміченої частини даних. Таким чином, до тренувальної вибірки потрапляє 90% усіх даних.

Види просторового розташування:

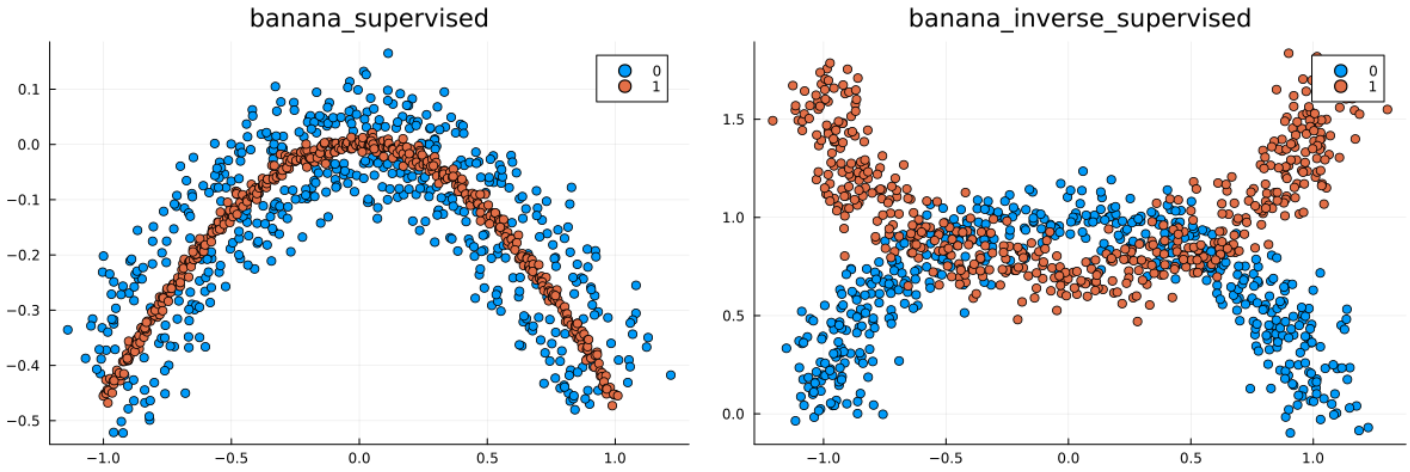

Рисунок 3.1. «Банан» та «Зворотній банан»

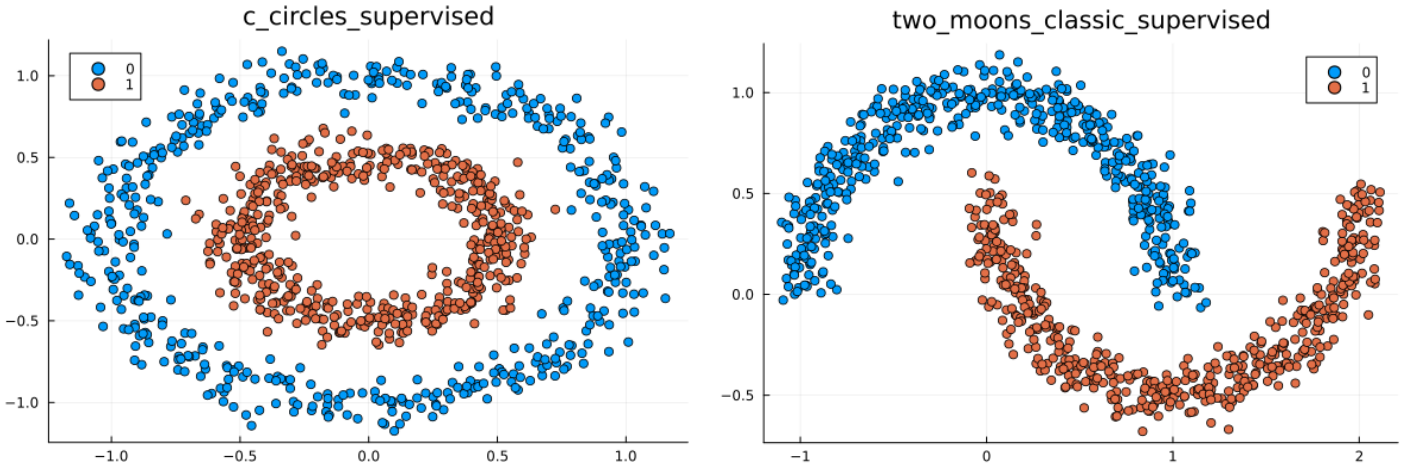

Рисунок 3.2. «Концентричні кола» та «Два місяці» класичні

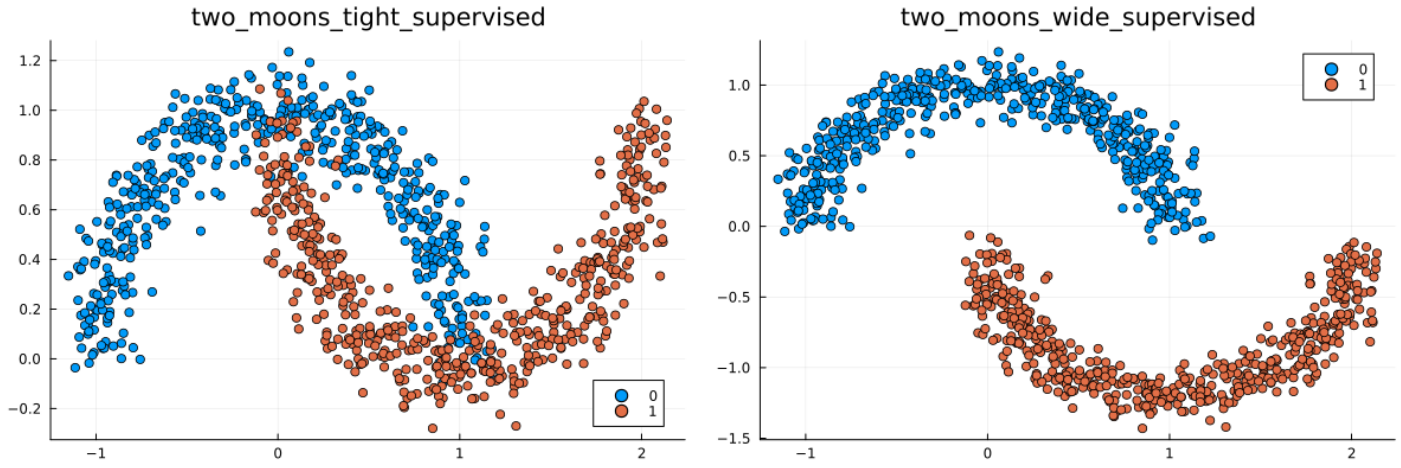

Рисунок 3.3. «Два місяці» щільні та «Два місяці» нещільні

Для більш ефективної роботи алгоритму дані варто попередньо обробити. Так, у даних, що відмічені 0, мітка заміняється на (-1). Також, для більш ефективної роботи деяких базових класифікаторів з палітри, дані потрібно попередньо нормувати. Робиться це за наступною формулою:

$$\hat{x}_{i,j} = \frac{x_{i,j} - \|\boldsymbol{x}_j\|_1}{\sigma_j}, \tag{3.9}$$

де $x_{i,j}$ – значення i-ї точки за j-ю ознакою, $\|\boldsymbol{x}\|_1 = \frac{1}{N}\sum_{i=1}^{N} x_i$, σ_j – стандартне відхилення j-ї ознаки.

Для деяких наборів даних може додатково знадобитися інженерія ознак. Наприклад, можливо додати кілька ознак виду $ax_1 + bx_2$, $a, b \in \mathbb{R}$.

3.3.2. Результати роботи алгоритму

Перевірка роботи алгоритму відбувалася при кількості базових класифікаторів 50, кількості найближчих сусідів 5, початковому значенні коефіцієнту при довжині кроку спуску 0.75 та максимальній кількості спроб покращити оцінку 15.

Таблиця 3.1. Результати на навчальній вибірці, 1000 прикладів, максимум 50 базових класифікаторів

	accuracy	precision (pos)	precision (neg)	recall (pos)	recall (neg)	f1	roc auc
50% banana	0,6389	0,5819	0,8667	0,9458	0,3414	0,7206	0,6436
50% banana_inverse	0,7678	0,7733	0,7622	0,7648	0,7708	0,7691	0,7678
50% c_circles	0,9978	0,9977	0,9978	0,9977	0,9978	0,9977	0,9978
50% two_moons_classic	0,9989	1,0000	0,9978	0,9978	1,0000	0,9989	0,9989
50% two_moons_tight	0,9600	0,9707	0,9496	0,9493	0,9709	0,9599	0,9601
50% two_moons_wide	0,9956	0,9978	0,9934	0,9933	0,9978	0,9955	0,9956
10% banana	0,4967	0,4967	0,0000	1,0000	0,0000	0,6637	0,5000
10% banana_inverse	0,7133	0,9381	0,6449	0,4457	0,9716	0,6043	0,7087
10% c_circles	0,9722	1,0000	0,9469	0,9449	1,0000	0,9717	0,9725
10% two_moons_classic	0,9811	0,9844	0,9778	0,9779	0,9843	0,9812	0,9811
10% two_moons_tight	0,9256	0,8947	0,9631	0,9672	0,8826	0,9295	0,9249
10% two_moons_wide	0,9956	0,9909	1,0000	1,0000	0,9914	0,9954	0,9957
1% banana	0,5444	0,5234	0,6490	0,8809	0,2154	0,6566	0,5481
1% banana_inverse	0,6267	0,6381	0,6171	0,5813	0,6718	0,6084	0,6266
1% c_circles	0,4722	0,4218	0,4821	0,1372	0,8103	0,2070	0,4737
1% two_moons_classic	0,8378	0,9410	0,7754	0,7169	0,9560	0,8138	0,8364
1% two_moons_tight	0,6722	0,8989	0,6124	0,3798	0,9582	0,5340	0,6690
1% two_moons_wide	0,9789	1,0000	0,9589	0,9584	1,0000	0,9788	0,9792

Таблиця 3.2. Результати на валідаційній вибірці, 1000 прикладів, максимум 50 базових класифікаторів

	accuracy	precision (pos)	precision (neg)	recall (pos)	recall (neg)	f1	roc auc
50% banana	0,6300	0,6250	0,6500	0,8772	0,3023	0,7299	0,5898
50% banana_inverse	0,7600	0,7143	0,8039	0,7778	0,7455	0,7447	0,7616
50% c_circles	0,9900	1,0000	0,9773	0,9825	1,0000	0,9912	0,9912
50% two_moons_classic	0,9900	1,0000	0,9821	0,9778	1,0000	0,9888	0,9889
50% two_moons_tight	0,9600	0,9773	0,9464	0,9348	0,9815	0,9556	0,9581
50% two_moons_wide	0,9800	0,9804	0,9796	0,9804	0,9796	0,9804	0,9800
10% banana	0,5300	0,5300	0,0000	1,0000	0,0000	0,6928	0,5000
10% banana_inverse	0,6300	0,9565	0,5325	0,3793	0,9762	0,5432	0,6778
10% c_circles	0,9500	1,0000	0,9153	0,8913	1,0000	0,9425	0,9457
10% two_moons_classic	0,9700	0,9783	0,9630	0,9574	0,9811	0,9677	0,9693
10% two_moons_tight	0,8600	0,7736	0,9574	0,9535	0,7895	0,8542	0,8715
10% two_moons_wide	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
1% banana	0,5900	0,5795	0,6667	0,9273	0,1778	0,7133	0,5525
1% banana_inverse	0,6900	0,7083	0,6731	0,6667	0,7143	0,6869	0,6905
1% c_circles	0,5100	0,4737	0,5185	0,1875	0,8077	0,2687	0,4976
1% two_moons_classic	0,8300	0,9750	0,7333	0,7091	0,9778	0,8211	0,8434
1% two_moons_tight	0,6300	0,9091	0,5513	0,3636	0,9556	0,5195	0,6596
1% two_moons_wide	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

Як видно з результатів, вибірки виду «банан» є доволі складними для правильної класифікації через те, що точки класів розташовані доволі щільно між собою у складній формі. Також, на вибірках з 1% відмічених даних результати помітно гірше, оскільки доволі складно описати форму даних через 10 відмічених точок. Покращення у цьому плані наявні при навчанні на вибірці з 10 000 прикладів, оскільки 1% від 10 000 це 100, а така кількість точок може відносно добре описати форму простору. Це можна спостерігати й на 10% вибірки з 1000 прикладів.

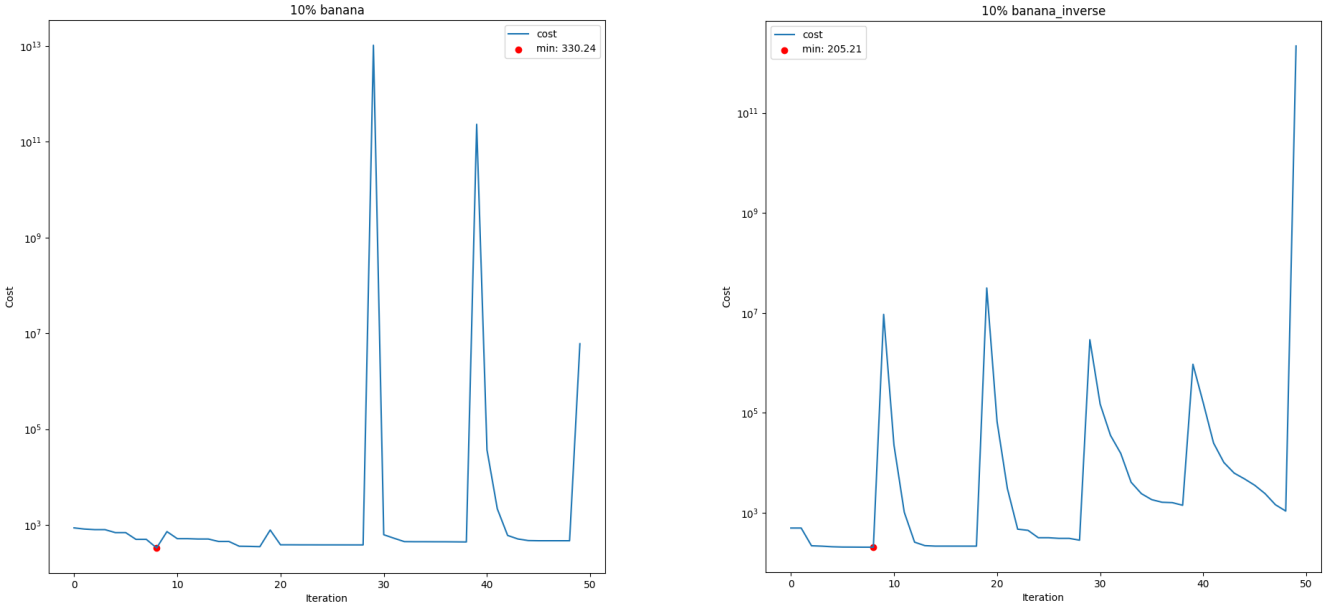

Рисунок 3.4. Графіки значення оцінки при навчанні на вибірках «банан» та «зворотній банан» відповідно. 10% відмічених даних

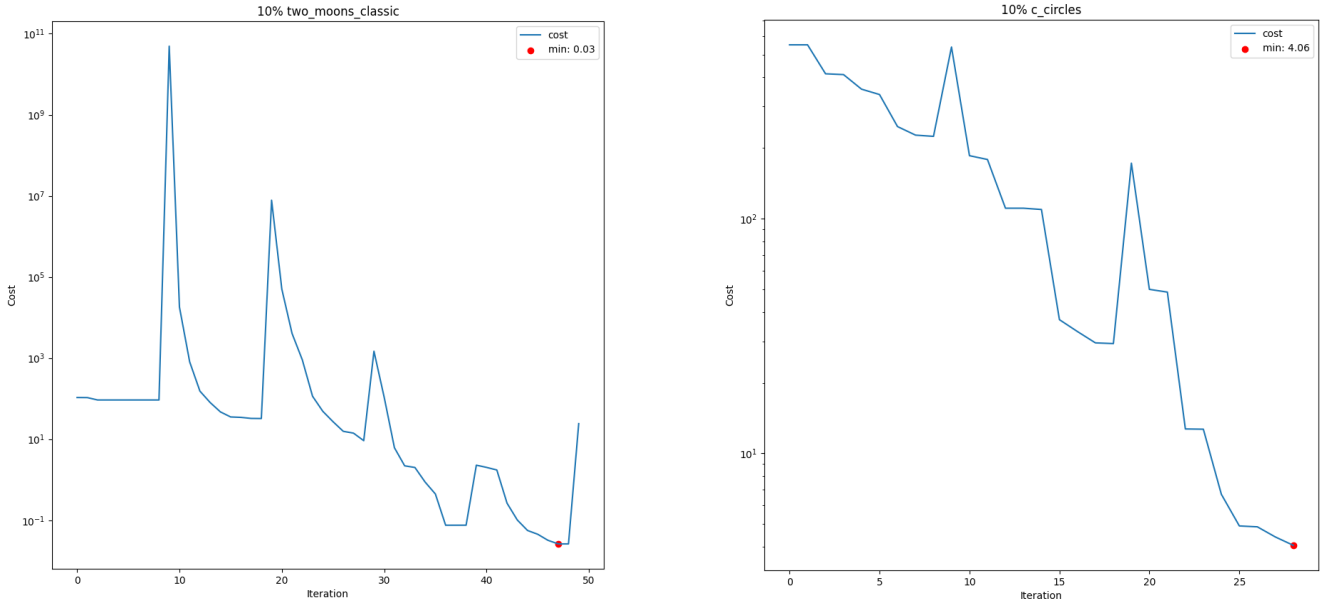

Рисунок 3.5. Графіки значення оцінки при навчанні на вибірках «два місяці» класичні та «концентричні кола» відповідно. 10% відмічених даних

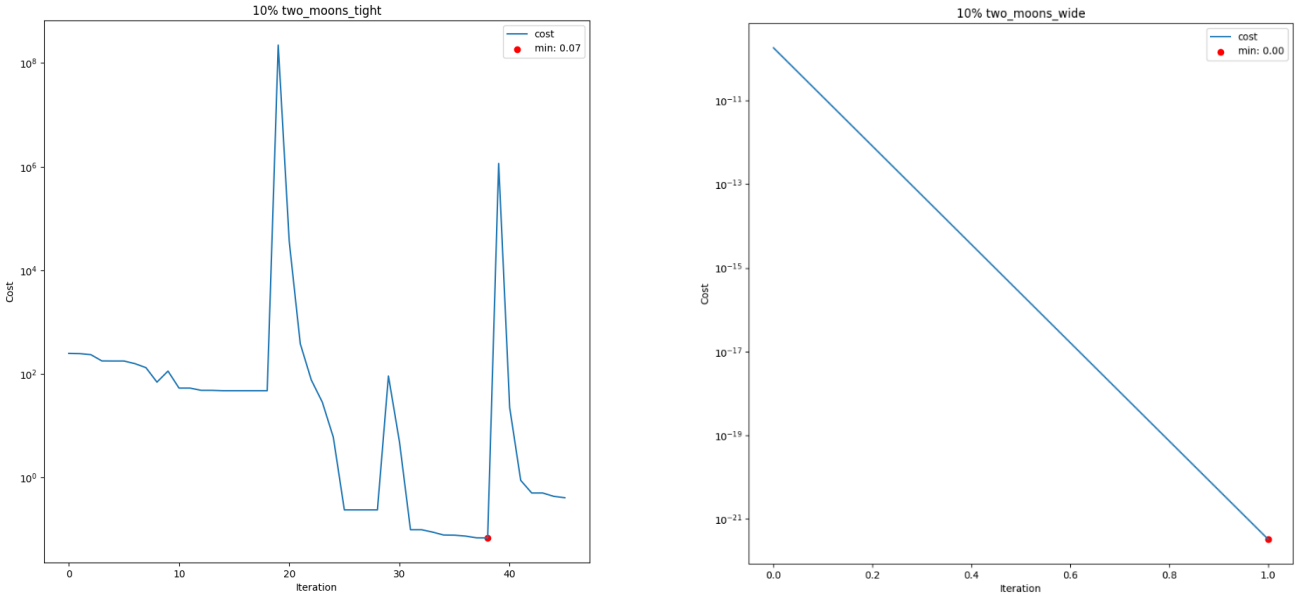

Рисунок 3.6. Графіки значення оцінки при навчанні на вибірках «два місяці» щільні та нещільні відповідно. 10% відмічених даних

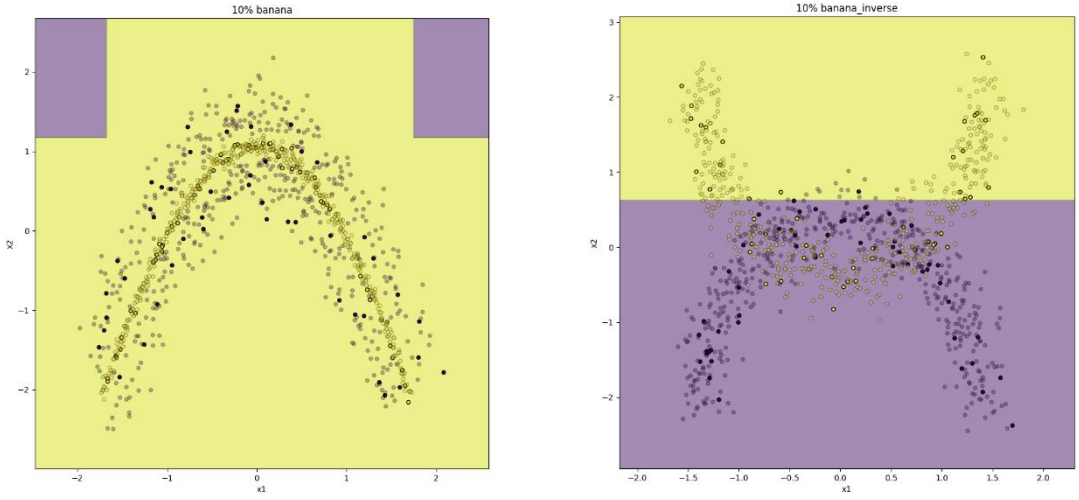

Рисунок 3.7. Межі рішень для «банан» та «зворотній банан». 10% відмічених даних

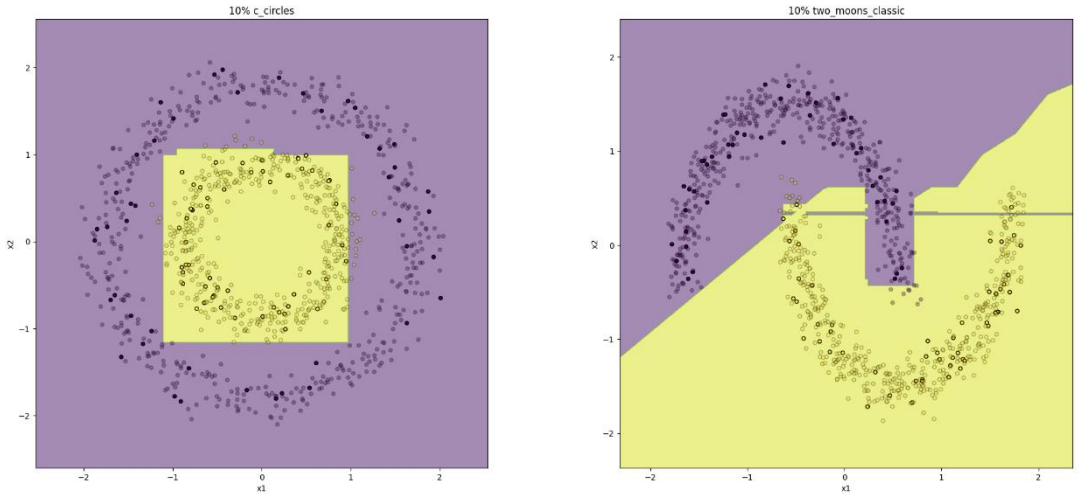

Рисунок. 3.8. Межі рішень для «концентричні кола» та «два місяці» класичні. 10% відмічених даних

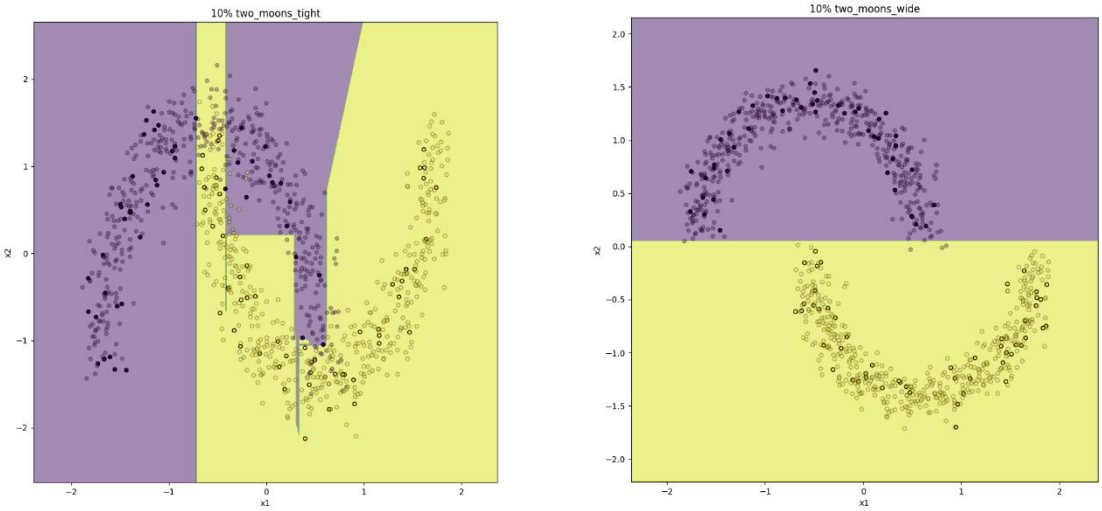

Рисунок 3.9. Межі рішень для «два місяці» щільні та нещільні. 10% відмічених даних

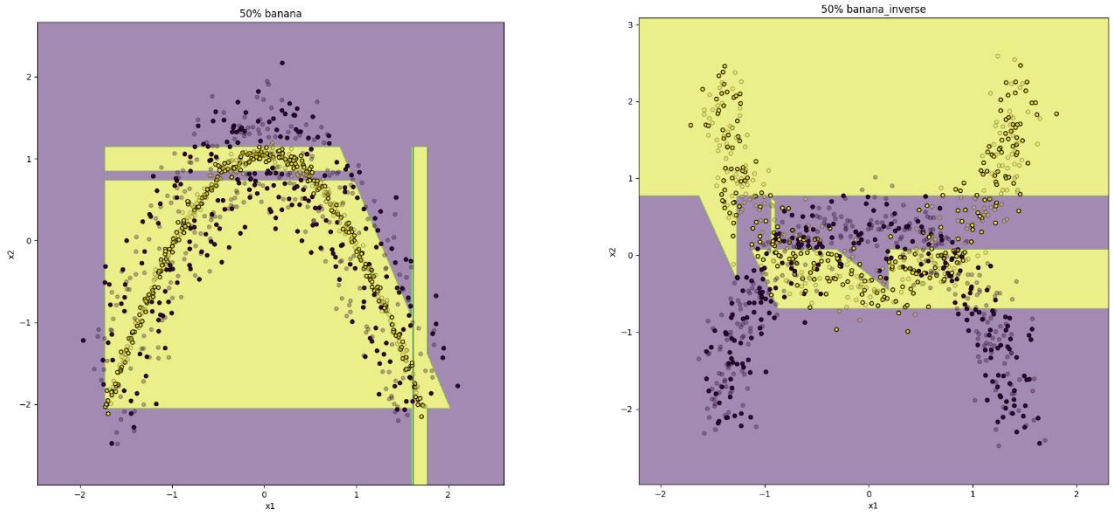

Рисунок 3.10. Межі рішень для «банан» та «зворотній банан». 50% відмічених даних

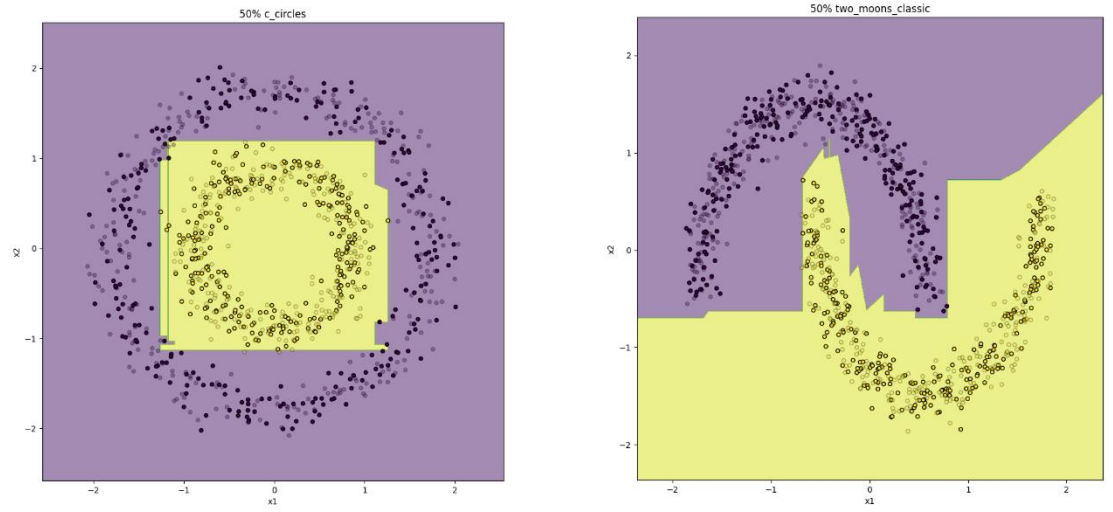

Рисунок 3.11. Межі рішень для «концентричні кола» та «два місяці» класичні. 50% відмічених даних

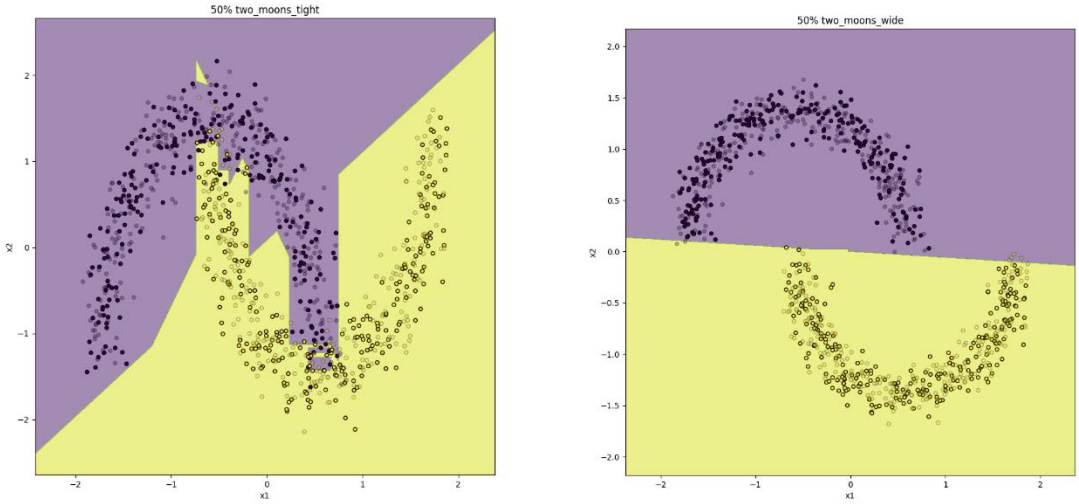

Рисунок 3.12. Межі рішень для «два місяці» щільні та нещільні. 50% відмічених даних

Таблиця 3.3. Результати на навчальній вибірці, 10 000 прикладів, максимум 50 базових класифікаторів

	accuracy	precision (pos)	precision (neg)	recall (pos)	recall (neg)	f1	roc auc
50% banana	0,6391	0,5811	0,9893	0,9970	0,2811	0,7343	0,6390
50% banana_inverse	0,7267	0,7048	0,7537	0,7797	0,6737	0,7404	0,7267
50% c_circles	0,9953	0,9952	0,9953	0,9954	0,9951	0,9953	0,9953
50% two_moons_classic	0,9940	0,9898	0,9984	0,9984	0,9897	0,9941	0,9940
50% two_moons_tight	0,9197	0,8924	0,9510	0,9543	0,8852	0,9223	0,9197
50% two_moons_wide	0,9979	0,9992	0,9966	0,9966	0,9992	0,9979	0,9979
10% banana	0,5730	0,5395	1,0000	1,0000	0,1457	0,7009	0,5729
10% banana_inverse	0,6540	0,5913	0,9967	0,9990	0,3087	0,7429	0,6538
10% c_circles	0,9898	0,9866	0,9931	0,9931	0,9865	0,9898	0,9898
10% two_moons_classic	0,9963	0,9982	0,9944	0,9943	0,9982	0,9963	0,9963
10% two_moons_tight	0,7805	0,7709	0,7908	0,7972	0,7638	0,7838	0,7805
10% two_moons_wide	0,9993	0,9998	0,9988	0,9988	0,9998	0,9993	0,9993
1% banana	0,5775	0,5413	1,0000	1,0000	0,1572	0,7024	0,5786
1% banana_inverse	0,6923	0,9250	0,6242	0,4187	0,9661	0,5765	0,6924
1% c_circles	0,5006	0,0000	0,5006	0,0000	1,0000	0,0000	0,5000
1% two_moons_classic	0,9984	0,9980	0,9988	0,9988	0,9980	0,9984	0,9984
1% two_moons_tight	0,7721	0,8073	0,7442	0,7150	0,8293	0,7584	0,7721
1% two_moons_wide	0,9938	0,9910	0,9968	0,9968	0,9909	0,9939	0,9938

Таблиця 3.4. Результати на валідаційній вибірці, 10 000 прикладів, максимум 50 базових класифікаторів

	accuracy	precision (pos)	precision (neg)	recall (pos)	recall (neg)	f1	roc auc
50% banana	0,6218	0,5762	0,8242	0,9357	0,3047	0,7132	0,6202
50% banana_inverse	0,8186	0,9067	0,7613	0,7118	0,9262	0,7975	0,8190
50% c_circles	0,9962	0,9945	0,9980	0,9980	0,9944	0,9962	0,9962
50% two_moons_classic	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
50% two_moons_tight	0,9451	0,9232	0,9694	0,9708	0,9195	0,9464	0,9452
50% two_moons_wide	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
10% banana	0,5748	0,5409	1,0000	1,0000	0,1479	0,7021	0,5739
10% banana_inverse	0,6553	0,5927	0,9971	0,9991	0,3097	0,7440	0,6544
10% c_circles	0,9914	0,9888	0,9942	0,9942	0,9886	0,9915	0,9914
10% two_moons_classic	0,9982	0,9978	0,9987	0,9987	0,9978	0,9982	0,9982
10% two_moons_tight	0,7793	0,7725	0,7867	0,7957	0,7628	0,7839	0,7792
10% two_moons_wide	0,9993	0,9996	0,9991	0,9991	0,9996	0,9993	0,9993
1% banana	0,5786	0,5429	1,0000	1,0000	0,1562	0,7037	0,5781
1% banana_inverse	0,6939	0,9250	0,6262	0,4202	0,9661	0,5779	0,6932
1% c_circles	0,5016	0,0000	0,5016	0,0000	1,0000	0,0000	0,5000
1% two_moons_classic	0,9924	0,9982	0,9868	0,9867	0,9982	0,9924	0,9924
1% two_moons_tight	0,9174	0,9147	0,9202	0,9206	0,9143	0,9176	0,9174
1% two_moons_wide	0,9942	0,9918	0,9966	0,9967	0,9918	0,9942	0,9942

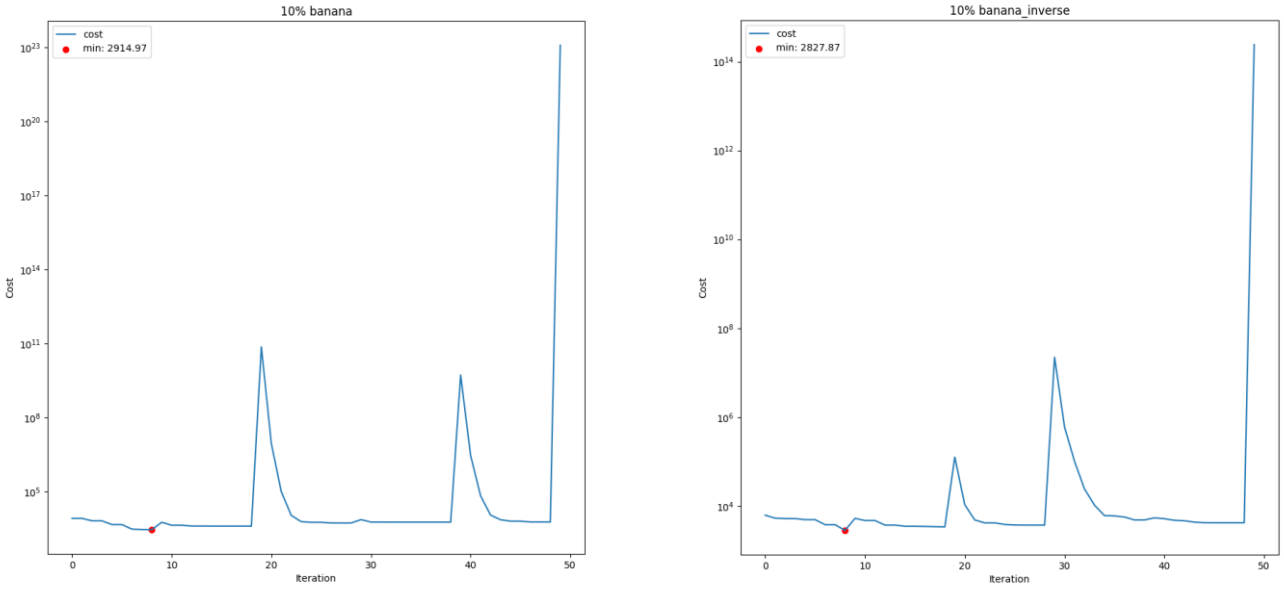

Рисунок 3.13. Графіки значення оцінки при навчанні на вибірках «банан» та «зворотній банан» відповідно. 10% відмічених даних

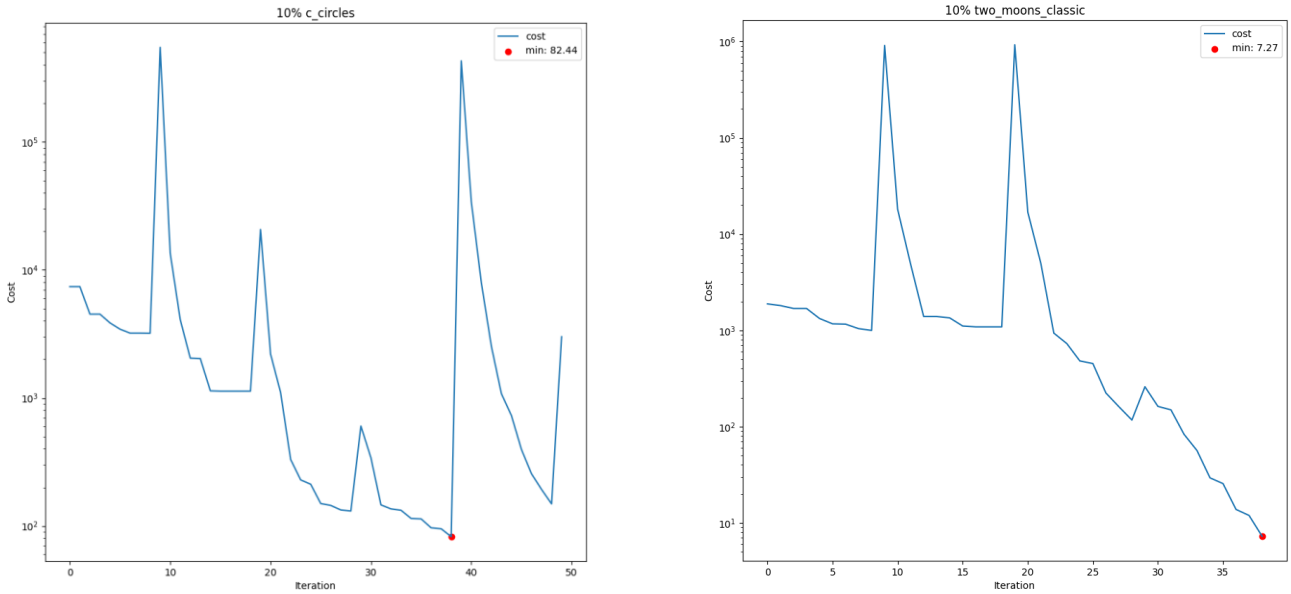

Рисунок 3.14. Графіки значення оцінки при навчанні на вибірках «два місяці» класичні та «концентричні кола» відповідно. 10% відмічених даних

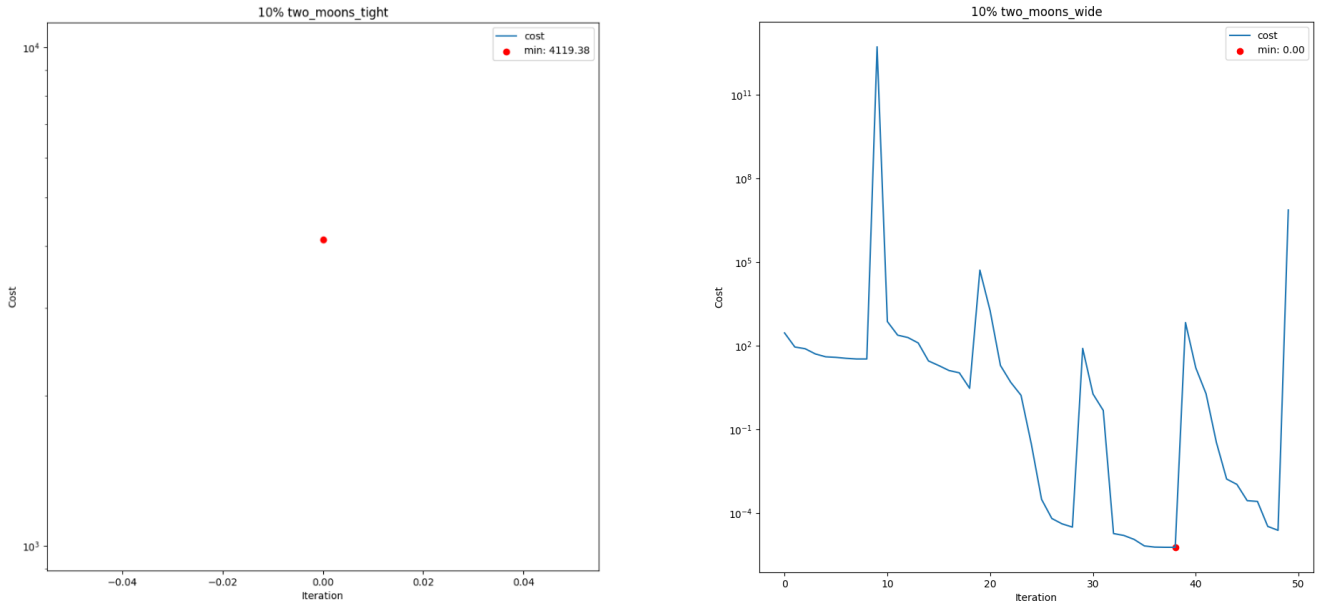

Рисунок 3.15. Графіки значення оцінки при навчанні на вибірках «два місяці» щільні та нещільні відповідно. 10% відмічених даних

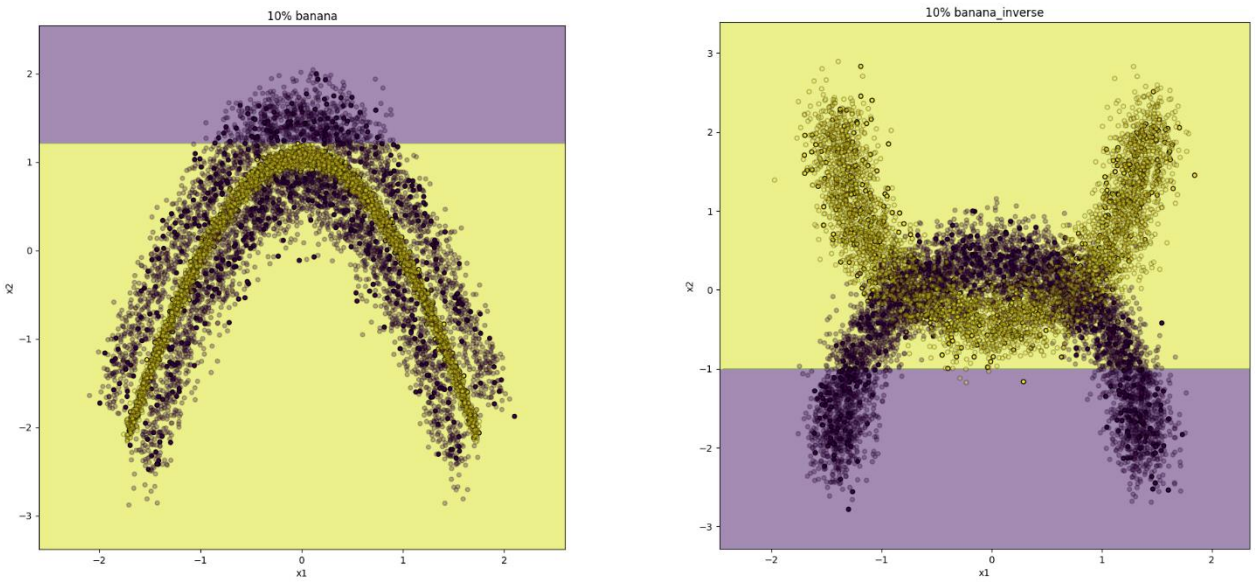

Рисунок 3.16. Межі рішень для «банан» та «зворотній банан». 10% відмічених даних

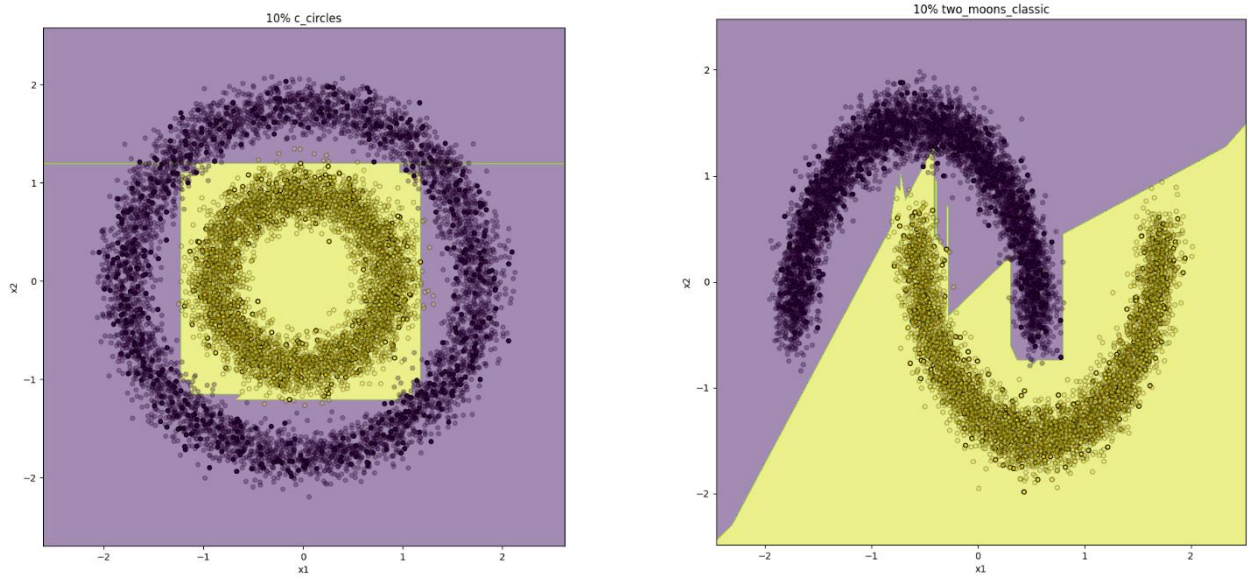

Рисунок. 3.17. Межі рішень для «концентричні кола» та «два місяці» класичні. 10% відмічених даних

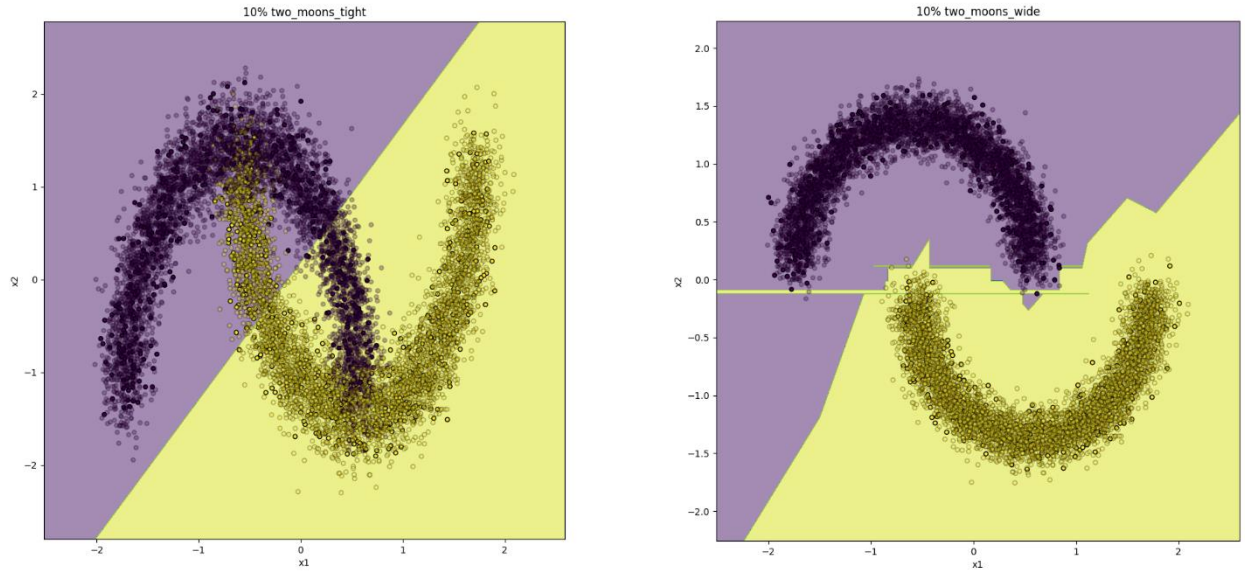

Рисунок 3.18. Межі рішень для «два місяці» щільні та нещільні. 10% відмічених даних

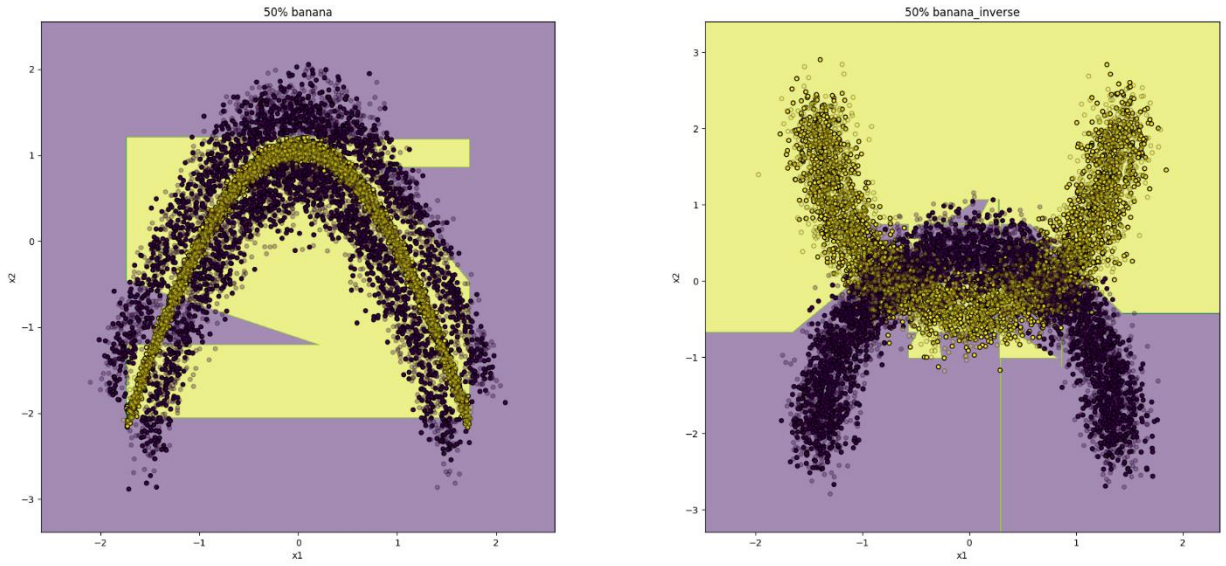

Рисунок 3.19. Межі рішень для «банан» та «зворотній банан». 50% відмічених даних

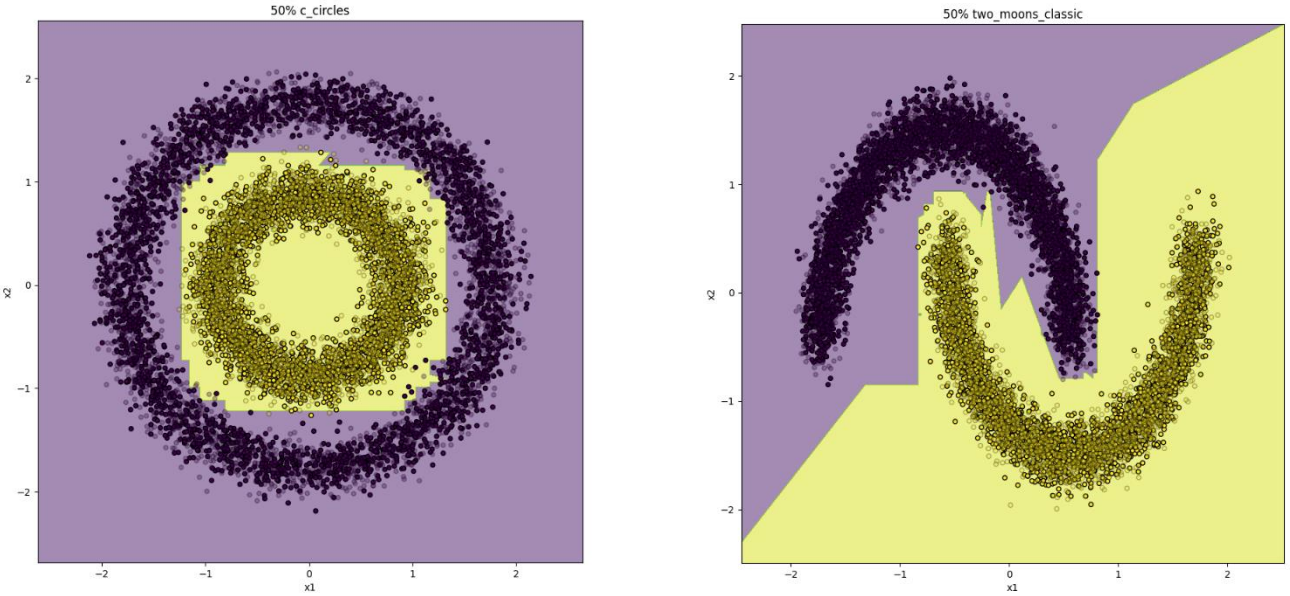

Рисунок 3.20. Межі рішень для «концентричні кола» та «два місяці» класичні. 50% відмічених даних

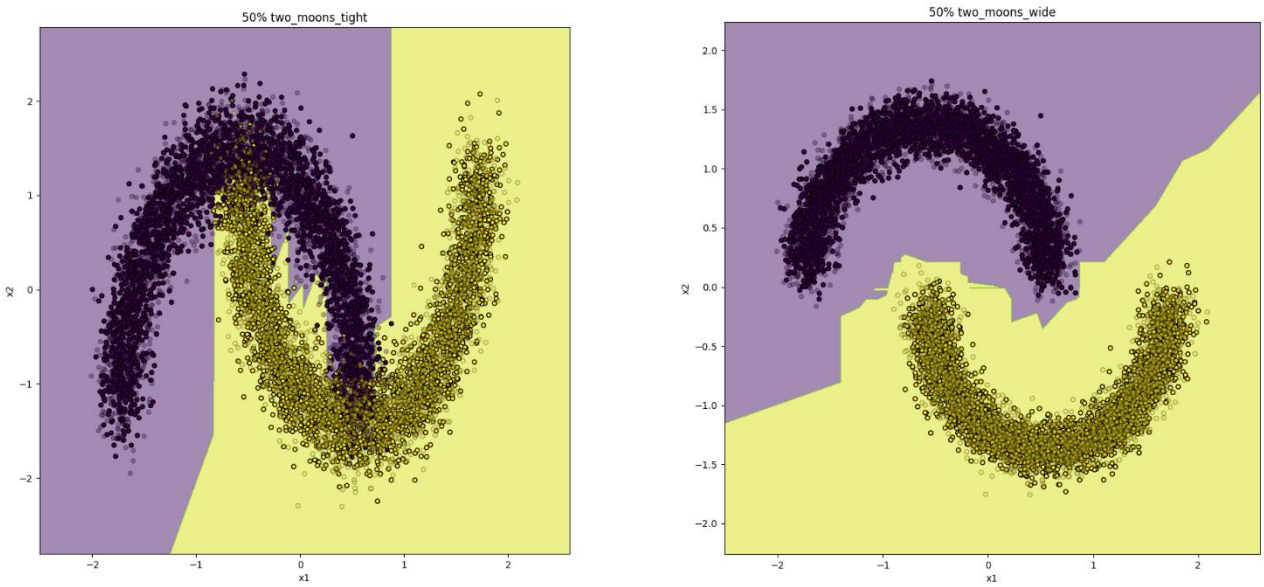

Рисунок 3.21. Межі рішень для «два місяці» щільні та нещільні. 50% відмічених даних

РОЗДІЛ 4. ВИКОРИСТАННЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ В МЕДИЧНІЙ ДІАГНОСТИЦІ

4.1. Довідка про хворобу

Інфекційний ендокардит – це захворювання, що розвивається внаслідок інфекції ендокарда клапанів серці (найчастіше), шлуночків та передсердь, або ендотелію великих кровоносних судин грудної клітки (наприклад, звуженого перешийка аорти), судинних анастомозів або чужорідних тіл у серці (наприклад, електродів кардіостимулятора). Найчастіше ІЕ пошкоджує аортальний та мітральний клапани, рідше тристулковий, а в приблизно 10% випадків – більше 1 клапана. Інфекційному ендокардиту передує бактеріємія – від < 2-х тижнів (у 80% випадків) до 2-5 місяців (у деяких хворих з ІЕ штучного клапана).

4.2. Побудова інтелектуальної діагностичної системи визначення серцевої недостатності

4.2.1. Підготовка набору даних

Робоча вибірка складається зі 112 прикладів та 346 ознак. Після аналізу виявилося, що ознаками, які несуть певну інформацію є лише 285 з них. Один з пацієнтів майже не мав записів, тому кількість корисних прикладів скоротилася до 111.

Частина ознак у вибірці представляє неперервні дані (наприклад, вага, вік, тривалість реанімаційного періоду), решта ознак – категоріальні. Для кращої

роботи алгоритму, числові дані мають бути нормалізовані. Виконуємо нормування ознак за формулою (3.9).

Оскільки для визначення простору з кількістю вимірів N необхідно мати хоча б $(N+1)$ точку даних. До того ж, деякі ознаки можуть виявитися непринциповими для класифікації.

Отже, для зменшення розмірності простору ознак, скористаємося методом головних компонент (англ. Principal Component Analysis, PCA). Використовуючи цю техніку, дані проектуються на простір меншої розмірності, втрачаючи при цьому найменшу кількість корисної інформації.

Робочу вибірку розділяємо на навчальну та валідаційну у співвідношенні 85:15. Штучно ділимо навчальну вибірку на розмічену та нерозмічену, де розмічені дані складають 100%, 50% та 20%.

4.2.2. Результати роботи алгоритму

У таблицях 4.1-4.3 приведені дані після 50 запусків алгоритму при кількості головних компонент 3, максимальній кількості слабких учнів 10, кількості найближчих сусідів 5 та початковому коефіцієнті при довжині кроку 0.6.

Як видно, на деяких запусках були створені моделі, що передбачали результати валідаційної вибірки на відмінно.

Таблиця 4.1. 20% відмічених даних.

	train (mean)	train (min)	train (max)	test (mean)	test (min)	test (max)
accuracy	0,786	0,596	0,926	0,734	0,294	0,941
balanced accuracy	0,751	0,514	0,903	0,685	0,208	0,958
precision (pos)	0,706	0,400	1,000	0,614	0,000	1,000
precision (neg)	0,846	0,656	0,974	0,795	0,462	1,000
recall (pos)	0,654	0,061	0,967	0,531	0,000	1,000
recall (neg)	0,849	0,594	1,000	0,838	0,364	1,000
f1	0,646	0,108	0,881	0,520	0,000	0,933
roc auc	0,751	0,514	0,903	0,685	0,208	0,958

Таблиця 4.2. 50% відмічених даних.

	train (mean)	train (min)	train (max)	test (mean)	test (min)	test (max)
accuracy	0,844	0,723	0,894	0,788	0,529	1,000
balanced accuracy	0,828	0,655	0,896	0,749	0,500	1,000
precision (pos)	0,755	0,583	0,913	0,676	0,000	1,000
precision (neg)	0,897	0,771	0,981	0,846	0,643	1,000
recall (pos)	0,783	0,467	0,970	0,642	0,000	1,000
recall (neg)	0,873	0,645	0,967	0,856	0,417	1,000
f1	0,762	0,519	0,848	0,625	0,000	1,000
roc auc	0,828	0,655	0,896	0,749	0,500	1,000

Таблиця 4.3. 100% відмічених даних.

	train (mean)	train (min)	train (max)	test (mean)	test (min)	test (max)
accuracy	0,864	0,819	0,926	0,789	0,588	1,000
balanced accuracy	0,858	0,775	0,936	0,781	0,367	1,000
precision (pos)	0,768	0,676	0,880	0,649	0,000	1,000
precision (neg)	0,921	0,831	0,982	0,885	0,545	1,000
recall (pos)	0,840	0,613	0,970	0,757	0,000	1,000
recall (neg)	0,876	0,803	0,952	0,805	0,462	1,000
f1	0,800	0,704	0,901	0,677	0,000	1,000
roc auc	0,858	0,775	0,936	0,781	0,367	1,000

РОЗДІЛ 5. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на бінарній класифікації на основі напівкерованого граничного бустингу.

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.

5.1. Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи класифікації на основі напівкерованого граничного бустингу. 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.

5.2. Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу розпізнавання та класифікації на основі напівкерованого граничного бустингу. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування.

F_2 – вибір методу реалізації завдання.

F_3 – вибір середовища програмування.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Python.

б) C++.

Функція F_2.

а) Власноруч реалізований алгоритм.

б) TensorFlow.

Функція F_3:

а) PyCharm.

б) Google colab.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

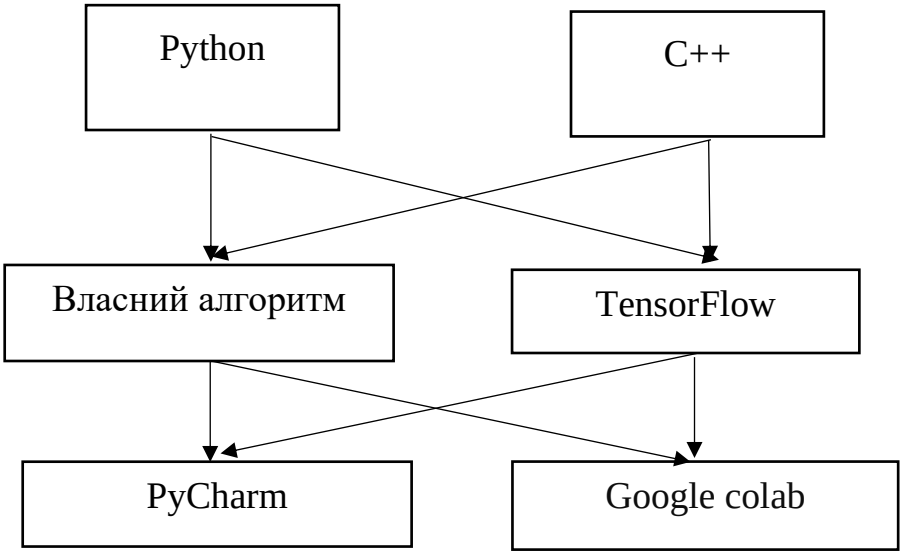

Рисунок 5.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантиів основних функцій. Позитивно-негативна матриця показана в таблиці 5.1.

Таблиця 5.1. Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Зручність, багатофункціональність, широкий вибір бібліотек	Швидкодія, обмежена підтримка апаратного прискорення
	Б	Висока швидкодія, доступ до оптимізаційних можливостей процесора,	Складність, менша кількість бібліотек, вимога до вміння ручного керування пам'яттю
F_2	*А*	Швидкодія, можливість глибоко розібратися в реалізації алгоритму.	Складність реалізації, великий об’єм коду.
	Б	Простий синтаксис, легкість у розробці та налагодженні моделей.	Менша швидкодія, та відсутність реалізації напівкерованого граничного бустингу.
F_3	*А*	Підтримка автодоповнення, інструменти для оптимізації, робота з великими проектами, наявність студентської підписки	Проблеми з налаштуванням деяких бібліотек, мала швидкодія.
	Б	Інтерактивність, можливість візуалізації даних	Потребує багато ресурсів, обмежена

		та результатів, зручність, швидкодія.	підтримка великих обсягів даних.

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:

Реалізація другого варіанту є проблематичною, тому доведеться реалізувати варіант А.

Функція F_3:

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

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2а – F_3б

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

5.3. 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 5.2.

Таблиця 5.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	10000	15000	18000
Об’єм пам’яті	X2	Мб	256	128	64
Час попередньої обробки даних	X3	мс	8	5	2
Потенційний об’єм програмного коду	X4	кількість рядків коду	1500	1000	500

За даними таблиці 5.3 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

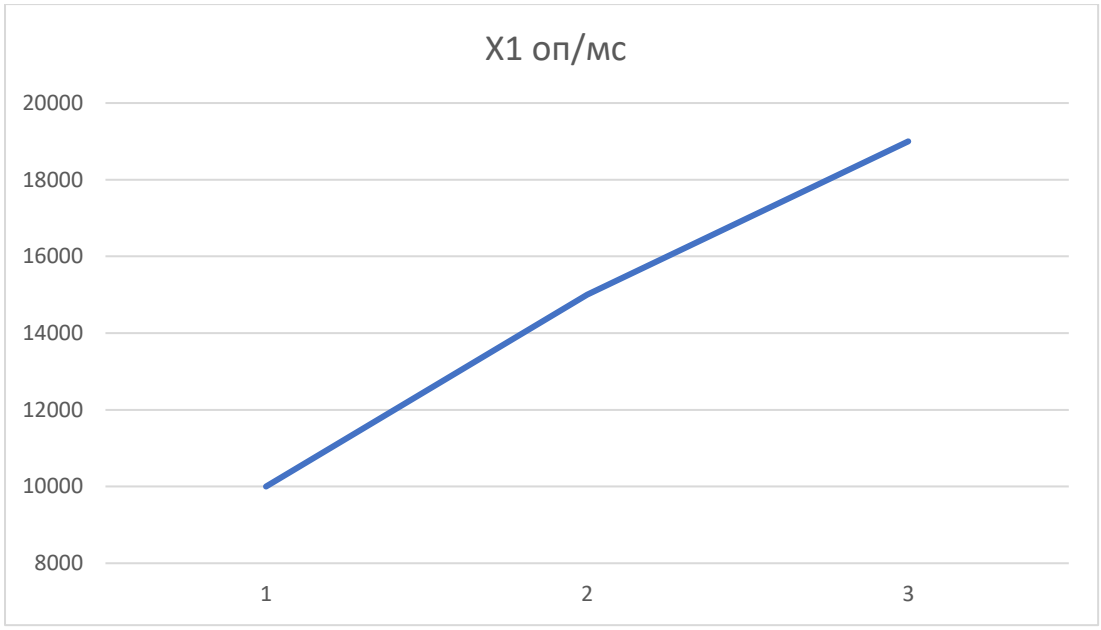

Рисунок 5.2 – X1, швидкодія мови програмування

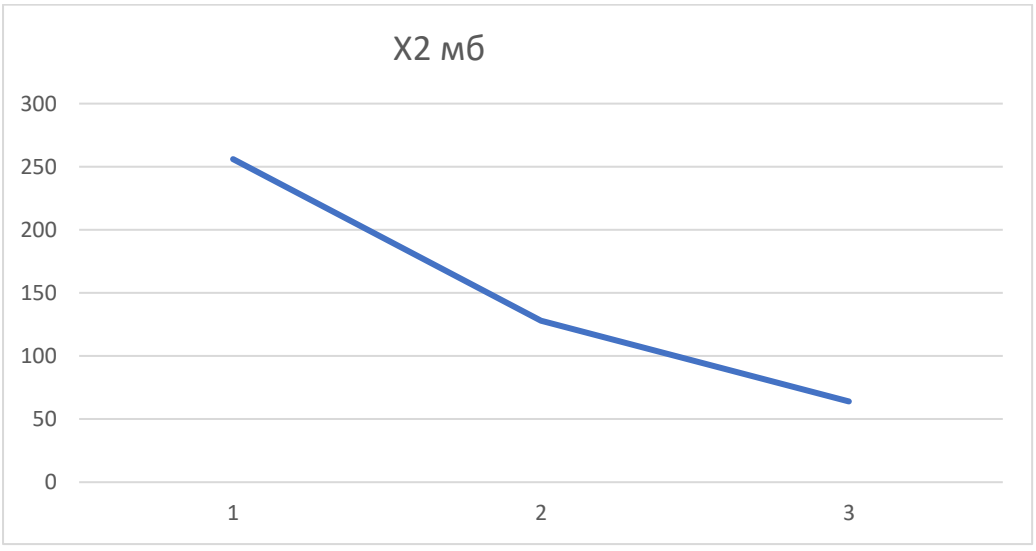

Рисунок 5.3 – X2, об’єм пам’яті

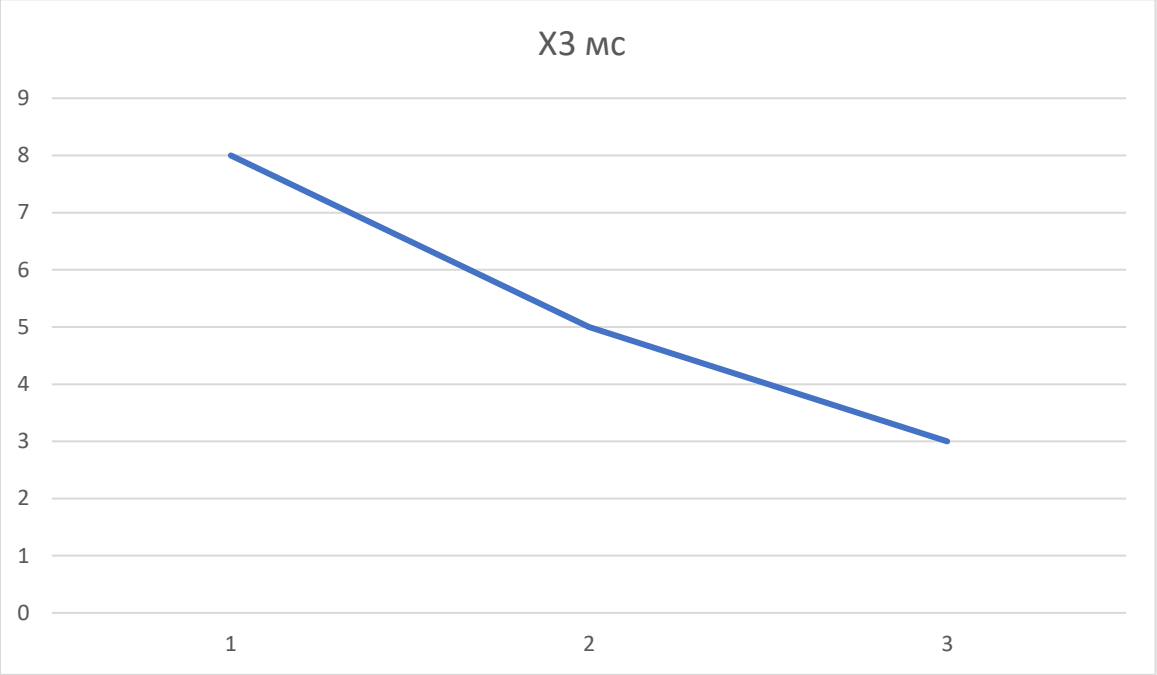

Рисунок 5.4 – X3, час попередньої обробки даних

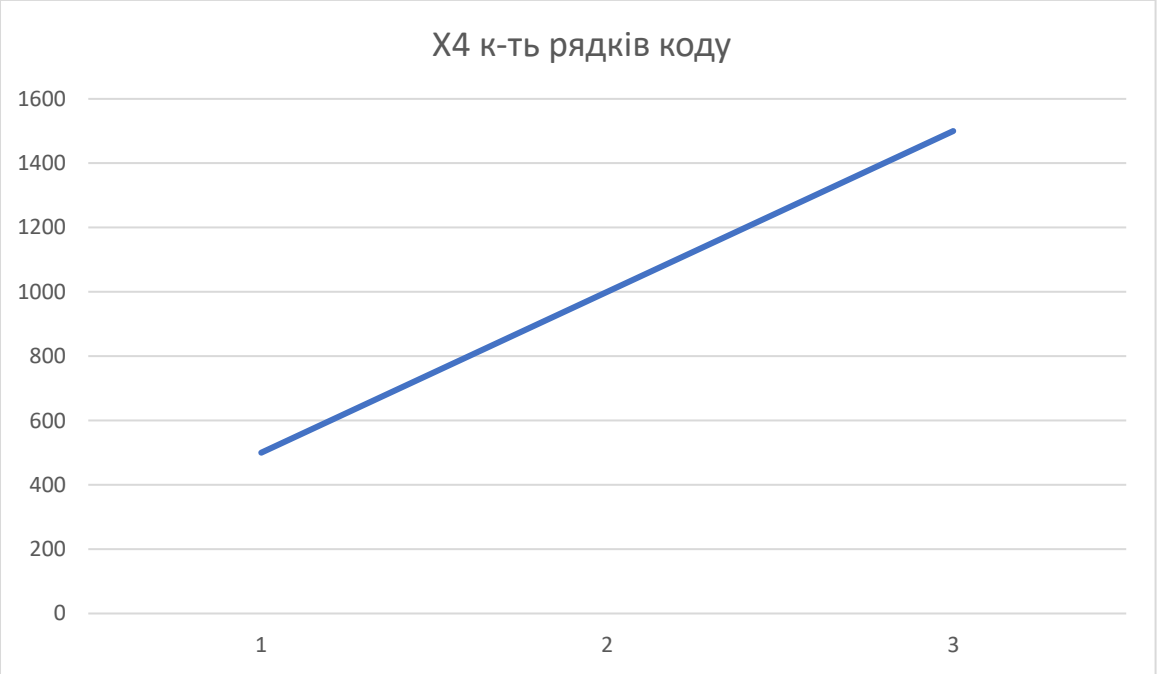

Рисунок 5.5 – X4, потенційний об’єм програмного коду

5.4. 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 5.3.

Таблиця 5.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці вимірy	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2

X1	Швидкодія мови програмування	С п/мс								14	-7	49
X2	Об’єм пам’яті	Mb								30	9	81
X3	Час попередньої обробки даних	мс								15	-6	36
X4	Потенційний об’єм програмного коду	Кількість рядків коду							6	25	4	16
	Разом		2	2	2	2	2	2	2	84	0	182

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

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

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 84, \qquad (5.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 21 \tag{5.2}$$

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

$$\Delta_i = R_i - T. \tag{5.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 182. \tag{5.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 182}{7^2(4^3 - 4)} = 0{,}742 > W_k = 0{,}67. \tag{5.5}$$

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 5.4.

Таблиця 5.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
X1 і X2								<	0,5
X1 і X3								<	0,5
X1 і X4								<	0,5
X2 і X3								>	1,5
X2 і X4								>	1,5
X3 і X4								<	0,5

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

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}. \quad (5.6)$$

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

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

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (5.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \tag{5.8}$$

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:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{5.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \tag{5.10}$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	1	2	3	4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1		,5	,5	,5	2,5	0,15	9,25	0,16	34,125	0,15
X2	,5		,5	,5	5,5	0,36	21,25	0,35	77,875	0,36
X3	,5	,5		,5	3,5	0,21	12,25	0,21	45,875	0,21

X4	,5	,5	,5		4 ,5	0, 28	1 6,25	0 ,28	5 9,125	0 ,28
Всього:					1 6	1	5 9	1	2 17	1

5.5. 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 5.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}\,, \qquad (5.11)$$

де n – кількість параметрів;

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

B_i – оцінка i–го параметра в балах.

Таблиця 5.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	15000	19	0,15	2,85
F3	А	X2	64	17	0,36	6,12
F4	А	X3	500	15	0,21	3,15
	Б	X4	1000	9	0,28	2,52

За даними з таблиці 5.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}], \quad (5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 2{,}85 + 6{,}12 + 3{,}15 = 12{,}12;$$

$$K_{K2} = 2{,}85 + 6{,}12 + 2{,}52 = 11{,}49.$$

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

5.6. 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:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (5.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

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

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – 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: $Т_Р$ = 26 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 $К_{СТ}$ = 0.7. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 26 \cdot 1.7 \cdot 0.7 = 30,94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 21 людино-днів, $К_П$ =1.4, $К_{СК}$ = 1, $К_{СТ}$ =0.9:

$$Т_2 = 21 \cdot 1.4 \cdot 0.9 = 26,46 \text{ людино-днів.}$$

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

$$Т_I = (30,94 + 23 + 5.4 + 26,46) \cdot 8 = 686,4 \text{ людино-годин.}$$

$$Т_{II} = (30,94 + 23 + 7.22 + 26,46) \cdot 8 = 700,96 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант ІІ.

В розробці беруть участь два програмісти з окладом 44 350 грн., один аналітик в області даних з окладом 38 806 грн. Визначимо середню зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.}, \quad (5.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_ч = \frac{44\,350 + 44\,350 + 38\,806}{3 \cdot 21 \cdot 8} = 252{,}98 \text{ грн.} \quad (5.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (5.16)$$

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:

І. $С_{ЗП} = 252{,}98 \cdot 686{,}4 \cdot 1.2 = 208\,381{,}23$ грн.

II. $С_{ЗП} = 252,98 \cdot 700,96 \cdot 1.2 = 212\ 801,44$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 208\ 381,23 \cdot 0.22 = 45\ 843,87$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 212\ 801,44 \cdot 0.22 = 46\ 816,31$ грн.

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

Так як одна ЕОМ обслуговує одного програміста з окладом 44 350 грн., з коефіцієнтом зайнятості 0,4 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 44\ 350 \cdot 0,4 = 212\ 880 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 212\ 880 \cdot (1 + 0.2) = 255\ 456 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 255\ 456 \cdot 0,22 = 56\ 200,32 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 34 500 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 34\ 500 = 10\ 350 \text{ грн.},$$

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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 34500 \cdot 0.05 = 2070 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 11) \cdot 8 \cdot 0.7 = 1355{,}2 \text{ год},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1355,2 \cdot 0,6 \cdot 0,4 \cdot 4,79 = 1557,94 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 34500 \cdot 0,67 = 23\ 115 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \quad (5.17)$$

$$С_{ЕКС} = 255\ 456 + 56\ 200,32 + 10\ 350 + 2070 + 1557,94 + 23\ 115 = 348\ 749 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 348\ 749 / 1355,2 = 257,34 \text{ грн/год.}$$

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:

$$С_М = С_{М-Г} \cdot Т, \quad (5.18)$$

I. $С_М = 257,34 \cdot 686,4 = 176\ 639$ грн.

II. $С_М = 257,34 \cdot 700,96 = 180\ 386$ грн.

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \quad (5.19)$$

I. $С_Н = 208\ 381,23 \cdot 0,67 = 139\ 615,42$ грн.

II. $С_Н = 212\ 801,44 \cdot 0,67 = 142\ 576,96$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \quad (5.20)$$

I. $С_{ПП} = 208\ 381,23 + 45\ 843,87 + 176\ 639 + 139\ 615,42 = 570\ 479,52$ грн.

II. $С_{ПП} = 212\ 801,44 + 46\ 816,31 + 180\ 386 + 142\ 576,96 = 582\ 580,71$ грн.

5.7. Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj} / С_{Фj}, \quad (5.21)$$

$$К_{ТЕР1} = 12,12 / 570\ 479,52 = 2,1245 \cdot 10^{-5},$$

$$К_{ТЕР2} = 11,49 / \ 582\ 580,71 = 1,9722 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 2,1245 \cdot 10^{-5}$.

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 $К_{ТЕР} = 2,1245 \cdot 10^{-5}$.

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.

5.8. Висновки до п’ятого розділу

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній роботі було проаналізовано існуючі роботи та підходи до реалізації методу напівкерованого граничного бустингу та запропоновано покращення для підвищення ефективності роботи алгоритму.

У першому розділі розглядається загальне поняття штучного інтелекту та його ознак, класифікація ШНМ і методів машинного навчання, обґрунтовано необхідність застосування напівкерованого машинного навчання.

У другому розділі наведено класифікацію методів напівкерованого навчання, проведено огляд існуючих підходів до реалізації алгоритму напівкерованого граничного бустингу та його складових компонент.

У третьому розділі проаналізовано недоліки існуючих підходів до реалізації алгоритму напівкерованого граничного бустингу та запропоновано відповідно покращення. Продемонстровано роботу алгоритму на штучно згенерованих двовимірних вибірках даних для бінарної класифікації з різним просторовим взаємним розташуванням класів. Наведено метрики якості роботи алгоритму.

У четвертому розділі продемонстровано роботу запропонованого алгоритму на справжніх даних, пов’язаних із медичною діагностикою такого захворювання як інфекційний ендокардит. Наведено довідку про хворобу, описано процес підготовки даних для роботи з алгоритмом та наведено метрики якості роботи алгоритму.

Четвертий розділ являє собою економічний аналіз вартості розробки програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Combining Labeled and Unlabeled Data with Co-Training. Avrim Blum, Tom Mitchell. [Електронний ресурс]. Режим доступу : cotrain.pdf (cmu.edu).
2. Semi-Supervised MarginBoost . F. d'Alche-Buc, Yves Grandvalet, Christophe Ambroise. [Електронний ресурс]. Режим доступу: https://papers.nips.cc/paper_files/paper/2001/file/931af583573227f0220bc568c65ce104-Paper.pdf.
3. Efficient Margin Maximizing with Boosting. Gunnar Raetsch, Manfred K. Warmuth. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/220319881_Efficient_Margin_Maximizing_with_Boosting.
4. L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In Advances in Large Margin Classifiers. MIT, 2000. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/243689632_Functional_gradient_techniques_for_combining_hypotheses.
5. J.-P. Aubin. L 'analyse non lineaire et ses applications d l'economie. Masson, 1984. [Електронний ресурс]. Режим доступу: https://www.scirp.org/%28S%28czeh2tfqyw2orz553k1w0r45%29%29/reference/referencespapers.aspx?referenceid=632035.
6. G.J. McLachlan and T. Krishnan. The EM algorithm and extensions. Wiley, 1997. [Електронний ресурс]. Режим доступу: http://www.leg.ufpr.br/~paulojus/EM/EM-Krishnan-McLachlan.pdf.
7. C. Ambroise and G. Govaert. EM algorithm for partially known labels. In IFCS 2000, july 2000. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/268243407_EM_Algorithm_for_Partially_Known_Labels.

8. SERBoost: Semi-supervised Boosting with Expectation Regularization Amir Saffari, Helmut Grabner, and Horst Bischof. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/221303725_SERBoost_Semi-supervised_Boosting_with_Expectation_Regularization.
9. On improving semi-supervised marginboost incrementally using strong unlabeled data. Thanh-Binh Le and Sang-Woon Kim; Department of Computer Engineering, Myongji University, 449-728, Yongin, South Korea. [Електронний ресурс]. Режим доступу: https://www.scitepress.org/Papers/2012/37212/37212.pdf.
10. Semi-supervised Learning via Regularized Boosting Working on Multiple Semi-supervised Assumptions Ke Chen, Senior Member, IEEE, and Shihai Wang. [Електронний ресурс]. Режим доступу: https://pubmed.ncbi.nlm.nih.gov/20421671/.
11. Information Theoretic Regularization for Semi-Supervised Boosting. Lei Zheng, Shaojun Wang, Yan Liu, Chi-Hoon Lee. [Електронний ресурс]. Режим доступу: https://cs.gmu.edu/~carlotta/teaching///CS775-s10/readings/InformationKDD09.pdf.
12. Mallapragada, P. K., J. R. J. A. K. L. Y. (2009). Semiboost: Boosting for semi-supervised learning. IEEE Trans. Pattern Anal. and Machine Intell., 31(11):2000–2014. [Електронний ресурс]. Режим доступу: https://pubmed.ncbi.nlm.nih.gov/19762927/.
13. Штучні нейронні мережі: Обчислення. М.А. Новотарський, Б.Б. Нестеренко, Інститут математики НАН України, Київ 2004. [Електронний ресурс]. Режим доступу: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=PeaNhQYAAAAJ&citation_for_view=PeaNhQYAAAAJ:q3oQSFYPqjQC.
14. Hopfield J.J. Neural with graded response have collective computational properties like those of two-state neurons // Proceedings of the National Academy of Science of the USA, 1984.–vol.81.–P.3088-3092.

15. Yoav Freund, Robert E Schapire // A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer and System Sciences, Volume 55, Issue 1, 1997, Pages 119-139, ISSN 0022-0000. [Електронний ресурс]. Режим доступу: https://doi.org/10.1006/jcss.1997.1504.
16. Decision and Classification Trees, Clearly Explained. [Електронний ресурс]. Режим доступу: https://youtu.be/_L39rN6gz7Y?t=700.
17. Інфекційний ендокардит. [Електронний ресурс]. Режим доступу: https://empendium.com/ua/chapter/B27.II.2.13.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

ssmb.py:

```
import numpy as np
import pandas as pd
from sklearn.base import ClassifierMixin, BaseEstimator
from sklearn.linear_model import Perceptron
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier

from src.algorithm.bc_palette import BaseClassifierPalette
from src.algorithm.boosting_classifier import BoostingClassifier


class SSMBoost(BoostingClassifier[BaseClassifierPalette], ClassifierMixin, BaseEstimator):
    def __init__(self, n_estimators: int = 10, n_neighbors: int = 5,
                 learning_rate: float = 0.5, max_lr_iter: int = 15,
                 random_state=None):
        self.n_estimators = n_estimators
        self.n_neighbors = n_neighbors
        self.learning_rate = learning_rate
        self.max_lr_iter = max_lr_iter
        self.estimators = []
        self.classifier_weights = np.zeros(n_estimators)
        self.random_state = random_state

        self.cost = []

    def fit(self,
            X_labeled: np.ndarray | pd.DataFrame,
            y_labeled: np.ndarray | pd.Series,
            X_unlabeled: np.ndarray | pd.DataFrame):
        X_labeled: np.ndarray = X_labeled.to_numpy() if isinstance(X_labeled, pd.DataFrame) else
X_labeled
        y_labeled: np.ndarray = y_labeled.to_numpy() if isinstance(y_labeled, pd.Series) else
y_labeled
        X_unlabeled: np.ndarray = X_unlabeled.to_numpy() if isinstance(X_unlabeled, pd.DataFrame)
else X_unlabeled

        self.estimators.clear()
        self.cost.clear()

        n_labeled_samples: int = np.size(X_labeled, axis=0)
        assert n_labeled_samples == np.size(y_labeled, axis=0)
        n_unlabeled_samples: int = np.size(X_unlabeled, axis=0)

        n_samples: int = n_labeled_samples + n_unlabeled_samples
        n_features: int = np.size(X_labeled, axis=1)
        assert n_features == np.size(X_unlabeled, axis=1)

        sample_weights: np.ndarray = np.full(n_samples, 1 / n_samples)

        eps: float = np.finfo(self.classifier_weights.dtype).eps
        fmin: float = np.finfo(self.classifier_weights.dtype).min
        fmax: float = np.finfo(self.classifier_weights.dtype).max

        knn = KNeighborsClassifier(n_neighbors=self.n_neighbors)
        knn.fit(X_labeled, y_labeled)
        pseudo_labels_proba = knn.predict_proba(X_unlabeled) if np.size(X_unlabeled, axis=0) > 0
else []
        pseudo_labels = np.array([[{0: -1, 1: 1}[np.argmax(p)], np.max(p)] for p in
```

```
pseudo_labels_proba])

        for t in range(1, self.n_estimators + 1):
            if t % 10 == 0:
                p10: int = np.ceil(n_unlabeled_samples * 0.1).astype(int)
                if p10 > 0:
                    # pick p10 strongest confident samples from unlabeled samples using
_confidence_class1
                    most_confident_indices = np.argsort(pseudo_labels[:, 1])[-p10:]

                    # add p10 strongest confident samples to labeled samples
                    X_labeled = np.concatenate((X_labeled, X_unlabeled[most_confident_indices]),
axis=0)
                    y_labeled = np.concatenate((y_labeled, pseudo_labels[most_confident_indices,
0]), axis=0)
                    # remove p10 strongest confident samples from unlabeled samples
                    X_unlabeled = np.delete(X_unlabeled, most_confident_indices, axis=0)
                    pseudo_labels = np.delete(pseudo_labels, most_confident_indices, axis=0)
                    # update n_labeled_samples and n_unlabeled_samples
                    n_labeled_samples += p10
                    n_unlabeled_samples -= p10

            # train a weak classifier on the new labeled samples
            # weak_classifier = DecisionTreeClassifier(max_depth=1, random_state=self.random_state)
            weak_classifier = BaseClassifierPalette([
                DecisionTreeClassifier(max_depth=1, random_state=self.random_state),
                LinearSVC(random_state=self.random_state),
                Perceptron(random_state=self.random_state),
            ])
            sw_labeled = sample_weights[:n_labeled_samples] /
np.sum(sample_weights[:n_labeled_samples])
            weak_classifier.fit(X_labeled, y_labeled, sample_weight=sw_labeled)

            h_t = weak_classifier.predict(np.concatenate([X_labeled, X_unlabeled], axis=0))

            # compute the weighted error
            weighted_error = np.clip(
                np.sum(sw_labeled * (h_t[:n_labeled_samples] != y_labeled)),
                eps, 1
            )

            # compute the classifier weight
            classifier_weight = self.learning_rate * np.log((1 - weighted_error) / weighted_error)

            self.estimators.append(weak_classifier)
            self.classifier_weights[t - 1] = classifier_weight

            g_t_labeled = self.decision_function(X_labeled) if n_labeled_samples > 0 else
np.array([])
            g_t_unlabeled = self.decision_function(X_unlabeled) if n_unlabeled_samples > 0 else
np.array([])

            JLt = np.sum(
                sample_weights[:n_labeled_samples] * y_labeled * h_t[:n_labeled_samples]
            ) if n_labeled_samples > 0 else 0
            JUt = np.sum(
                sample_weights[n_labeled_samples:] * 2 * g_t_unlabeled * h_t[n_labeled_samples:]
            ) if n_unlabeled_samples > 0 else 0
            JSt = JLt + JUt

            if JSt <= 0:
                self.estimators.pop()
                break

            cost = self.cost_functional(X_labeled, X_unlabeled, y_labeled)
            if t % 10 != 0 or n_unlabeled_samples == 0:
                for _ in range(1, self.max_lr_iter + 1):
                    if len(self.cost) == 0 or cost < self.cost[-1]:
```

```
                    break
                self.classifier_weights[t - 1] /= 2
                cost = self.cost_functional(X_labeled, X_unlabeled, y_labeled)
            self.cost.append(cost)

            # update the sample weights
            if n_labeled_samples > 0:
                sample_weights[:n_labeled_samples] = self.margin_cost_derivative(y_labeled *
g_t_labeled)
            if n_unlabeled_samples > 0:
                sample_weights[n_labeled_samples:] = self.margin_cost_derivative(g_t_unlabeled ** 2)

            # normalize the sample weights
            sample_weights_sum = np.clip(np.sum(sample_weights), fmin, fmax)
            if np.abs(sample_weights_sum) < eps:
                break
            sample_weights /= sample_weights_sum

        # find the first minimal cost index and drop the rest of classifiers
        min_cost_index = np.argmin(self.cost)
        self.estimators = self.estimators[:min_cost_index + 1]

    def cost_functional(self, X_labeled, X_unlabeled, y_labeled):
        n_labeled_samples: int = np.size(X_labeled, axis=0)
        n_unlabeled_samples: int = np.size(X_unlabeled, axis=0)

        return (np.sum(self.margin_cost(y_labeled * self.decision_function(X_labeled)))
                if n_labeled_samples > 0 else 0) + \
            (np.sum(self.margin_cost(self.decision_function(X_unlabeled) ** 2))
             if n_unlabeled_samples > 0 else 0)

    @staticmethod
    def margin_cost(x):
        return np.exp(-x)

    @staticmethod
    def margin_cost_derivative(x):
        return -np.exp(-x)  # \frac{\partial\exp(-x)}{\partial x} = -\exp(-x)
```

boosting_classifier.py:

```
import numpy as np
import pandas as pd

from typing import TypeVar, Generic

C = TypeVar('C')


class BoostingClassifier(Generic[C]):
    estimators: list[C]
    classifier_weights: np.ndarray

    def decision_function(self, X: np.ndarray | pd.DataFrame) -> np.ndarray:
        if isinstance(X, pd.DataFrame):
            X = X.to_numpy()

        predictions = np.zeros(np.size(X, axis=0), dtype=np.float32)

        clf: C
        clf_weight: np.float32
        for clf, clf_weight in zip(self.estimators, self.classifier_weights):
            predictions += clf_weight * clf.predict(X)
```

```
        return predictions

    def predict(self, X: np.ndarray | pd.DataFrame) -> np.ndarray:
        return np.sign(self.decision_function(X))

    def predict_proba(self, X: np.ndarray | pd.DataFrame) -> np.ndarray:
        predictions = self.decision_function(X) / np.sum(self.classifier_weights)
        return np.array([[np.abs(p) if p < 0 else 1 - p, p if p > 0 else 1 + p]
                         for p in predictions])
```

bc_palete.py:

```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin


class BaseClassifierPalette(BaseEstimator, ClassifierMixin):
    def __init__(self, classifiers: list):
        self.classifiers = classifiers
        self.classifier = None

    def fit(self, X, y, sample_weight=None):
        errors = []
        if sample_weight is None:
            sample_weight = np.ones(np.size(X, axis=0)) / np.size(X, axis=0)
        for classifier in self.classifiers:
            classifier.fit(X, y, sample_weight=sample_weight)
            predictions = classifier.predict(X)
            errors.append(np.clip(np.sum(sample_weight * (predictions != y)), 0, 1))
        self.classifier = self.classifiers[np.argmin(errors)]

    def predict(self, X):
        return self.classifier.predict(X)
```