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

Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

«На правах рукопису»

УДК 004.85:004.93’1:61](043.3)

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Системи і методи штучного інтелекту»
зі спеціальності 122 «Комп'ютерні науки»
на тему:
“Інтелектуальна система класифікації на основі графового підходу”

Виконав:
студент ІІ курсу, групи КІ-31-мп
Ручкін Олександр Костянтинович _____

Науковий керівник:
професор кафедри ШІ, д.т.н., професор,
Синєглазов Віктор Михайлович _____

Рецензент:

_____

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

Київ - 2024

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

Навчально-науковий інститут прикладного системного аналізу
Кафедра штучного інтелекту

Рівень вищої освіти – другий (магістерський)
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«29» серпня 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Ручкін Олександр Костянтинович

1. Тема дисертації «**Інтеллектуальна система классифікації на основі графового підходу**»,

науковий керівник дисертації: Синєглазов Віктор Михайлович, професор кафедри ШІ, д.т.н., професор, затверджена наказом по НН ІПСА від "07" листопада 2024 р. № 5001-с

2. Термін подання студентом дисертації «13» грудня 2024 року

3. **Об’єкт дослідження:** методи напівкерованого навчання на основі графів GSSL, що застосовують рівняння дифузії для розповсюдження міток та використовуються для вирішення задач класифікації даних

4. **Вихідні дані:** наукові статті по методам напівкерованого навчання зазначені у переліку джерел

5. **Перелік завдань, які потрібно розробити**: дослідити сучасний стан напряму GSSL, розробити узагальнений метод GSSL – дифузне розповсюдження міток DLS; розробити програмне забезпечення інтелектуальної системи классифікації на основі GSSL – DLS; провести тестування та аналіз результатів роботи інтелектуальної системи; на основі інтелектуальної системи підготувати стартап-проект; зробити висновки з отриманих результатів та оформити пояснювальну записку

6. **Орієнтовний перелік графічного (ілюстративного) матеріалу:** презентація

7. Орієнтовний перелік публікацій: заплановано апробування результатів дослідження на всеукраїнських, міжнародних і закордонних конференціях, опублікування у періодичних виданнях з переліку наукометричної бази Scopus.

8. Консультанти розділів роботи

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

9. Дата видачі завдання: «30» серпня 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формулювання і уточнення теми та постановка задачі	01.09.2024 - 07.09.2024	Виконано
2.	Проведення дослідження сучасного стану задачі та огляд методів за напрямом напівкерованого навчання на основі графів	07.09.2024 - 21.09.2024	Виконано
3.	Розробка узагальненого методу напівкерованого навчання на основі графів	21.09.2024 - 07.10.2024	Виконано
4.	Розробка інтелектуальної системи классифікації на основі графового підходу	07.10.2024 - 21.10.2024	Виконано
5.	Тестування та аналіз результатів роботи інтелектуальної системи	21.10.2024 - 07.11.2024	Виконано
6.	Підготовка старт – ап проекту на основі інтелектуальної системи классифікації	07.11.2024 - 21.11.2024	Виконано
7.	Підготовка пояснювальної записки	21.11.2024 - 13.12.2024	Виконано
8.	Захист магістерської дисертації		Виконано

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник проекту _____ _____

(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація: 116 с., 250 рис., 2 табл., 44 посилань, 2 додатка.

GRAPH BASED SEMI-SUPERVISED LEARNING, LABEL PROPAGATION, LABEL SPREADING, POISSON LEARNING, DIFFUSE EQUATION

Назва дослідження: «Інтелектуальна система класифікації на основі графового підходу».

Мета роботи: розробка нових і поліпшення існуючих методів GSSL та на їх основі розробка інтелектуальної системи вирішення задач класифікації для синтетичних та реальних даних.

Об'єкт дослідження: методи напівконтрольованого навчання на основі графів, що використовуються для вирішення задач класифікації, коли є лише обмежена кількість мічених даних у великих наборах.

Предмет дослідження: предметом дослідження є гібридні методи GSSL засновані на вирішення рівняння дифузії для застосування у алгоритму розповсюдження міток у великих графових моделях.

Наукова новизна: запропоновано новий метод GSSL - узагальнений підхід Diffuse Label Propagation на основі рівнянь Пуассону.

Практичне значення: розроблено програмне забезпечення інтелектуальної системи классифікації даних. Програма дозволяє: відображати дані, будувати граф даних, обчислювати метрики, будувати графік точності для заданої послідовності початкових даних, обчислювати та будувати статистичні характеристики середнього та середньої похибки точності методу. Система орієнтована на науковців, інженерів даних та студентів, які працюють із задачами машинного навчання та класифікації. Програма також орієнтована на застосування у медицині як інструмент автоматичної медичної діагностиці та класифікації ризиків серцево-судинних захворювань.

Результати цієї роботи були апробовані на міжнародних конференціях.

ABSTRACT

Master's thesis: 122 p., 25 figures, 2 tables., 44 references, 2 appendix.

Keywords: GRAPH BASED SEMI-SUPERVISED LEARNING, LABEL PROPAGATION, LABEL SPREADING, POISSON LEARNING

Title of the investigation: "Intelligent Classification System Based on a Graph Approach".

Objective: The aim of the work is to develop new and improve existing GSSL methods and, on their basis, create an intelligent system for solving classification tasks for synthetic and real-world data.

The object of research: The object of study is graph-based semi-supervised learning (GSSL) methods used for solving classification tasks when only a limited amount of labeled data is available in large datasets.

The subject of research: The subject of study is hybrid GSSL methods based on solving Laplace and Poisson equations for label propagation in large graph models.

Scientific novelty: A new GSSL method is proposed—a generalized Diffuse Learning approach based on Poisson equations.

Practical significance: Software for an intelligent data classification system has been developed. The program provides the following functionalities: visualizing datasets, constructing data graphs; calculating metrics; plotting accuracy graphs for a given sequence of initial data; calculating and visualizing statistical characteristics of the method’s mean accuracy and mean error.

The system is designed for researchers, data engineers, and students working on machine learning and classification tasks. The program can also be used as a tool for automatic medical diagnostics and risk classification of cardiovascular diseases.

The results of this work were tested at international conferences.

ЗМІСТ

ВСТУП

Актуальність: У останні роки графові методи напівконтрольованого навчання (Graph-Based Semi-Supervised Learning, GSSL) набувають популярності у різноманітних теоретичних напрямах машинного навчання: классифікації, кластеризації, навчанні та інш. GSSL методи набули великої популярності за рахунок простоти їх застосування та достатньої точності результату. GSSL методи мають широкий спектр застосувань у багатьох практичних напрямах, зокрема у машинному навчанні, обробці зображень, аналізі соціальних мереж та медичних дослідженнях.

Для вирішення задачі классифікації - напівкероване навчання потребує застосування мічених та немічених даних. Але у багатьох реальних випадках отримання мічених даних є складним, дорогим та тривалим процесом. Наприклад, так відбувається при класифікуванні захворювань під час пандемії. Немічені дані експоненціально зростають, а мічені зростають дуже - дуже повільно, що є критичним фактором.

Сучасні алгоритми GSSL дозволяють вирішити ці задачі, но потребують величезних числових та часових ресурсів, та відповідно фінансових і енергетичних витрат. Зменшення цих енергетичних витрат є важливою та актуальною задачею «зеленої» інтелектуалізованої сучасності. Розробка нових та удосконалення існуючих методів GSSL дозволяє просунутися в цьому напрямі та отримати нові поліпшені результати. Дослідженню цього напряму присвячена дана магістерська дисертація.

Також у сучасний час у напрямі напівкерованого навчання збільшується кількість прикладів застосування моделей розповсюдження міток, які використовують диференціальні рівняння у часткових похідних другого порядку – однорідні та неоднорідні рівняння дифузії. Та їх математичні моделі - рівняння Лапласа, рівняння Пуассона та інш. Одним із добре відомих підходів у цьому

напрямку є використання рівняння Лапласа, також відомий, як Лапласове навчання. Однак, як показують нещодавні дослідження, використання рівняння Пуассона в напівкерованому навчанні має більш широкі перспективи на майбутнє. Хоча навчання за Лапласом та навчання за Пуассоном мають багато спільного, але є і суттєва різниця.

Відомо, що рівняння Лапласа застосовується до різних фізичних процесів, таких як розподіл тепла (рівняння теплопровідності), розподіл електростатичного потенціалу та інші. Однак рівняння Лапласа можна застосовувати тільки в окремих випадках, де існують однорідні фізичні поля, такі як однорідне електростатичне поле або однорідне теплове поле. Якщо нам потрібно розглянути неоднорідну задачу і неоднорідне поле, то використовується узагальнення рівняння Лапласа на неоднорідне поле, відоме як рівняння Пуассона. Взагалі рівняння Пуассона узагальнює розв'язок рівняння Лапласа на неоднорідні фізичні поля або, в нашому випадку, на неоднорідні інформаційні поля даних.

У теорії інформації та даних наявність зав'язків між даними означає, що дані представлені в неоднорідному просторі при дослідженні поля даних і їх розподілу у просторі даних. Також необхідно припустити, що структура неоднорідності коректна і може бути описана аналогічним ітераційним алгоритмом. Отже, використання рівняння Пуассона для виявлення класів даних і залежностей всередині класу в неоднорідному просторі даних може бути більш перспективним, ніж використання рівнянь Лапласа. оскільки функція неоднорідності застосовується в побудові рішення. Це суттєво, коли у нас є невелика кількість вихідних точок даних (маркованих даних). При невеликій кількості маркованих та великій кількості загальних даних навчання Пуассона має значну перевагу перед навчанням Лапласа також у часі обчислення.

Метод розповсюдження міток на основі рівняння Лапласу **(Label Propagation)** було запропоновано **[1,2] [Xiaojin (Jerry) Zhu].** Метод полегшує поширення відомих ваг по графу, дозволяючи вивести немічені вузли на основі їх відстані та зав'язків з поміченими аналогами. Такий підхід не тільки збільшує

цінність невеликих обсягів доступних розмічених даних, але й ефективно використовує величезні запаси нерозмічених даних. Розвиток цього напряму вважається дуже перспективним та ефективним для вирішення багатьох прикладних задач. Пуассонове навчання (Poisson Learning) [10] аналогічне Лапласовому, але розповсюдження міток відбувається за допомогою рівняння Пуассону, а не рівняння Лапласу. Пуассонівське навчання дозволяє використовувати меншу кількість маркованих даних, та збільшену кількість загальних немаркованих. Таким чином, дослідження та використання Пуассонового навчання, зокрема напівкерованого навчання на основі графів із застосуванням методу Пуассона, є більш перспективним напрямком, який вимагає подальших досліджень для різних прикладних задач. Далі в роботі будемо застосовувати дослідження навчання за Пуассоном**.**

Назва дослідження: «Інтелектуальна система классифікації на основі графового підходу».

Мета роботи: є розробка нових і поліпшення існуючих методів GSSL та на їх основі розробка інтелектуальної системи вирішення задач класифікації для синтетичних та реальних даних.

Задачі роботи:

- дослідити сучасний стан напряму та провести огляд методів за напрямом напівкерованого навчання на основі графів;
- розробити узагальнений метод напівкерованого навчання на основі графів. А саме, розробити узагальнений метод GSSL – Diffuse Label Spreading (DLS) на основі неоднорідних рівнянь дифузії (рівнянь Пуассону) та додаткової регулярізації;
- розробити програмне забезпечення інтелектуальної системи классифікації на основі графового підходу та запропонованого узагальненого методу DLS;
- провести тестування та аналіз результатів роботи інтелектуальної системи

- на основі інтелектуальної системи підготувати стартап-проект.

Об'єкт дослідження: є методи напівконтрольованого навчання на основі графів (Graph based Semi-Supervised Learning, GSSL), що активно використовуються для вирішення задач класифікації, коли є лише обмежена кількість мічених даних у великих наборах.

Предмет дослідження: є гібридні методи GSSL засновані на вирішення рівняння Лапласу та Пуассону для застосування у розповсюдження міток у великих графових моделях. Зокрема, буде досліджувано механізми, що використовують властивості рівняння Пуассона для поширення міток з мічених до немічених зразків. Цей підхід дозволе ефективно використовувати інформацію про структуру графа для поширення міток, зменшуючи вплив шуму та покращуючи якість класифікації навіть при малій кількості мічених точок. Використання модифікованого рівняння Пуассона дозволе створити гладке поле міток, яке враховує топологію графа і забезпечує плавну зміну міток між зразками, що сприяє кращій узгодженості класифікації. Також буде проведено дослідження впливу регулярізації та оптимізації на цей підхід.

Очікуємо наукову новизну: буде запропоновано новий метод GSSL - узагальнений підхід Diffuse Label Spreading на основі рівнянь Пуассону та додаткової регулярізації. Застосування рівняння Пуассону до розповсюдження міток у GSSL є новим підходом який досліджено лише у декілька сучасних роботах, загалом не більше 5 джерелах.

Практичне значення: буде розроблено програмне забезпечення інтелектуальної системи классифікації даних, яке має практичне значення. Програма дозволяє: відображати дані, будувати граф даних, обчислювати метрики, Будувати графік точності для заданої послідовності початкових даних, обчислювати та будувати статистичні характеристики середнього та середньої похибки точності методу для різноманітних практичних задач класифікування. Так програма може бути застосована як інструмент автоматичної медичної діагностиці та класифікації ризиків серцево-судинних захворювань

Взагалі система орієнтована на науковців, інженерів даних та студентів, які працюють із задачами машинного навчання та класифікації.

Структура дисертації: магістерська дисертація складається з п’яти суттєвих розділів: вступ; перший розділ - огляд методів за напрямом напівкерованого навчання; другий розділ - розробка узагальненого методу напівкерованого навчання на основі графів; третій розділ - розробка інтелектуальної системи классифікації на основі графового підходу; четвертий розділ - тестування та аналіз результатів роботи інтелектуальної системи; п’ятий розділ - підготовка старт–ап проекту на основі інтелектуальної системи классифікації; висновки.

РОЗДІЛ 1 ОГЛЯД МЕТОДІВ ЗА НАПРЯМОМ НАПІВКЕРОВАННОГО НАВЧАННЯ НА ОСНОВІ ГРАФІВ

У цьому розділі проведемо дослідження сучасного стану напряму напівкерованого навчання на основі графів та проведемо огляд методів за напрямом напівкерованого навчання на основі графів.

1.1 Розвиток методів напівкерованого навчання в останні роки

На сьогодні існує безліч підходів до реалізації напівкерованого навчання, детально розглянутих у численних оглядах [1–7, 9, 11]. Ефективність графових структур полягає в їх здатності інкапсулювати складні взаємозв'язки в даних. Розглядаючи дані як структуровану множину, можна скористатися прихованою інформацією, закладеною в топологічній структурі.

Поточні дослідження в області напівкерованого навчання на графах (GSSL) зосереджені на таких методах, як поширення міток, гаусові випадкові поля, гармонійні функції та інші. Серед робіт, що вивчають ці методи, можна виділити наступні: [1–6, 9]. Деякі з цих робіт надають загальну таксономію методів GSSL.

Одним із перших фундаментальних досліджень у цій галузі є дисертація Чжу [1]. У цій роботі проведено комплексний аналіз напівкерованого навчання на основі графів із використанням рівняння Лапласа та гармонійних функцій. На основі цих результатів пізніше було опубліковано детальніший огляд [2], який охоплює роботи, пов'язані із застосуванням цього підходу та напівкерованим навчанням із використанням традиційних класифікаторів. У статті також обговорюються результати застосування різних елементів регуляризації та їх

використання у практичних задачах, таких як розпізнавання візуальних об'єктів та картографування слів.

Подальші фундаментальні огляди напівкерованого навчання представили класифікацію та таксономію існуючих математичних підходів у GSSL [3–7]. В огляді [3] автори зосереджуються на поділі методів напівкерованого навчання на індуктивні та трансдуктивні. Індуктивні методи, які зазвичай розширюють контрольовані алгоритми для включення немаркованих даних, диференціюються на основі того, як вони використовують немарковані дані: на етапі попередньої обробки, безпосередньо в цільовій функції або через крок псевдомаркування. Трансдуктивні методи базуються на графах і групуються на основі вибору, зробленого на різних етапах процесу навчання.

Дослідження [4] фокусується на масштабованості методів GSSL для великих наборів даних, тобто великих графів. Для покращення масштабованості методів напівкерованого навчання на графах пропонується використовувати механізм гранулярності.

У роботі [5] розглядаються та порівнюються наступні методи напівкерованого навчання: Laplace Label Propagation, Directed Regularization, Manifold Regularization, Deformed Graph Laplacian, Poisson Learning, Factorization Based Methods, Lazy Random Walk та інші. Робота [6] присвячена огляду моделей глибокого навчання, особливо використанню генеративних моделей та їх різних типів.

Слід також відзначити кілька цікавих методів, досліджених у літературі: MBO-схема для класифікації [34], метод вбудованого ядра [35], Sparse Label Propagation [36], Weighted Non-Local Laplacian (WNLL) [37]. Незважаючи на свою ефективність, не існує універсального методу, який би апріорі визначав найкращий підхід для конкретної проблеми. Більше того, неможливо гарантувати, що введення немаркованих даних не призведе до погіршення продуктивності. Таке погіршення спостерігалося на практиці і, ймовірно, недооцінене через упередженість публікацій [2]. Проблема погіршення продуктивності була

виявлена й в інших дослідженнях [38], [39]. Це питання особливо актуальне у випадках, коли високої продуктивності можна досягти за допомогою суто керованих класифікаторів.

У кількох роботах [11], [16] проведено незалежну оцінку ефективності різних методів напівкерованого навчання на різних наборах даних. У роботі [38] емпірично порівняли одинадцять різних алгоритмів напівкерованого навчання, включаючи напівкеровані опорні вектори, методи найближчого сусіда, поширення міток та регуляризацію розмаїття, застосовуючи гіперпараметричну оптимізацію до кожного алгоритму. Порівнюючи продуктивність алгоритмів на восьми різних наборах даних, автори показали, що жоден алгоритм не перевершує інші рівномірно. Деякі набори даних показали значне покращення продуктивності порівняно з базовою лінією, тоді як інші показали зниження продуктивності.

В останні роки варіаційний підхід, який використовує диференціальні рівняння в частинних похідних, зокрема рівняння Лапласа, став популярним методом напівкерованого навчання [1, 2, 9, 10]. Навчання за Лапласом передбачає, що в наборі даних є набір початкових міток, які поширюються до всіх вузлів за допомогою гармонійної функції, заданої рівнянням Лапласа. Цей метод використовувався в роботах [1, 2], [40], [41]. У деяких роботах використовувалися більш складні диференціальні рівняння, наприклад, у [8] розглядалася адаптація двох диференціальних рівнянь в частинних похідних: рівняння p-Лапласіана та рівняння Ейконала.

У [9] запропоновано новий підхід під назвою Poisson Learning, який використовується для напівкерованого навчання на графах з високою швидкістю. Навчання за Пуассоном вирішує проблему виродження лапласівського напівкерованого навчання. Цей метод вводить неоднорідну функцію (так звані "джерела" і "стоки", засновані на початкових значеннях позначених вершин) і розв'язує отримане рівняння Пуассона на графі. Отримані результати є більш стабільними та інформативними порівняно з навчанням за Лапласом. Poisson Learning є ефективним і простим у застосуванні, а численні експерименти

демонструють його перевагу над іншими сучасними підходами напівкерованого навчання на таких наборах даних, як MNIST, FashionMNIST і CIFAR-10. Крім того, запропоновано вдосконалену модифікацію Poisson Learning, яка називається Poisson MBO, що забезпечує вищу точність і дозволяє заздалегідь знати розміри класів. Диференціальні рівняння, включаючи рівняння Лапласа, p-Лапласіана, Ейконала і Пуассона, відіграють вирішальну роль у варіаційному підході до напівкерованого навчання на графах. Рівняння Пуассона особливо корисне, коли кількість маркованих даних дуже мала.

Напівкеровані методи, засновані на поширенні міток із використанням дискретних диференціальних рівнянь в частинних похідних, є потужним математичним інструментом для розв'язання задач класифікації. Однак навіть при використанні цих підходів виникають проблеми, коли вони не забезпечують необхідних результатів навчання. До таких випадків належать зашумленість даних, розрідженість даних, перекриття класів тощо. Використання підходів регуляризації розглянуто в [1–7, 17–26] і дозволяє частково вирішити ці проблеми.

Питання регуляризації детально розглянуто в [1], [5]. Фактично, багато класичних методів GSSL можна звести до пошуку функції помилки на графі, яка повинна задовольняти двом критеріям: бути якомога ближчою до заданих міток і бути гладкою на всьому графі. Ці дві умови можуть бути виражені в загальній регуляризаційній структурі, де функція втрат розкладається на дві частини: контрольоване обмеження втрат та обмеження втрат при регуляризації графа.

Одним із маловивчених методів напівкерованого навчання є метод множинної регуляризації, який успішно використовується для розв'язання практичних задач, в яких налаштування параметрів є складним завданням [19–23]. Наприклад, у роботі [19] автори розглядають задачу, в якій сигнал і шум адитивно комбінуються. Для її розв'язання вони використовують метод адаптивного підбору параметрів для множинної регуляризації. У [20] метод множинної регуляризації використовується для знаходження оптимальної швидкості навчання нейронної мережі.

У статті [21] розглянуто нову схему регуляризації для відновлення розв'язку лінійного некоректно поставленого операторного рівняння за зашумленими даними в гільбертовому просторі. У [22] обговорюється регуляризація з множинними штрафами, яка була успішно використана для розв'язання невизначених розріджених регресійних задач незмішаного типу. Перевага методів із множинними штрафами полягає в тому, що будь-яка апріорна інформація може бути включена в додаткові штрафи. Наприклад, у роботі [23] для розв'язання задачі екстраполяції точки прогнозу включаються апріорні знання при побудові екстраполяційної оцінки.

Варто також відзначити новітні дослідження в області напівкерованого навчання на графах, які зосереджені на використанні глибоких нейронних мереж та графових нейронних мереж (GNN). У роботі [27] автори запропонували Graph Convolutional Networks (GCN) для напівкерованого навчання на графах, що стало проривом у цій галузі. Подальші дослідження [28–30] розвивають цю ідею та пропонують різні архітектури GNN для покращення продуктивності.

Зокрема, у роботі [31] було запропоновано Graph Attention Networks (GAT), які використовують механізм уваги для присвоєння різних ваг сусіднім вершинам у графі, що дозволяє моделі фокусуватися на більш релевантних зв'язках. У [32] представлено метод Graph Mixup для покращення узагальнювальної здатності моделей шляхом інтерполяції між вузлами в латентному просторі.

Крім того, в останні роки набули популярності підходи, що поєднують напівкероване навчання з методами контрастивного навчання [33]. Ці методи використовують немарковані дані для навчання моделей, що можуть ефективно витягувати значущі представлення даних.

Таким чином, розвиток напівкерованого навчання на графах є активно досліджуваною областю, і багато сучасних робіт зосереджуються на поєднанні традиційних методів з новітніми підходами глибокого навчання та регулярізації.

1.2 Повністю контрольоване навчання на основі графу

Повністю контрольований алгоритм навчання це випадок коли використовуються всі мітки при побудові класифікатора. Наприклад, можна застосовувати метод-найближчих сусідів KNN. Метод найближчого сусіда працює досить добре, коли дані групуються навколо якихось центрів, а зв’язків між класами мало. Для цього випадку для поширення міток застосовують методи напівконтрольованого навчання.

1.3 Методи напівкерованого навчання на основі графів

У цьому пункті буде розглянуті основні положення та поняття про напікероване навчання та напівкерованного навчання на основі графів. Проаналізовані проблеми застосування навчання по Лапласу та Пуассону.

1.3.1 Загальний підхід до напівконтрольованого навчання.

Графові методи напівконтрольованого навчання (Graph-Based Semi-Supervised Learning, GSSL) мають трансіндуктивну та індуктивну основу. Основна відмінність від повністю контрольоване навчання – часткове залучення розмічених даних, можливо навіть невеликої кількості міток. Індуктивний підхід передбачає послідовне виконання алгоритму напівкерованого навчання: побудова графу, вибір початкових міток, алгоритм розповсюдження міток, правило вибору остаточних значень міток, врахування шуму, оптимізація та масштабування.

Напівкероване навчання на основі графів (GSSL) стикається з кількома ключовими проблемами на різних етапах. Перелічимо нижче деякі з недоліків, та методи для їх компенсації або усунення.

Етап побудови графу. Побудова відповідного графу, що відображає основні взаємозв'язки між точками даних, має вирішальне значення. Були запропоновані різні методи, такі як k-ближайші сусіди, ϵ-найближчі сусіди, Лапласіан на графі та інші підходи. Ці методи спрямовані на визначення локальної або глобальної структур і зв'язку даних. Деякі з них перелічені у роботі [5] [Z. Song].

Етап розповсюдження міток. Основна мета GSSL — розповсюдження значень міток в умовах перетину класів (у наборі даних з дуже малою кількістю маркованих вузлів) [1] [Xiaojin (Jerry) Zhu]. Однак інформація, на основі якої мітки отримують своє значення може бути невірною, що в свою чергу призводить до отримання невірних значень міток, що погіршує результати класифікації. Для вирішення цієї проблеми було розроблено різні алгоритми: дифузія міток, тощо. **[5] [Z. Song].** Ці алгоритми використовують структуру графа для формування міток, враховуючи при цьому збіг між точками даних.

Етап масштабування. Масштабування методів GSSL для наборів даних, розмір яких постійно змінюється, може вимагати більших витрат часу, фінансових та людських ресурсів. Для оптимізації цих витрат при масштабованості використовуються методи апроксимації, паралельні обчислення. Деякі підходи, такі як кластеризація графів і подрібнення графів, спрямовані на зменшення розміру і складності графа без додаткової інформації. [Anton Tsitsulin].

Етап обробки обмеженої кількості розмічених даних: GSSL прагне використовувати як розмічені, так і нерозмічені дані. Однак, коли розмічених даних недостатньо, можна використовувати такі методи, як самонавчання, спільне навчання та активне навчання [5] [Z. Song,]. Ці методи ітеративно відбирають інформативні зразки для маркування або використовують декілька представлених даних для покращення процесу навчання.

Стійкість до шуму. Шум і викиди в даних можуть суттєво впливати на продуктивність методів GSSL. Надійні заходи сходження, та алгоритми виявлення викидів використовуються для пом'якшення впливу зашумлених точок даних на результати класифікації[11] Y. Chong].

В цілому можна зробити наступний висновок:

по-перше – методи напівкерованого навчання є достатньо потужними для вирішення багатьох задач класифікації із різними початковими умовами,

по-друге – найбільш продуктивним методами напівкерованого навчання є методи розповсюдження міток на основі ітеративного розв'язку диференціальних рівнянь у часткових похідних (Лапласа та Пуассона);

по–третє – для складних випадків даних застосовуються методи регулярізації та множинної регулярізації.

В цій роботі буде досліджено навчання нейронної мережі при вирішенні задачі класифікації на основі алгоритму рівняння Пуассона з застосуванням регуляризації L2. Далі сформулюємо постановку задачі з урахуванням розв'язку існуючих недоліків інших методів.

1.4 Навчання по Лапласу

1.4.1 Постановка задачі напівкерованного навчання на основі графа

Для множини вершин $X = x_1, x_2, \dots x_n$ деякого графу G із наступними умовами маємо $W = (w_{ij})_{i,j=1}^{n}$ - матриця ваг графу, яка складається з складених на попередньому шляху ваг ребер. W додатна та симетрична. $w_{ij} \approx 1$, якщо вузли x_i та x_j "схожі"; $w_{ij} \approx 0$ якщо вузли x_i та x_j "не схожі". Чим більше співпадають

значення двох вузлів тим більш вони “схожі”, тобто використовується припущення про гладкість.

Ступінь вузла визначаємо за формулою $d_i = \sum_{j=1}^{n} w_{ij}$. Для випадку багато класової класифікації із k класів введемо вектор $e_i \in \mathbb{R}^k$, що відображатиме і-тий клас. Перші m вершин множини X отримують відповідні мітки $\{y_1, y_2, \dots y_n\}$ із множини $\{e_1, e_2, \dots e_k\}$. Кількість мічених даних менша за загальну, тобто m < n. Задачею напівкерованого навчання із застосуванням графу є поширення міток від стартових вершин до усіх інших, тобто: x_{m+1}, x_{m+2},..., x_n.

Маємо набір вершин і множину ребер неорієнтованого графа, вагова матриця якого симетрична. Ваги ребер близькі до одиниці, якщо вершини з’єднані ребром і рівні нулю, якщо не пов’язані ребром. Тому матриця ваг – це досить велика розріджена матриця, що складається з нулів і одиниць.

Мітки встановлюються для перших m вершин. Загалом, це вектори, які вказують на приналежність до класу, якщо дано k класів. Задано мітки $\{y_1, y_2, \dots y_m\} \in \{e_1, e_2, \dots e_k\} \in R^k$. Тому базисними векторами e_i називаються гарячі вектори з нулями одиниць, які характеризують приналежність до класу. Необхідно розширити мітки на інші вершини, слідуючи якомусь закону.

Напівконтрольоване навчання на основі графів називається навчання в якому граф будується на основі позначених і нерозмічених даних, які використовують геометричні властивості графа або різних інших структур. Мета напівконтрольованого навчання на основі графів – отримати хорошу продуктивність класифікатора, використовуючи набагато менш марковані дані.

1.4.2 Рішення за допомогою регулярізації Лапласа

Єдиного рішення проблеми поширення міток на графу не існує. Одним з популярних методів вирішення проблеми регулярізації є регулярізація за

допомогою рівняння Лапласа. Регуляризація Лапласа означає, що у вас є мітки на перших m значеннях, а потім ці мітки поширюються далі, використовуючи гармонічну функцію Лапласа, описану рівняннями Лапласа. Це метод вперше запропонувалили [Zhu, Zhou, Ando, Xu].

Щоб розповсюдити мітки на графу, можна отримати u з рівняння

$$Lu(x_i) = \sum_{j=1}^{n} w_{ij} \left(u(x_i) - u\left(x_j\right) \right) = 0 \quad (\text{m+1=< i=<n})$$

$$u(x_i) = \frac{\sum_{j=1}^{n} w_{ij} \{u\left(x_j\right)\}}{\sum_{j=1}^{n} w_{ij} = d}$$

Тобто функція u буде гармонійною і задовольняти середньому значенню на графу і це середньозважена величина своїх сусідів. Це поширення мітки як показано в [Gju 2005], ітеративно замінюючи на u^{k+1} та u^{k}

$$u(x_i)^{k+1} = \frac{\sum_{j=1}^{n} w_{ij}\, u\left(x_j\right)^{k}}{\sum_{j=1}^{n} w_{ij} = di}$$

Коли ця ітерація сходиться, отримуємо точний розв’язок рівняння Лапласа і на кожному кроці встановлюємо мітки в відомих точках. Якщо говорити про роздачу міток, то в більшості випадків маємо на увазі роздачу міток по Лапласу. Математично це метод Якобі для вирішення рівняння Лапласа і це один з найповільніших методів, який можна використовувати.

1.4.3 Основні проблеми навчання по Лапласу

Розглянемо навчання Лапласа.

Задачі вирішення рівняння Лапласа – ітеративним методом Якобі, одним з найповільніших методів навчання слабо застосовуються до вибірки з великою кількістю міток. Виродження у нулі – це погана точність навчання при відборі проб з невеликою кількістю міток, що фактично означає пошук рішення з точки, близької до нуля. Однак можливо використовувати ітерацію спряжених градієнтів для розв'язання рівняння Лапласа для прискорення. Тренування Лапласа має кількісну варіаційну формулювання.

Варіаційною проблемою навчання по Лапласу є мінімізація енергії Діріхле. Енергія Діріхле зводиться до мінімуму з урахуванням того, що мітки збігаються в заданих вузлах. При цьому необхідно, щоб виконувалася гладкість виконання функції. Для цього вводиться штраф за негладкість, як, тренування Лапласа з м'якими обмеженнями. Також можна використовувати інші регуляризатори, які краще підходять для конкретного прикладу.

Тренування Лапласа досить добре працює тільки для помірних обсягів розмічених даних. Якщо використовувати тренування Лапласа для великого обсягу розмічених даних, то отримаємо результати, схожі на випадкове вгадування (випадкова ходьба – випадкове блукання).

При цьому рішення буде постійним, тобто постійна функція має енергію Діріхле, рівну нулю (Nadler 2009, 2011). Постійні функції підходять для спрямованих енергій. Вартість цих міток вже не така велика. Тобто краще взяти постійну функцію, ніж плавно інтерполювати з одного значення функції в інше. Випадкове вгадування залежить від вибірки.

1.5 Навчання по Пуассону

Розглянемо навчання Лапласа і його зв’язок з навчанням Пуассона

Фізичне трактування рівняння Пуассона – стаціонарний стан для рівняння теплопровідності, зокрема, вихідний член в правій частині повинен мати середнє значення, щоб зміна тепла дорівнювало нулю. Щоб було стаціонарне рішення проблеми. Отже, роль центрування на осі у означає, що член у правій частині має нульове середнє. Це дозволяє знайти розв’язок цього рівняння та єдності (з нетривіальністю ядра графа Лапласа).

По суті, різниця між навчанням за Лапласом і навчанням за Пуассоном полягає у позначках у вузлах: перший метод фіксує позначки вузлів, а другий фіксує гладкість позначок у вузлах до певних констант.

Заміняємо рівняння Лапласа, рівнянням Пуассона

$$L(u_i) = \sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij}\,,\ 1 \le i \le m$$

$$L(u_i) = 0, \qquad m+1 \le i \le n$$

Отже, якщо центруюча роль така, що член у правій частині повинен мати середнє значення, рівне нулю графом, це вказує на те, що умови (існування та унікальність розв’язку) унікальності лапласіанського ядра задоволені; нижня умова означає, що середньозважене дорівнює нулю, зваженому за ступенем;

Рішення те ж саме, але в тренуванні Пуассона можна врахувати незбалансовані розміри класів або різну кількість навчальних прикладів в кожному класі, просто перезваживши рішення Про метку, подібну до джерел точок повторного зважування Щоб зробити їх важчими на заняттях з меншою кількістю балів:

1.6 Порівняння навчання по Пуассону та по Лапласу

Проведемо порівняння по Пуасону та по Лапласу.

Одним з найбільш широко використовуваних методів напівконтрольованого навчання на основі графів є навчання за Лапласом. Однак навчання за Лапласом може давати дуже погані результати класифікації [**Nadler et al.]; [El Alaoui et al.]** для невеликої кількості міток.

Це часто пояснюється тим, що рішення розвивають локалізовані спайки поблизу зазначених точок і майже постійні далеко від зазначених точок. Зокрема, позначки значень погано поширюються лапласіанським підходом до навчання. Для вирішення цієї проблеми було запропоновано p-лапласове навчання (**El Alaoui et al.).** Деякі роботи ретельно вивчали регуляризацію за Лапласом з декількома мітками **(Slepcev & Thorpe; Calder)**, і нещодавні чисельні результати показують, що $p > 2$ краще до навчання за Лапласом при низьких показниках міток (Flores et al.). Випадок $p = \infty$ називається ліпшицевим навчанням **(Kyng et al.**), який шукає абсолютно мінімальне ліпшициве розширення навчальних даних.

Інші методи боротьби з проблемою низької швидкості розповсюдження міток включають лапласове регулювання більш високого **порядку (Zhou & Belkin)** та спектральні обмеження **(Belkin).**

Деякі недавні підходи були спрямовані на повторне зважування графа, який більш щільно примикає до міток, щоб надати їм більшого впливу, коли швидкість поширення міток дуже низька. В один бік для повторного зважування графіка є зваженим нелокальним лапласіаном (WNLL) **(Shi та інш.),** що збільшує ваги ребер безпосередньо пов'язаних з поміченими вузлами. WNLL досягає кращих результатів при помірно низькому рівні міток, але погано працює при дуже низьких показниках міток **(Flores et al)**. Щоб вирішити цю проблему, **(Calder & Slepcev)** запропонували правильно зважений лапласіан, який повторно зважує графік способом, добре поставленим при довільно низьких ставках міток.

Пуасонове навчання - це напівкерований підхід до навчання на основі графа, що базується на розв’язку відповідного рівняння. Рівняння Пуассона – це диференціальне рівняння у приватних похідних, яке часто використовується у фізиці та техніці для моделювання фізичних явищ, таких як електростатика, потік

рідини та теплопередача. У контексті напівконтрольованого навчання з урахуванням графа рівняння Пуассона використовується поширення міток від мічених вузлів до немічених на графі.

Ідея полягає в тому, щоб розглядати лапласіан графа як оператор, який перетворює значення міток у вузлах, а потім вирішувати рівняння Пуассона, щоб знайти гладке рішення, що задовольняє граничним умовам, що накладаються позначеними вузлами.

У порівнянні із Лапласовим навчанням, особливості рівняння Пуасону дозволяють краще вирішувати задачі напівкерованого навчання на основі графів. Основною перевагою навчання Пуасона є його точність роботи при дуже незначній кількості розмічених даних. Це пов’язано у першу чергу із тим, що через неоднорідну форму

Загальна мета напівконтрольованого навчання - отримання хорошої продуктивності класифікатора, використовуючи набагато менш марковані дані.. Єдиного універсального рішення проблеми поширення міток на графу не існує. Навчання по Лапласу та Пуассону припускає що поширення міток відбувається за формулою гармонічної функції Лапласа або Пуассона. Однак ці методи мають наступні недоліки. та полягають у наступних площинах.

1) Перша площина пов’язана з можливостями застосування математичних моделей рівняння Лапласа та рівняння Пуасона.
2) Друга площина пов’язана с застосуванням ймовірностей підходів.
3) Третья площина пов’язана зі застосуванням чисельних методів.
4) Четверта площина пов’язана з взаємовідносинами між досліджуємими класами.
5) П’ята площина пов’язана з питанням перенавчання.

Взагалі методи вирішення цих рівнянь базуються на ітераційних процесах та потребують контролювання питання сходження до аналітичних рішень.

Найбільш відомим ітераційним методом для вирішення рівняння Лапласу є метод Якобі. Однак він є найповільнішим методом і не може застосовуватися до

великих даних з цього приводу. Також відома проблема методу Якобі це виродження у нулі. Це означає що метод навчання по Лапласу не працює для невеликої кількості початкових розмічених даних. Застосування ітераційного методу спряжених векторів для вирішення рівняння Лапласу обмежене тим що він буде уповільнюватися при мітках які будуть належати до одного класу. Тобто якщо дуже довго розмічення буде відповідати до одного класу, то ймовірність того що наступна мітка буде також буде відповідати до цього класу зростає.

Часто для вирішення рівняння Лапласу застосовуються ймовірності методи основані на фізичних та інших явищах. Це випадкове блукання, броунівський рух, дифузія, інваріантні потоки Маркова та інш. Застосування ймовірнісних методів має наступні недоліки. Основний з яких – втрата (забування) навчальної інформації (мітки) після деякої кількості ітерацій. Другий – можливість не розмітити усі нерозмічені дані. Також ці ймовірності методи погано підходять для класів, які мають короткий зв'язок і не підходять для класів, які мають широкий перетин. Та взагалі ці ймовірності процеси займають багато часу та потребують усереднення та однорідного розподілу даних по класам. Таким чином можливість застосування навчання по Лапласу дуже обмежена.

Диференціальні методи вирішення рівня Пуассону приводять до ітераційних процесів. Ітераційні методи рівняння Пуассону менш сприйнятливі до процесу забування початкової інформації (начальних розмічених даних). Також в тренуванні Пуассона можливо врахувати незбалансовані розміри класів або різну кількість навчальних даних в кожному класі, просто переваживши рішення.

Розглядаючи питання збіжності ітераційних диференціальних методів слід відзначити що для задачі великої кількості даних тим краще чим скоріше збіжність. Тому у подальшому потрібно розглядати диференціальні методи які будуть збігатися скоріше. Це методи з акумулюванням довжини кроку (з моментами) та адаптивні методи (наприклад, Nesterov, Adam, Adagrad).

До навчання Пуасона теж можна застосувати випадкове блукання. Застосування випадкового блукання буде деяке інакше ніж до навчання Лапласа, але проблеми подібні: швидкість та забування.

Навчання за Пуассоном буде ліпше ніж навчання за Лапласом для невеликої кількості початкових розмічених вершин. Але при збільшенні початкових даних може наступити перенавчання.

Виріши проблему перенавчання можливо за допомогою різноманітних регуляризаторів. Регуляризуючи добавки можна поділити на функціональні та статистичні. Функціональні добавки впливають на гнучкість та гладкість графу навчання. Та базуються вариаційному принципі диференціальна модель рівнянь Лапласа та Пуассона з умовою Діріхле в межах області та граничною умови Неймана на кордоні всієї області.

За допомогою статистичних регуляризаторів можливо враховувати розподіл мічених та немічених по класам, щільність розподілу даних у кожному класі, обмеження обсягу даних та інші питання. Існує також багато алгорітмичних підходів таких як переважування графів, використування регуляризаторів вищих порядків, застосування центрованих матриць.

Однак всі ці методи не є універсальними та потребують більш ретельних досліджень.

1.7 Практичне застосування GSSL

Напівкероване навчання на основі графів має велике практичне застосування. Розгляне декілька таких робіт.

У роботі **[19]** надається вичерпний огляд найсучасніших методів навчання на основі графів для класифікації гіперспектральних зображень (HSI), включаючи напівкеровану класифікацію на основі спектральної інформації на основі графів і

напівконтрольовану класифікацію на основі спектрально-просторової інформації. -контрольована класифікація. Крім того, пов’язані методи класифікуються на наступні підтипи: (1) Графове напівконтрольоване навчання на основі різноманіття для класифікації HSI (2) Графове напівконтрольоване навчання на основі розрідженого представлення для класифікації HSI. Для кожної техніки обговорюються методології, зразки навчання та тестування, різні технічні труднощі, а також характеристики. Крім того, вказані майбутні дослідницькі проблеми, пов’язані з моделлю на основі графіків.

У **[20]** досліджуються питання діагностування за допомогою методів напівкерованого навчання. У діагностиці раку значна кількість даних отримується під час планових досліджень. Нещодавно машинне навчання було використано для створення класифікаторів, завданням яких є виявлення раку та допомога в прийнятті клінічних рішень. Більшість цих класифікаторів базується на навчанні під наглядом (SL), яке потребує витрат часу та витрат на ручне маркування зразків медичними експертами для навчання моделі. Напівконтрольоване навчання (SSL), однак, працює лише з частиною мічених даних, включаючи немічені зразки для абстракції інформації, і, таким чином, може використовувати величезну розбіжність між доступними міченими даними та загальними доступними даними в діагностиці раку. У цьому огляді надано вичерпний огляд основних функціональних можливостей і припущень SSL, а також огляд ключових досліджень щодо лікування онкологічних захворювань, розрізняючи додатки на основі зображень і без них. В роботі висвітлюєтеся поточні сучасні моделі в гістопатології, радіології та радіотерапії, а також геноміці. Крім того, обговорено потенційні підводні камені в дизайні дослідження SSL, такі як розбіжності в розподілі даних і порівняння з базовими моделями SL, і вказуємо на майбутні напрямки для SSL в онкології. Ми вважаємо, що добре розроблені моделі SSL роблять значний внесок у комп’ютерну діагностику злоякісних захворювань, долаючи поточні перешкоди у вигляді рідкісних мічених і великої кількості немічених даних.

1.8 Висновки до розділу 1

У розділі проаналізовано сучасний стан проблеми напівкерованого навчання на основі графів. Проведено аналіз літератури та методів напівкерованного навчання. Показано переваги напівкерованого навчання на основі графового підходу, яке застосовує рівняння Пуассону.

А саме, у розділі проаналізовано метод навчання по Лапласу та метод навчання по Пуассону та деякі їхні модифікації. Для вирішення розв’язку рівняння Пуассона пропонується застосовувати варіаційний підхід для пошуку мінімуму функції та наступні диференціальні варіації оптимізаційних методів. Ймовірності методи дуже залежать від розподілу вибірки розмічених міток. Градієнтні методи менш залежать від розподілу вибірки розмічених даних. Деякі з цих градієнтних та ймовірнісних методів будемо розглядати та порівнювати між собою у наступному розділі.

Важливим кроком є використання додакових регуляризаторів особливо класичних регуляризатора Тихонова. Деякі з них будуть застосовані далі.

РОЗДІЛ 2 РОЗРОБКА УЗАГАЛЬНЕНОГО МЕТОДУ НАПІВКЕРОВАННОГО НАВЧАННЯ НА ОСНОВІ ГРАФІВ

2.1 Постановка задачі класифікації

Нехай задана множина даних X та множина міток Y та на початку процесу класифікації мітки будуть задані тільки у частині даних $X_0 \subset X$, $Y_0 \subset Y$. Класифікувати дані - це означає встановити мітки даним, що залишилися, з множини X. Якщо потрібно провести класифікацію даних за допомогою методу напівкерованого навчання на основі графів GSSL то спочатку будуть граф звязків даних, а потім застосовують ітераційний алгоритм розповсюдження значень міток на всі дані множині.

Загальний підхід напівкерованного навчання на основі зваженого для вирішення задачі классифікації складається з наступних етапів

1) Побудова зваженого графу
2) Побудова функції похибки
3) Побудова функції розповсюдження міток
4) Задання розмічених вершин
5) Ітераційне рішення рівняння (ітераційний алгоритм) розповсюдження міток
6) Зупинка алгоритму та вибір класифікаційних значень міток

В даній магістерський дисертації потрібно розробити новий алгоритм розповсюдження міток на графі, який буде застосовувати диференціальні методи GSSL основані на ітераційному розв’язку узагальненого рівняння дифузії на графі. На основі запропонованого підходу розробити інтелектуальну систему классифікації. В даному розділі розробимо узагальнений метод напівкерованного навчання на основі застосування Лапласіану.

2.2 Побудова зваженого графу

На першому кроці потрібно побудувати граф (даних), де вершини будуть відзначати дані, а ребра зв’язки між вершинами. Якщо не задано ваги ребер, то їх частіше розраховують за наступними формулами: геометричні ваги, Гаусови ваги, KNN ваги (їх можна вважати відстанню між вершинами).

$$w_{ij} = \eta\left(\frac{|x_i - x_j|}{\varepsilon}\right) \text{– геометричні ваги,}$$

$$w_{ij} = \eta\left(\frac{|x_i - x_j|}{\varepsilon_k(x_i)}\right) \text{– KNN ваги,} \qquad (2.1)$$

$$w_{ij} = exp\left(\frac{-|x_i - x_j|^2}{2\sigma^2}\right) \text{– Гаусови ваги}$$

Де $X = \{x_1, x_2, \ldots, x_n\}$ - множина вершин графа, η - функція, ε – параметр сусідів, σ параметр, який контролює дисперсію сусідів

На малюнку нижче приведений приклад побудови графу за допомогою алгоритму KNN – алгоритму найближчих сусідів з параметрами knn=5, knn=10. Тобто з кожної вершини буде виходити 5 або 10 ребер відповідно. Фактично ми зменшуємо кількість сусідів до заданого числа та суттєво знижуємо складність задачі.

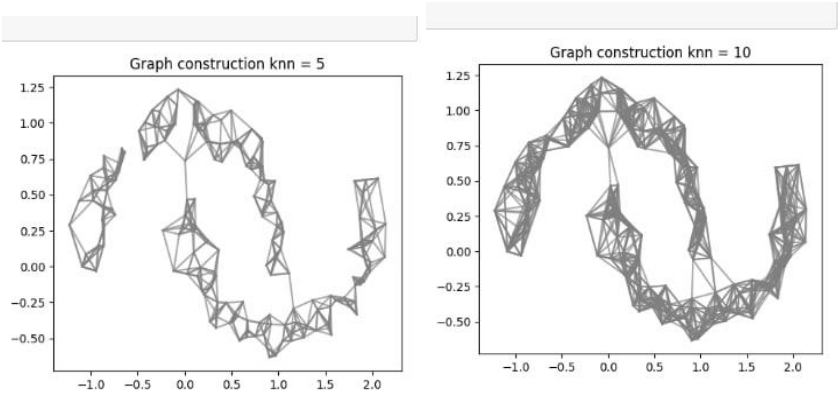

Рис. 2.1. Побудова графу при knn=5, knn=10

Існує проблема вибору значення к при побудові KNN графу. Маленькі значення приводить до появи компонент незв’язності., а велике ускладнює кількість операцій алгоритму розповсюдження міток. Взагалі теоретично задачу класифікації можливо вирішити за допомогою класифікатору, але практично це не можливо оскільки - це np повна задача. Метод KNN краще враховує структуру множини даних.

Метод Гаусових ваг дозволяє визначати ваги згідно Гаусового розподілу та найближчих сусідів, не обмежуючи кількість сусідів. Цей метод більш підходить до множин, які краще груповані. Коли розподіл даних невідомий – краще застосовувати KNN метод.

2.3 Функція похибки та рівняння розповсюдження міток на основі принципу дифузії

Методи GSSL працюють на спеціально розробленому графі, в якому навчальні вибірки представлені у вигляді вузлів, а кожна пара вузлів пов’язана вагою для позначення основної схожості. Деякі вузли позначені, інші - ні. В результати необхідно побудувати граф, щоб зробити ці задачі піддатливими до наступних підходів GSSL. Однак варто зазначити, що більшість алгоритмів на основі графів призначені для кроку виведення міток. Після побудови графа наступним кроком у вирішенні проблеми SSL за допомогою графових методів є введення позначених даних на підмножину вершин графа з подальшим виведенням міток для не позначених вершин. Хоча більшість підходів до виведення на основі графів є трансдуктивними, існують також деякі індуктивні підходи до GSSL. Відповідно до структури SSL, функція втрат підходів GSSL також може бути узагальнена в рамках функції втрат SSL, яка складається з трьох частин, як показано в рівнянні. (2.2)

$$L(f) = \underbrace{\mathcal{L}_s(f, \mathcal{D}_\ell)}_{\textit{supervised loss}} + \underbrace{\mathcal{L}_u(f, \mathcal{D}_u)}_{\textit{unsupervised loss}} + \mu \underbrace{\mathcal{L}_r(f, \mathcal{D})}_{\textit{regularization loss}} \quad (2.2)$$

де L_s (f, D_l) це контрольована втрата на маркованих даних, а L_u (f, D_u) це неконтрольована втрата немаркованих даних та $L_r(f, D)$ є втратою регуляризації. Крім того, λ та μ є гіперпараметрами для балансування. Однак для GSSL неконтрольовані втрати часто поглинаються втратами при регуляризації, оскільки в частині втрат при регуляризації не використовується інформація про мітки. Тому функцію втрат для GSSL можна узагальнити так, як показано в рівнянні.

$$\mathcal{L}(\mathrm{f}) = \mathcal{L}_s(\mathrm{f}, \mathcal{D}_\ell) + \mu \mathcal{L}_r(\mathrm{f}, \mathcal{D}) \quad (2.3)$$

Як відомо рівняння розповсюдження міток можна отримати за рахунок вирішення задачі оптимізації (мінімізації) функції похибки. Якщо потрібне ітераційне рішення, то можна застосовувати диференціальні рівняння дифузії та які основані на понятті Лапласіану.

2.4 Навчання на основі застосування лапсасіану

Існує досить велика група методів напів-контрольованого навчання на основі графу використовує так званий Лапласіан - матрицю, що відображає суть різноманітної структури графу даних.

Нехай заданий зважений граф G, де x_i– вершини графа, v_i– ребра графа, $w_{i,j}$ – вага ребра графа (x_i, x_j), $u_i = u(x_i)$– функція міток x_i вершини графа. $i = 1 \dots n$, n – кількість вершин графа.

Далі будемо розглядати наступні види оператора Лапласа (лапласіану)

Ненормалізованій лапласіан L_{un} та нормалізованій лапласіан L_n

$$L = L_{un} = D - W = D\,(I - D^{-1}W) \;=\; L_{un}u(x_i) = \sum\nolimits_{j=1} w_{ij}\left(u(x_i) - u(x_j)\right) \qquad (2.4)$$

$$L_n = D^{-\frac{1}{2}}L_{un}\,D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}W\,D^{-\frac{1}{2}} = \; L_n u(x_i) = \sum\nolimits_{j=1} \frac{1}{\sqrt{d_i d_j}} w_{ij}\left(u(x_i) - u(x_j)\right) \qquad (2.5)$$

де $W = \left(w_{ij}\right)_{i,\,j=1}^{n}$ – матриця ваг ребр графа (x_i, x_j),

D - матриця вага ребра діагональних елементів графа (x_i, x_i)

Найбільш відомо застосування Лапласіану у так званому - навчанні по Лапласу та навчанні по Пуассону – **Label Propagation; Label Spreading; Poisson Learning.** Ці методи ми будемо розглядати детальніше далі.

2.5 Навчання за Лапласом – Label Propagation

Розглянемо навчання по Лапласу (Zhu and Ghahramani; Zhu et al.).. Запишемо для графа G енергію Діріхле, вважаючи що $u_i = u(x_i)$ – функція міток x_i вершини графа.

$$E(u) = \frac{1}{2}\boldsymbol{u}^T L\boldsymbol{u} = \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}\left(u(x_i) - u(x_j)\right)^2 \qquad (2.6)$$

L – ненормований оператор Лапласа, $u = (u(x_1), u(x_2), \dots, u(x_n)) = (u_1, u_2, \dots u_n)$ - вектор функції міток вершин, $y = (y(x_1), y(x_2), \dots, y(x_n)) = (y_1, y_2, \dots y_n)$ - вектор функція початкових міток, так званої «гарячий» вектор.

Тоді завдання Лапласового навчання може бути подане як завдання мінімізації енергії Діріхле $E(u)$ на графі G

$$\min E(u) = min\frac{1}{2}u^T Lu = min\frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}\left(u(x_i) - u\left(x_j\right)\right)^2 \quad (2.7)$$

Завдання мінімізації функціоналу (2.7) приводить до вирішення рівняння Лапласа з початковими умовами

$$Lu = 0, \quad (2.8)$$

$$u = y$$

Або

$$\sum_{j=1}^{n} w_{i,j}\left[u(x_i) - u\left(x_j\right)\right] = 0,\ i = m+1..n, \quad (2.9)$$

$$u(x_i) = y(x_i),\ i = 1..m$$

Де $u_i = y_i \in \{0,1\},\ i = 1,2,\dots,m$ - задані початкові мітки для m вершин.

Для інших $n - m$ вершин значення функції $u_i\ i = m+1,\dots,n$ потрібно визначити ітеративно.

Існує багато різних підходів для вирішення рівняння (2.9). Один з яких називається метод Якобі. Метод Якобі призводить до ітераційного рішення для рівняння (2.7). У теорії напівнавчання цей метод приводить до рівнянь

$$u^{\{k+1\}}(x_i) = \frac{1}{\sum_j w_{i,j}}\sum_{j=1}^{n} w_{i,j}u^k\left(x_j\right) \quad (2.10)$$

$$u^0(x_i) = y_i,\ \ i = 1..m$$

Вважаючи $d_i = \sum_j w_{i,j}$ запишемо (2.10) у вигляді

$$u^{\{k+1\}}(x_i) = \frac{1}{d_i}\sum_{j=1}^{n} w_{i,j}u^k(x_j) \tag{2.11}$$

$$u^0(x_i) = y_i,\ i = 1..m$$

Тут $u^{\{k+1\}}(x_i) - k + 1$ ітерація функції мітки вершини x_i, $u^0(x_i)$- початкові значення функції мітки.

Формула (2.11) реалізує алгоритм, так званого випадково го блукання **Random Walk** на графі та прагне до стаціонарного Марківського потоку. Вона також описує рівняння дифузії на графі. Після перетворень формула може бути подана у вигляді **дискретного рішення рівнянь дифузії на графі.** Таким чином ми отримали **представлення функції міток** для навчання по Лапласу.

Правило вибору мітки виглядає так:

$$\ell(\mathrm{x_i}) = \underset{\mathrm{j} \in \{1,\ \ldots, m\}}{\operatorname{argmax}} \{\mathrm{u_j(x)}\} \tag{2.12}$$

Формула (2.3) також може бути практично представлена у вигляді градієнтного спуску та для неї можуть бути застосовані формули прискореного градієнту, Нестерова, Адама. Тобто за рахунок цих оптимізаційних формул може бути збільшення швидкість збіжності рішення, а формула (2.4) має верхню межу швидкості збіжності і не може бути збільшена більше її передільного значення.

Таким чином алгоритм може бути представлений у вигляді

Алгоритм **–Label Propagation (Random Walk)**

Input: $W, Y^0 = [y_1, y_2, \ldots, y_m,\ \ 0, 0, \ldots 0], T$

1. $D = diag\ (W\mathbf{1})$
2. $L_{un} = D - W$
3. $U^0 = Y^0$

4. $\boldsymbol{For}\ k\ =\ 1\ \boldsymbol{to}\ T$:
5. $\boldsymbol{U^{k+1} = D^{-1}(WU^{k})}$
6. $l_k = arg\,max_{1\leq j\leq 2}\ \ U^k{}_j$

Output: $l = [l_1, l_2, \ldots, l_n]$

2.6 Навчання із застосуванням рівняння Пуассону

У нещодавніх роботах **[Jeff Calder]** було запропоновано новий метод на основі Лапласіану - **Poisson Learning**. Цей метод вирішує деякі проблеми навчання Лапласа, зокрема погане навчання з малою кількістю мічених даних. Однак за іншими параметрами воно схоже з **Label Propagation**.

Нехай заданий неорієнтований зважений граф $G(X, V, W)$ з n вершинами. X $= \{x_1, x_2, \ldots, x_n\}$ – множина вершин графа, V– множина ребер, $W = \left(w_{ij}\right)_{i,\,j=1}^{n}$ – матриця ваг графа G. Покладемо $w_{ij} = 1$, якщо вершини x_i, x_j подібні, и $w_{ij} = 0$, якщо вершини x_i, x_j різні. Ступінь вузла x_i визначимо по формулі $d_i = \sum_{j=1}^{n} w_{ij}$. Згідно з Пуассонівським навчанням **[Jeff Calder] розповсюдження міток відбувається за допомогою рішення** рівняння Пуассона, які мають вид **рішення** рівняння Пуассона, які мають вид

$$L(u_i) = \sum_{j=1}^{m}\left(y_j - \bar{y}\right)\delta_{ij}\,,\ 1\ \leq\ i \leq m \qquad (2.13)$$

$$L(u_i) = 0, \qquad m+1 \leq i \leq n$$

Де L – ненормований оператор Лапласа, x_i - вершини неорієнтованого зваженого графу, $y_i = y(x_i)$ – початкові мітки вершин графу, $u_i = u(x_i)$ - функція міток вершин графу, w_{ij}- вага ребра $\left(x_i, x_j\right)$, n – загальна кількість вершин графу, перші

m з яких вважаються поміченими. $\bar{y} = \frac{1}{m}\sum_{j=1}^{m} y_j$. $\delta_{i,j}$ - символ Кронекера. $\sum_{j=1}^{n} d_i u(x_i) = 0$.

$$E(u) = \frac{1}{2}\, u^T L u - \sum_j \left(y_j - \overline{y}\right)\delta_{i,j} \tag{2.14}$$

Тут L - оператор Лапласа, $\overline{y} = \frac{1}{m}\sum_j y_j$,

Правило вибору мітки :

$$l(x_i) = argmax_{j\in 1,..,k} s_j u_j(x) \tag{2.15}$$

де $s_j = \left(b_j \middle/ \bar{y}\right)$

Алгоритм **Poisson Learning**

Input: $W, F = [y_1, y_2, \dots, y_m], T$

1. $D = diag\,(W1)$
2. $L = D - W$
3. $c = \frac{1}{m} F1$
4. $B = [F - c, Z\,(2, n-m)]$
5. $U = Z\,(n, 2)$
6. $\boldsymbol{For}\ i = \mathbf{1}\ \boldsymbol{to}\ \boldsymbol{T}$:
7. $\boldsymbol{U^{k+1} = U^k + D^{-1}\left(B^T - LU^k\right)}$
8. $l_k = arg\, max_{1\le j\le 2}\ \ U^k{}_j$

Output: $l = [l_1, l_2, \dots, l_n]$

2.7 Нормалізоване навчання по Лапласу- Label Spreading

Poisson Learning. **Label Propagation** — це техніка, при якій інформація про мітки проходить через краї графа від мічених вузлів до немічених вузлів залежно від структури графа. Процес ітеративно оновлює мітки немаркованих точок на основі міток їх сусідів, доки мітки не зійдуться. Це метод простоти та інтуїтивності, який є популярним вибором для напівконтрольованого навчання. Це особливо ефективно, коли дані добре кластеризовані, а структура графіка точно відображає різноманітність даних. Але у випадках, коли ми маємо «невідповідні дані».

З іншого боку, **Label Spreading** можна розглядати як більш узагальнений підхід до напівконтрольованого навчання на графіках. Він не тільки враховує пряме поширення мітки, але також враховує щільність і розподіл даних по графу. Цей метод розповсюджує інформацію про мітки таким чином, щоб враховувати як локальну, так і глобальну структуру даних, потенційно забезпечуючи більш детальне розуміння різноманітності. **Label Spreading** може бути особливо корисним у завданнях, де розподіл даних нерівномірний або де граф даних має складну геометрію.

Розглянемо задачу напівнавчання для якої функція помилки складається з двох додатків. Нормалізованого Лапласіану. Запишемо для графа G енергію Діріхле, вважаючи що $u_i = u(x_i)$– функція міток x_i вершини графа. **Laplace Label Spreading**

$$E(u) = \frac{1}{2}(\boldsymbol{u}-\boldsymbol{y})^T(\boldsymbol{u}-\boldsymbol{y}) + \alpha\frac{1}{2}\boldsymbol{u}^T L_n \boldsymbol{u} = \frac{1}{2}\sum(u_i - y_i)^2 + \frac{1}{2}\alpha\sum_{i,j=1}^{n} w_{i,j}\left(\frac{u(x_i)}{\sqrt{d_i}} - \frac{u(x_i)}{\sqrt{d_j}}\right)^2 \quad (2.16)$$

Де α- Гіперпараметр.

Тобто Лапласова добавка виступає, як регуляризуючий член, а функція помилки - різниця квадратів, може бути отримана методом найменших квадратів. Цей метод був вперше отриманий **(Zhou et al., 2004).**

Алгоритм **– Laplace Label Spreading**

$\boldsymbol{Input}: W, Y^0 = [y_1, y_2, \dots, y_m,\ \ 0,0, \dots 0], T$

1. $D = diag\ \ (W\mathbf{1})$
2. $L\ = D - W$
3. $L_n = D^{-1/2} L D^{1/2}$
4. $U^0 = Y^0$
5 $\boldsymbol{For}\ k\ =\ 1\ \ \boldsymbol{to}\ T$:
6. $\boldsymbol{U^{k+1} = (1 - \alpha) U^0 + \alpha\, L_n U^k}$
7. $l_k = arg\, max_{1 \le j \le 2}\ \ {U^k}_j$

$\mathbf{Output}: l = [l_1, l_2, \dots, l_n]$

Переваги цього підходу

1. Гарантована хороша збіжність методу
2. Збігається до точного рішення
3. Гарантоване існування мінімуму.
4. Власні числа нормалізованої матриці Лапласа позитивні, що гарантує обчислення матриці D

Недоліки

1. Використання точного розв'язку обмежене технічними можливостями обчислення оберненої матриці, точніше її розмірами.
2. Np - повна задача
3. Для невеликої кількості початкових міток точність класифікації падає.

2.8 Регулярізація навчання за допомогою L2

Розглянемо L2 регуляризацію, яка допоможе запобігти перенавчанню, додавши до функції втрат штрафний член, пропорційний квадрату ваг. Регулюючий член має наступний вигляд:

$$P(w) = \frac{1}{2}\|w\|^2 = \frac{1}{2}\sum_i^m w_i^2 \tag{2.17}$$

Цільова функція з використанням регуляризації Тихонова матиме модифікований запис:

$$E(u) = E(u) + \frac{\alpha_1}{2}\|w\|^2 \tag{2.18}$$

Якісною відмінністю в поведінці, заголовна поведінка регуляризації L2 є розрідженість розв'язку, отриманого за допомогою регресії Лассо, тобто оптимальне значення деяких параметрів дорівнює нулю. Вище ми окреслили шляхи вдосконалення запропонованого підходу для розв'язання задачі класифікації, коли кількість маркованих даних занадто мала і класи перетинаються. Далі в рамках обраного GSSL-підходу буде запропоновано новий метод з подальшим його вдосконаленням у вигляді умов регуляризації та оптимізатора

2.9 Оптимізація навчання за рахунок градієнтного спуску, ADAM, Нестерова

Розглянемо метод напівкерованого навчання з використанням модифікованого рівняння Пуассона та додаткової регуляризації для більш ефективного застосування на незручних вибірках з трьома методами - градієнтного спуску, ADAM, Нестерова.

Розглянемо етап оптимізації та застосуємо алгоритм ADAM. Алгоритм ADAM часто використовується для оптимізації пошуку мінімуму функції помилки через його стабільну роботу. Структура алгоритму алгоритму показано нижче.

ADAM (розширення від Adaptive Moment Estimation) - один з найкращих алгоритмів оптимізації, який зазвичай використовується в глибокому навчанні для налаштування параметрів моделі під час навчання. На практиці виявилося, що він добре працює і часто перевершує інші алгоритми оптимізації в багатьох завданнях. ADAM відстежує експоненціальне зростаюче середнє значення попередніх градієнтів. Це допомагає прискорити збіжність, враховуючи попередні градієнти під час розрахунку.

Для алгоритму градієнтного спуску крок оптимізації виглядає наступним чином.

$$g_{\{k\}}(x_i) = \frac{1}{d_i}\left(\sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} - \sum_{j=1}^{n} w_{ij}\left(u_{\{k\}}(x_i) - u_{\{k\}}(x_j)\right)\right) + \alpha_1 ||u_{\{k\}}(x_i)|| \qquad (2.19)$$

$$u_{\{k+1\}}(x_i) = u_{\{k\}}(x_i) + g_{\{k\}}(x_i)$$

Для алгоритму ADAM крок оптимізації виглядає наступним чином

$$g_{\{k\}}(x_i) = \frac{1}{d_i}\left(\sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} - \sum_{j=1}^{n} w_{ij}\left(u_{\{k\}}(x_i) - u_{\{k\}}(x_j)\right)\right) + \alpha_1 ||u_{\{k\}}(x_i)||$$

$$\mathrm{m_{\{k\}}} = \beta_1 \mathrm{m_{\{k-1\}}} + (1-\beta_1)\mathrm{g_{\{k\}}},\ \overline{m_{\{k\}}} = \frac{m_{\{k\}}}{1-\beta_1^k}, \qquad (2.20)$$

$$\mathrm{v_{\{k\}}} = \beta_2 \mathrm{v_{\{k-1\}}} + (1-\beta_2)\mathrm{g_{\{k\}}^2},\quad \overline{v_{\{k\}}} = \frac{v_{\{k\}}}{1-\beta_2^k}$$

$$\mathrm{u_{\{k+1\}}(x_i)} = \mathrm{u_{\{k\}}(x_i)} - \frac{\beta \overline{\mathrm{m_{\{k\}}}}}{\sqrt{\overline{\mathrm{v_{\{k\}}}}} + \epsilon}$$

Для алгоритму Нестерова крок оптимізації виглядає наступним чином

$$g_{\{k\}}(x_i) = \frac{1}{d_i}\left(\sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} - \sum_{j=1}^{n} w_{ij}\left(u_{\{k\}}(x_i) - u_{\{k\}}(x_j)\right)\right) + \alpha_1 ||u_{\{k\}}(x_i)||$$

$$\mathrm{v_{\{k\}}} = \gamma \mathrm{v_{\{k-1\}}} - \alpha \mathrm{g_{\{k\}}}, \qquad (2.21)$$

$$\mathrm{u_{\{k+1\}}(x_i)} = \mathrm{u_{\{k\}}(x_i)} + \mathrm{v_{\{k\}}}$$

Де: $g_{\{k\}}$ - градієнт на будь-якому кроці k; $m_{\{k\}}$ - це момент імпульсу (в межах середнього градієнта); β_1 - швидкість експоненціального затягування імпульсу члена;

ADAM перемасштабує градієнт, використовуючи експоненціальне затухаюче середнє квадратів попередніх градієнтів.

$v_{\{k\}}$ - член швидкості, середній квадрат градієнта.

β_2 - швидкість експоненціального зменшення члена швидкості.

Внаслідок ініціалізації $m_{\{k\}}$ та $v_{\{k\}}$ на початку (ініціалізовані нулями), вони можуть бути зсунуті до нуля. Щоб протидіяти цьому, ADAM включає крок корекції змін. Використовуючи виправлені члени імпульсу та швидкості, параметри оновлюються: α - швидкість навчання. Вона визначає розмір кроку у параметрах простору. ε - невелика константа, яка запобігає діленню на нуль.

Метод ADAM застосовується до задачі мінімізації функції енергії E(u). ADAM дозволяє ефективно адаптувати кроки навчання для кожної змінної, що особливо корисно для задач із зашумленими або розрідженими градієнтами. Це робить процес оптимізації більш стабільним і швидшим, ніж звичайний градієнтний спуск.

2.10 Новий алгоритм - узагальнене дифузне навчання на основі графового підходу з регуляризацією

По аналогії з попередніми методами введемо - узагальнене дифузне навчання на основі графового підходу з регуляризацією - **Diffuse Label Propagation**
Задамо енергію Діріхле на графі

$$E(u) = \frac{1}{2}\sum(u_i - y_i)^2 + \frac{1}{2}\boldsymbol{\alpha}(\sum_{i,j=1}^{n} w_{i,j}\left(\frac{u(x_i)}{\sqrt{d_i}} - \frac{u(x_i)}{\sqrt{d_j}}\right)^2 - \sum_j \left(y_j - \overline{y}\right)\delta_{i,j}) \qquad (2.22)$$

Та відповідну ітераційну функцію розповсюдження міток

$$u^{\{k+1\}}(x_i) = (1-\alpha)u^{\{k\}}(x_i) + \alpha\frac{1}{d_i}\left(\sum_{j=1}^{m}(y_j - \bar{y})\delta_{ij} - \sum_{j=1}^{n} w_{i,j}\left(\frac{u(x_i)}{\sqrt{d_i}} - \frac{u(x_i)}{\sqrt{d_j}}\right)^2\right) \qquad (2.23)$$

Розробимо новий графовий метод напівконтрольованого навчання, заснований на цьому навчанні. Цей алгоритм схожий на поширення міток або дифузію на графі, де відомі мітки поширюються на немічені вузли через ітераційні оновлення. Мета — призначити мітки всім вузлам на графі на основі початкового набору відомих міток і відносин, закодованих у матриці ваг графа. Взагалі цей

підхід широко використовується в напівконтрольованому навчанні, коли лише невелика частина даних має мітки. Алгоритм складається з таких кроків.

Вхідні дані:

- W: матриця ваг, що представляє граф, де вузли — це точки даних, а ребра представляють подібність між точками. Матриця W зазвичай має розмір $(n \times n)$, де n — кількість вузлів (точок даних).
- F: матриця відомих міток, де $(F = [y_1, y_2, \dots, y_m])$. Це мітки для перших m точок даних, тоді як решта n-m точок даних не мають міток.
- T: кількість ітерацій для алгоритму.

Крок 1. Обчислення матриці ступенів D:

$$D = \text{diag}(W1)$$

Цей крок обчислює матрицю ступенів D, яка є діагональною матрицею, де кожен елемент на діагоналі є сумою відповідного рядка в матриці ваг W. Цей крок допомагає нормалізувати граф.

Крок 2. Обчислення Лапласіана L:

$$L = D - W, \ \boldsymbol{L_n = D^{-\frac{1}{2}} L \ D^{-\frac{1}{2}}}$$

Лапласіан графа L обчислюється шляхом віднімання матриці ваг W від матриці ступенів D. Лапласіан є важливим для захоплення структури графа і часто використовується в напівконтрольованому навчанні.

Крок 3. Обчислення середнього вектора міток c:

$$c = \frac{1}{m} F1$$

Тут обчислюється вектор c як середнє відомих міток у F. Цей крок використовується для центрування інформації про мітки.

Крок 4. Побудова матриці B:

$$B = [F - c, Z(2, n - m)]$$

Цей крок будує матрицю B, де $F - c$ — це центрована матриця міток, а $Z(2, n - m)$ — це нульова матриця, яка представляє невідомі мітки для немічених точок даних.

Крок 5. Ініціалізація U:

$$U = Z(n, 2)$$

Матриця U ініціалізується як нульова матриця розміром $(n \times 2)$. Ця матриця оновлюватиметься ітераційно для поширення інформації про мітки.

Крок 6. Ітераційний процес:

Запускається цикл на T ітерацій, де T — це кількість ітерацій, визначена користувачем.

Формула оновлення: На кожній ітерації застосовується формула оновлення U_j^k (2.20), щоб оновити елементи матриці U. Цей крок, ймовірно, включає поширення міток через граф на основі відносин, зафіксованих у матриці ваг W.

Крок 7. Призначення міток:

Після ітераційних оновлень алгоритм призначає мітки, вибираючи мітку, яка відповідає максимальному значенню для кожного вузла:

$$l_k = \arg\max 1_{1 \le j \le 2} U_j^k$$

Це призначає кожному вузлу k одну з двох можливих міток на основі значень у матриці U.

Вихід: Остаточний вихід — це вектор міток ($l = [l_1, l_2, \dots, l_n]$),де кожен елемент представляє призначену мітку для кожної точки даних. Кратко алгоритм виглядає наступним чином:

Алгоритм –**Diffuse Label Spreading**

Вхід: $\boldsymbol{W, F = [y_1, y_2, \dots, y_m], T}$

1. Обчислити матрицю ступенів: $\boldsymbol{D = \mathbf{diag}(W\mathbf{1})}$

2. Обчислити Лапласіан: $\boldsymbol{L = D - W, L_n = D^{-\frac{1}{2}} L\ D^{-\frac{1}{2}}}$

3. Обчислити середній вектор міток: $\boldsymbol{c = \frac{1}{m} F\mathbf{1}}$

4. Побудувати матрицю: $\boldsymbol{B = [F - c, Z(2, n - m)]}$

5. Ініціалізувати: $\boldsymbol{U = Z(n, 2)}$

6. Для $i = 1$ до T:

- Оновити: $\boldsymbol{U_j^k}$ використовуючи формулу оновлення (2.21)

7. Призначення міток: $\boldsymbol{l_k = \mathbf{arg\,max}\, \mathbf{1}_{1 \le j \le 2}\, U_j^k}$

Вихід: Вектор міток ($l = [l_1, l_2, \dots, l_n]$),

2.11 Висновки по розділу 2

У даному розділі розглянуто загальний алгоритм напівкерованого навчання на основі графового підходу та його основні кроки – побудова графу, розповсюдження міток, оптимізація, регулярізація.

Детально розглянуто та проведено порівняльний аналіз диференціальних методів напівконтрольованного навчання на основі графів **Label Propagation; Label Spreading; Poisson Learning.**

Запропонований новий узагальнюючий метод напівконтрольованого навчання на основі графів - узагальнене дифузне навчання на основі графового підходу з регуляризацією – Diffuse Label Spreading. Описан ітеративний алгоритм та надана узагальнена формула навчання.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМИ КЛАССИФІКАЦІЇ ДАНИХ

3.1 Технічні вимоги до розробки інтелектуальної системи

У даному розділі буде описана розробка інтелектуальної інтерактивної системи для вирішення задач классифікації реальних та синтетичних даних за допомогою узагальнених методів дифузного розповсюдження міток на основі графового підходу - «INTELLIGENT INTERACTIVE SYSTEM FOR SOLVING REAL AND SYNTHETIC DATASETAS CLASSIFICATION PROBLEMS USING GENERALIZED GRAPH-BASED DIFFUSE LABELS PROPAGATION METHODS».

Програмне забезпечення потрібно бути реалізовано як інтерактивне вікно з настройками відображенням та мати зручний інтерфейс користувача. Програмне забезпечення потрібно бути реалізовано на мові високого рівня – Python. Програмне забезпечення потрібна дозволяти:

- повинна мати можливість роботи з синтетичними та реальними даними, особливо з медичними даними по захворюванню серця,
- вибирати базу даних,
- вибирати метод,
- відображати дані,
- будувати граф даних,
- обчислювати метрики методу,
- будувати графік точності для заданої послідовності початкових даних,
- обчислювати та будувати статистичні характеристики середнього та середньої похибки точності методу.

Система повинна бути орієнтована на науковців, інженерів даних та студентів, які працюють із задачами машинного навчання та класифікації. Програма також повинна бути орієнтована на застосування у медицині як інструмент автоматичної медичної діагностиці та класифікації ризиків серцево-судинних захворювань.

3.2 Опис програми

Програма призначена для проведення інтерактивної класифікації синтетичних і реальних даних з використанням напівконтрольованих методів навчання, а також для аналізу впливу параметрів методу і візуалізації результатів і точності. Програма розроблена на мові Phyton і складається з одного інтерактивного вікна, в яке вводяться і виводяться дані. Загальний вигляд програми представлений на рис. 3.1.

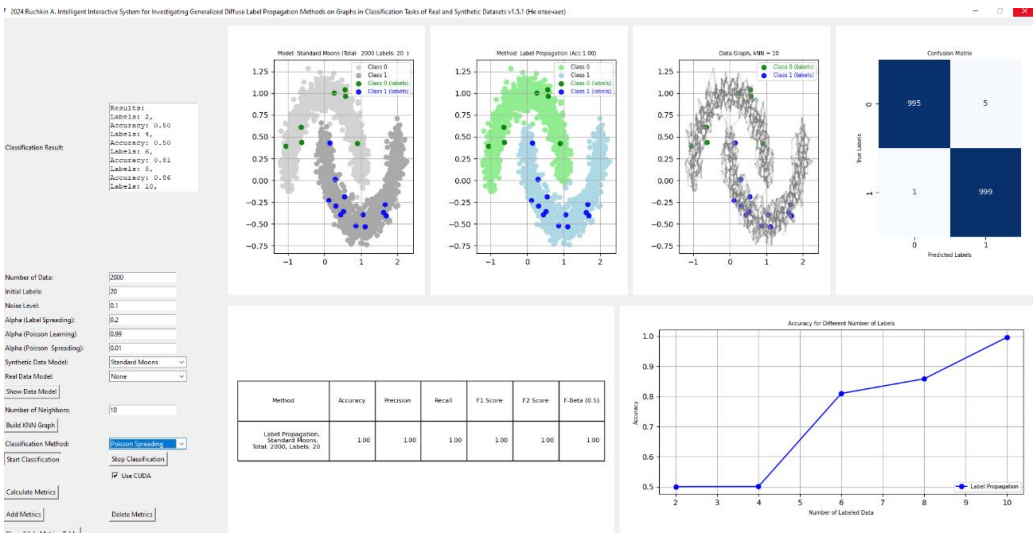

Рис. 3.1. Загальний вигляд програми

Програма включає в себе наступний функціонал:

- Панель введення параметрів (ліворуч), кнопки, меню що випадає, а також текстовий і числовий вивід (рисунок 3.2).
- Панелі візуалізації (зверху по центру і праворуч) - Візуалізація вихідного набору даних (зверху зліва), результатів класифікації (вгорі по центру), графіка даних (вгорі праворуч):
- Панель показників і результатів (внизу ліворуч і в центрі), таблиця показників, матриця помилок
- Панель графіка точності (внизу праворуч) - Графік точності (внизу праворуч)

Розглянемо ці елементи докладніше

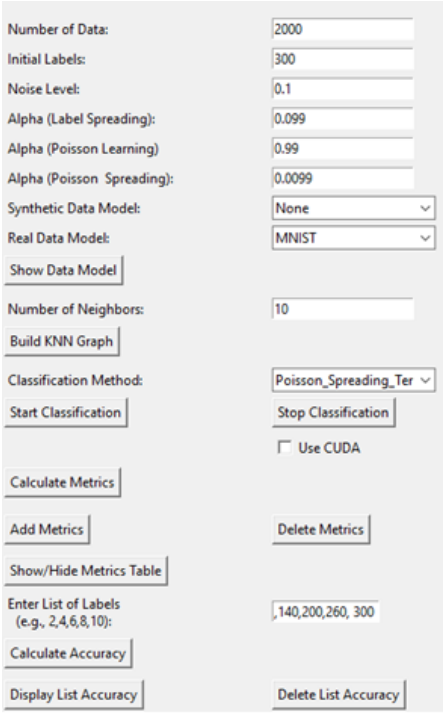

Рис.3.2. – Панель введення параметрів

3.2.1 Опис першої частини панелі параметрів

Функціональність панелі. Ця панель дозволяє користувачеві:

1. Налаштуйте параметри даних (гучність, рівень шуму та розмітку).
2. Підбирати та налаштовувати алгоритми класифікації.
3. Будуйте графіки даних для роботи з методами.
4. Розраховуйте та аналізуйте класифікаційні показники.
5. Проводьте експерименти з різною кількістю позначених даних і порівнюйте їх результати.

Нижче наведено докладний опис кожного елемента інтерфейсу:

Налаштування параметрів даних

- **Number of Data:**
  поле для введення загальної кількості точок даних у наборі даних. Наприклад, тут вказано 2000 балів.
- **Initial Labels:**
  поле для введення кількості позначених точок даних. Наприклад, в даному випадку 300 балів.
- **Noise Level:**
  рівень шуму в наборі даних (впливає на складність класифікації). Наприклад, вказано 0,1.

Параметри алгоритму

- **Alpha (Label Spreading):**

Коефіцієнт для методу **Label Spreading**. Значення визначає, наскільки мітки «змішуються» в процесі.

- **Alpha (Poisson Learning):**
  параметр алгоритму **Poisson Learning**, який керує процесом поширення міток.
- **Alpha (Poisson Spreading):**

специфічний параметр для методу **Poisson Spreading**, який контролює швидкість та інтенсивність процесу.

Вибір моделі даних

- **Synthetic Data Model:**

розкривний список для вибору синтетичної моделі даних (наприклад, Два Місяці або інші стандартні набори даних). Тут вибрано Nonon, що може означати, що користувач не використовує синтетичні дані.

- **Real Data Model:**

  випадаючий список для вибору реальних даних. У цьому випадку вибирається модель MNIST, яка представляє собою набір рукописних цифр.

- **Show Data Model (кнопка):**

  відображає вибрану модель даних, щоб користувач міг візуально переконатися в її правильності.

Налаштування графіка

- **Number of Neighbors:**

  кількість найближчих сусідів для побудови графіка (**k-Найближчі сусіди**). У цьому випадку значення дорівнює 10.

- **Build KNN Graph (кнопка):**

  Побудуйте графік на основі методу найближчого сусіда. Ця операція важлива для класифікації, оскільки такі методи, як **Label Propation**, вимагають структури графа.

Встановіть метод класифікації

Classification Method:

випадаючий список для вибору методу класифікації. У цьому випадку вибирається метод Poisson_Spreading_Ter.

Start Classification (кнопка):

Запустіть процес класифікації за допомогою обраного методу.

Stop Classification (кнопка):

за необхідності перервати процес класифікації.

Use CUDA (галочка):

Увімкніть обчислення GPU, щоб прискорити обробку.

Метрики та аналіз

- **Calculate Metrics (кнопка):**
  Обчислюйте показники (точність, бали Формули-1 та інші) після завершення класифікації.
- **Add Metrics (кнопка):**
  додавайте спеціальні показники, щоб аналізувати результати.
- **Delete Metrics (кнопка):**
  видаляйте показники, які більше не потрібні.
- **Show/Hide Metrics Table (кнопка):**
  Увімкніть або вимкніть відображення таблиці показників.

Порівняльний аналіз за кількістю оцінок

- **Enter List of Labels:**
  поле для введення списку кількості позначених даних для експериментів. Наприклад, це значення 140, 200, 260, 300.
- **Calculate Accuracy (кнопка):**
  Обчисліть точність для кожного значення у списку позначених даних.
- **Display List Accuracy (кнопка):**
  відображає графік точності на основі кількості позначених даних.
- **Delete List Accuracy (кнопка):**
  видалити графік точності списку, якщо він більше не потрібен.

3.2.2 Опис другої частини поля параметру

Цей фрагмент інтерфейсу користувача призначений для налаштування параметрів даних і алгоритмів, що використовуються для експериментів з напівконтрольованим навчанням.

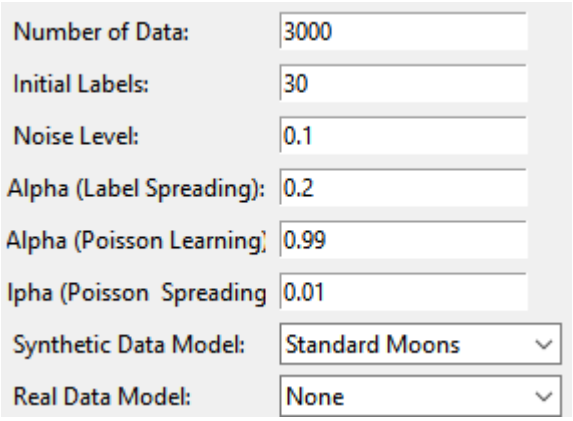

Рис. 3.3. Поле параметрів

Налаштування параметрів даних

- **Кількість даних:**
  вказує загальну кількість точок даних, які будуть використані в наборі. (3000)
- **Initial Labels:**
  кількість точок даних, які будуть спочатку позначені (тобто з відомими мітками класів). (Вказано 30 позначених точок.)
- **Noise Level:**
  рівень шуму в даних, які додаються для створення складнішого набору даних. Приклад: установлено значення 0,1 (10% шуму).

Параметри алгоритму

Alpha (Label Spreading):

коефіцієнт для методу **росповсюдження міток**. Цей параметр контролює, наскільки сусіди впливають на оновлення міток. (Встановлено значення 0,2, що вказує на помірне змішування етикеток.)

Alpha (Poisson Learning):

параметр для методу **Poisson Learning**, який контролює інтенсивність дифузії міток. (Значення 0,99 свідчить про сильний дифузійний процес.)

Alpha (Poisson Spreading):

параметр для методу **Poisson Spreading**, який регулює швидкість і стабільність процесу. (Дане значення дорівнює 0,01.)

Вибір моделі даних

Synthetic Data Model:

випадаючий список для вибору синтетичного набору даних.

(Вибрані стандартні **місяці)**

Real Data Model:

випадаючий список для вибору реального набору даних.

(Встановлено значення **None**, що означає, що фактичний набір даних не використовується.)

Функціональність

Користувач задає розмір набору даних, розмітку та параметри шуму для експериментів.

Регулює коефіцієнти, що впливають на роботу алгоритмів навчання.

Вибирає тип даних (синтетичні або реальні) для класифікації.

3.2.3 Поле вибору бази даних description

Цей блок панелі відповідає за вибір набору даних, який буде використовуватися в експериментах, а також за його візуалізацію.

Функціональність

Вибір набору даних:

- Дозволяє користувачеві вибирати між синтетичними та реальними даними для проведення експериментів над різними типами завдань.

Візуалізація:

- Дозволяє швидко оцінити поточний набір даних і переконатися в їх правильності перед виконанням алгоритмів.

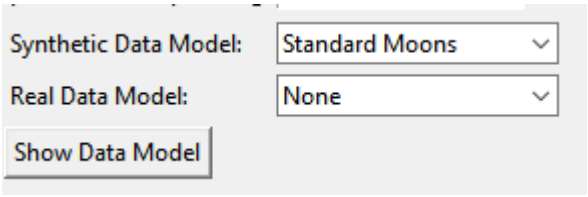

Рис. 3.4. Поле вибору бази даних

3.2.4 Опис вибору синтетичних даних

Цей випадаючий список надає вибір між різними типами синтетичних наборів даних. Ці набори даних призначені для тестування та аналізу алгоритмів

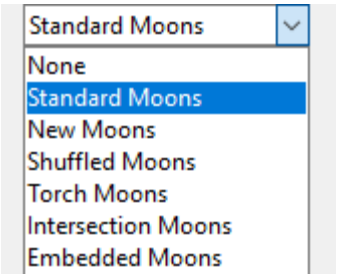

Рис. 3.5. Поле вибору синтетичних даних

Synthetic Data Model – штучний набір даних – варіації двох місяців:

a) **Standard Moons** - класична версія з бібліотеки SKlearn
b) **New Moons** - класичний варіант з квесту
c) **Shuffled Moons** - класична версія з Shutlle
d) **Torch Moons** – варіант двох місяців з дотиком
e) **Intersection Moons** - варіант двох місяців з перетином

f) **Embedded Moons** вбудовані в друг в друга

Варіанти на вибір:

None:

Вибір цього значення означає, що синтетичний набір даних не використовується. Програма буде чекати, поки буде обраний інший тип даних (наприклад, реальний з іншого випадаючого списку).

Standard Moons:

Стандартний набір даних Two Moons. Це дві напівкруглі групи точок, що перетинаються, які часто використовуються для перевірки методів класифікації. Цей набір даних добре підходить для тестування алгоритмів на складних, нелінійно відокремлюваних даних.

New Moons:

Альтернативна версія набору «Два місяці», з деякими модифікаціями, можливо, більш щільними або широкими дугами. Він може бути використаний для аналізу опору алгоритму модифікаціям структури даних.

Shuffled Moons:

Версія набору даних Two Moons, у якій точки перемішуються або переміщуються в інший порядок. Він використовується для перевірки алгоритмів на стійкість до випадкових перестановок даних.

Torch Moons:

Версія «Двох місяців», де стикаються дві групи точок.

Intersection Moons:

Версія «Двох місяців», де дві групи точок перетинаються або зливаються сильніше, ніж стандартна версія. Зростає складність класифікації, що дає можливість тестувати алгоритми на більш складних завданнях.

Embedded Moons:

Вкладені Місяці – це набір даних, в якому групи точок розташовані одна всередині одної. Це складний випадок для методів, оскільки класи знаходяться в просторі, де потрібно обробляти вкладеність.

3.2.5 Опис вибору реальних даних

За допомогою цього спадного списку можна вибрати фактичний набір даних, який використовуватиметься в експериментах.

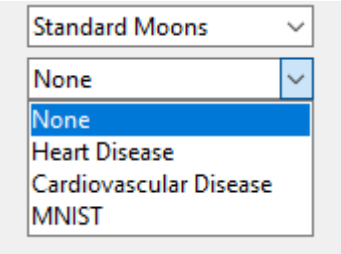

Рис.3.5. Поле для вибору реальних даних

Real Data Model – реальні бази даних

a) **Heart Disease (1000)**
b) **Cardiovascular Heart Disease (70 000)**
c) **MNIST. (42 000)**

Варіанти на вибір:

None:

- Вибір цього значення означає, що фактичний набір даних не використовується. Користувач може працювати тільки з синтетичними даними, які вибрані в списку **Synthetic Data Model**.

1. **Heart Disease: (1000)**

(https://www.kaggle.com/code/desalegngeb/heart-disease-predictions/notebook)

Набір даних, пов'язаних з діагностикою серцевих захворювань.Може містити такі характеристики, як вік, артеріальний тиск, рівень холестерину тощо. Він використовується для тестування алгоритмів на завданнях медичної класифікації.

2. **Cardiovascular Disease: (70 000)**

https://www.kaggle.com/datasets/jocelyndumlao/cardiovascular-disease-dataset

Ще один медичний набір даних зосереджений на виявленні серцево-судинних захворювань. Можливо, включає в себе більш детальні параметри в порівнянні з «Хворобами серця».Він використовується для вивчення алгоритмів діагностики та прогнозування.

3. **MNIST: (42 000)**

https://www.kaggle.com/code/cdeotte/how-to-choose-cnn-architecture-mnist/input

Стандартний набір даних зображень, що містить рукописні цифри (0–9). Використовується в задачах класифікації зображень. Особливість: Цей набір може бути попередньо оброблений для роботи в алгоритмах графів.

3.2.6 Опис останньої частини панелі параметрів

Цей блок панелі призначений для налаштування параметрів графіка, вибору методу класифікації, управління процесом класифікації та розрахунку метрик продуктивності. Ось опис кожного елемента:

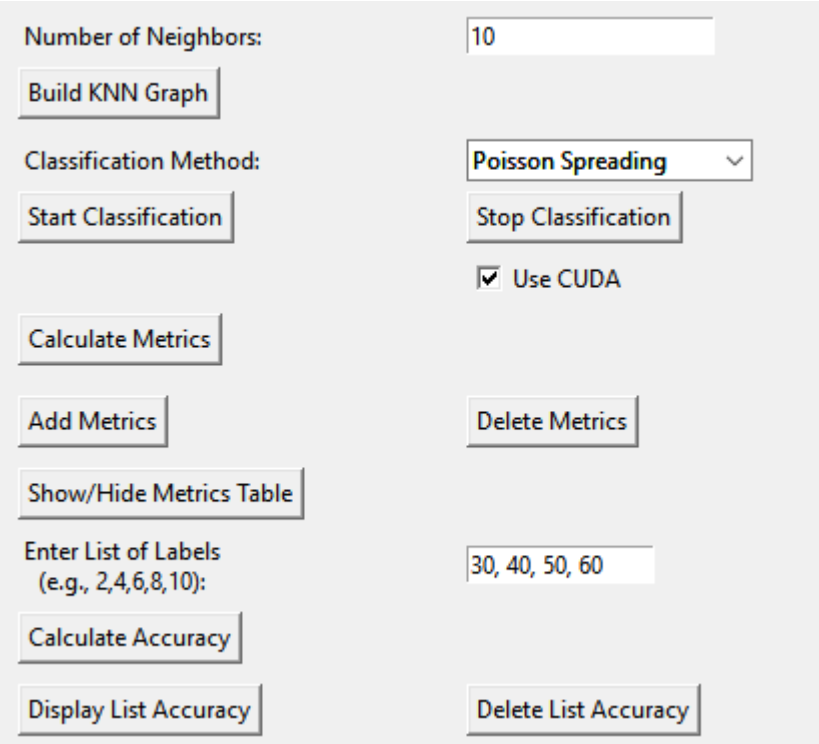

Рис.3.6. Панель введення параметрів (остання частина)

Налаштування параметрів графіка

Number of Neighbors:

поле для введення кількості найближчих сусідів для побудови графіка (**k-Nearest Neighbors**).

В даному випадку значення дорівнює 10, а це означає, що кожна точка буде асоціюватися з 10 найближчими сусідами.

Build KNN Graph (кнопка):

Кнопка для побудови графіка на основі методу найближчого сусіда.

- Це обов'язковий етап перед початком класифікації, так як алгоритми працюють зі структурою графа.

Вибір і управління методом класифікації

Classification Method:

випадаючий список для вибору методу класифікації.

- У цьому випадку вибирається метод **розкидання Пуассона** .

Start Classification (кнопка):

кнопка для початку класифікації за допомогою вибраного методу. Починається процес класифікації з використанням побудованого графіка і заданих параметрів.

Stop Classification:

кнопка для зупинки класифікації.

Корисно, якщо вам потрібно перервати процес через тривалий час виконання або помилку.

Use CUDA (галочка):

опція для ввімкнення підтримки **CUDA** (працює на графічному процесорі).

Коли ця функція активована, розрахунок класифікації може бути виконаний швидше за допомогою графічного процесора.

Класифікаційні метрики

Calculate Metrics (кнопка):

обчислює ключові показники ефективності (наприклад, точність, оцінка Формули-1, повнота тощо) на основі результатів класифікації.

Add Metrics (кнопка):

Дозволяє додавати додаткові користувацькі показники для аналізу.

Delete Metrics (кнопка):

видаляє раніше додані показники, які більше не потрібні.

Show/Hide Metrics Table (кнопка):

вмикає або вимикає відображення таблиці з обчислюваними показниками.

Експеримент з кількістю мічених даних

Enter List of Labels:

поле для введення списку значень, які вказують на кількість позначених даних, які використовуватимуться в експериментах. Наприклад, вводяться **30, 40, 50, 60**, а це означає, що розрахунок точності буде проводитися для випадків з 30, 40, 50 і 60 відміченими точками.

Calculate Accuracy (кнопка):

Обчислює точність класифікації для кожного значення у списку позначених даних. Корисно для аналізу впливу кількості позначених точок на продуктивність алгоритму.

Display List Accuracy (кнопка):

відображає графік точності на основі кількості позначених даних. Це наочний інструмент для аналізу.

Delete List Accuracy (кнопка):

видаляє графік точності, якщо він більше не потрібен.

Головні особливості цієї панелі

Побудова графіка:

Налаштуйте та побудуйте граф (k-найближчі сусіди) для подальшого використання в класифікації.

Вибір методу:

дозволяє вибрати один з методів класифікації (наприклад, **поширення мітки** або **поширення пуассона**).

Управління класифікацією:

Почніть і зупиніть процес класифікації за допомогою графічного процесора (CUDA).

Розрахунок показників:

підтримує обчислення основних і вторинних показників ефективності для аналізу результатів.

1. **Експерименти з розміткою:**

   Інструменти для аналізу залежності точності алгоритму від кількості мічених даних.

3.3 Робота з програмою

Загальний алгоритм роботи з програмою виглядає наступним чином:

1) Вибирайте основу - синтетичну або справжню;
2) Встановити загальний обсяг даних для аналізу;
3) Встановіть кількість мічених даних;
4) Встановити параметр шуму;
5) Встановити гіперпараметри;
6) Оберіть методику навчання;
7) Натисніть показати дані;
8) Встановити параметри для KNN;
9) Натисніть на побудову графіка;
10) Натисніть класифікувати;
11) Розраховувати показники;
12) Додайте метрики в таблицю;
13) Видалити метрики з таблиці;
14) Приховати/показати стіл;
15) Ввести список позначених вершин, для яких буде обчислюватися точність і заноситися в таблицю точності і графік точності;
16) Відобразити список;
17) Видаліть список.

Для того щоб провести класифікацію, потрібно скористатися наступним порядком дій

1) Вибираємо основу - синтетичну або справжню
2) Вкажіть загальний обсяг даних для аналізу
3) Установлення обсягу позначених даних
4) Встановіть параметр шуму
5) Встановити гіперпараметри

6) Оберіть методику навчання
7) Натисніть показати дані
8) Встановити параметри для KNN
9) Натисніть «Побудувати графік»
10) Натисніть классифікувати

Результат тестування і робота програми будуть продемонстровані в наступному розділі

3.4 Висновки до розділу 3

В розділі було розглянута розробка програмного забезпечення інтелектуальної системи классифікації даних за допомогою напівконтрольованих методів на основі графів.

А саме розроблене програмне забезпечення інтелектуальної системи классифікації даних дозволяє: відображати дані, будувати граф даних, обчислювати метрики, будувати графік точності для заданої послідовності початкових даних, обчислювати та будувати статистичні характеристики середнього та середньої похибки точності методу.

Система орієнтована на науковців, інженерів даних та студентів, які працюють із задачами машинного навчання та класифікації. Програма також орієнтована на застосування у медицині як інструмент автоматичної медичної діагностиці та класифікації ризиків серцево-судинних захворювань.

Так програма працює з наступними реальними базами даних: (**Heart Disease (1000), Cardiovascular Heart Disease (70 000), MNIST (42 000)).**

Та синтетичними даними: **«Two moons classic», «Two moons torch», «Two moons Intersection», «Two moons embeded».**

РОЗДІЛ 4 ТЕСТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Робота з реальними даними

В **розділі проведемо** тестування роботи інтелелекталльної системи з наступними реальними базами даних: **Heart Disease (1000), Cardiovascular Heart Disease (70 000), MNIST (42 000) т**а синтетичними даними: **«Two moons classic», «Two moons torch», «Two moons Intersection», «Two moons embeded».**

4.1.1 База даних – Cardio Vascular Disease

База даних – Cardio Vascular Disease включає в себе набір характеристик людей для визначення ступеня схильності до серцево-судинних захворювань. Загрузка та тестуання початкових даних

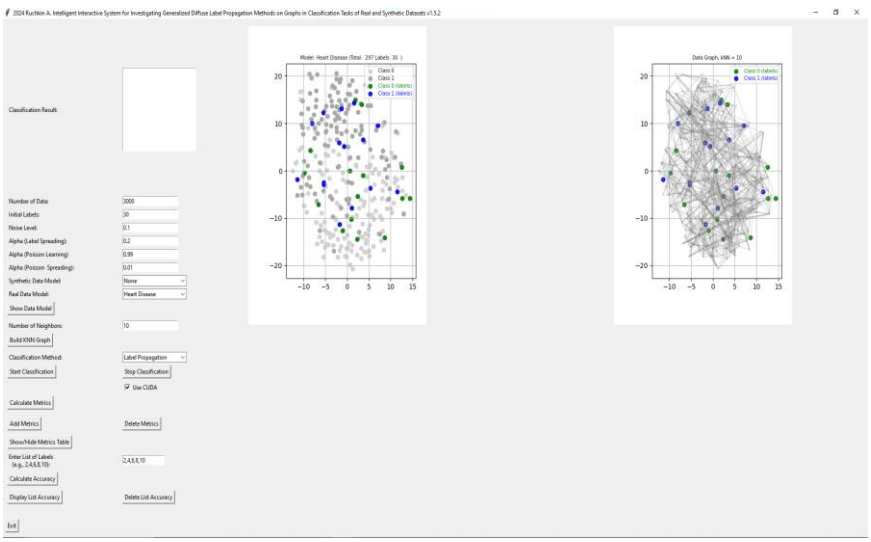

Рис.4.1. Для бази даних CardioVascularDisease було отримано початкову вибірку та побудовано графік.

Тестування класифікування. Графік будується за методом KNN (кількість сусідів вибирається вручну)

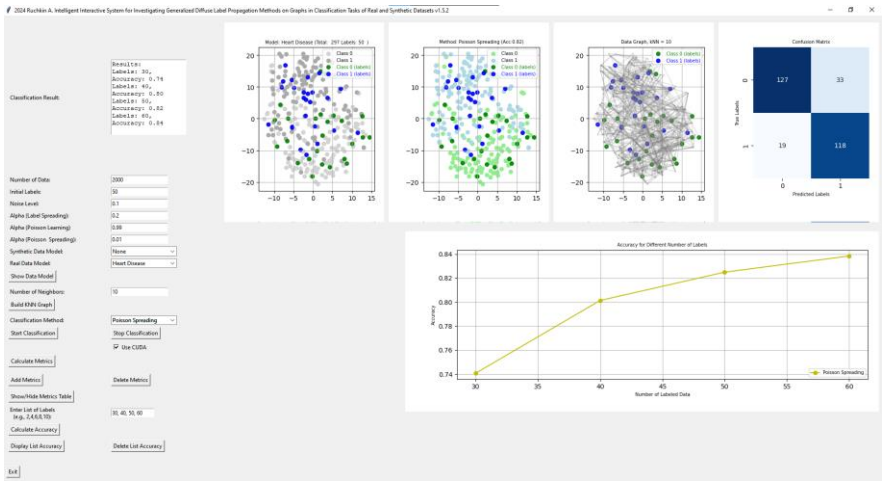

Рис.4.2. Для бази Cardio Vascular Disease результати класифікації

Тестування обчислення точноті класифікування та порявняння з іншими методами.

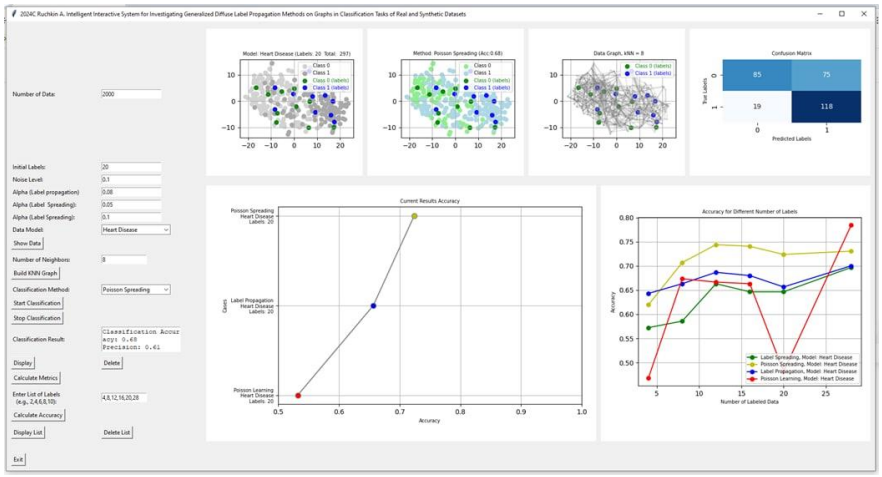

Рис.4.3. Для бази даних Cardio Vascular Disease порівняння результатів класифікації за чотирма методами

4.1.2 База даних - MNIST

Результати класифікації для бази даних MNIST (включаючи графіки продуктивності та порівняльні графіки з класичними методами половинного навчання графів). Тестування класифікування точності та порівняння з іншими методами.

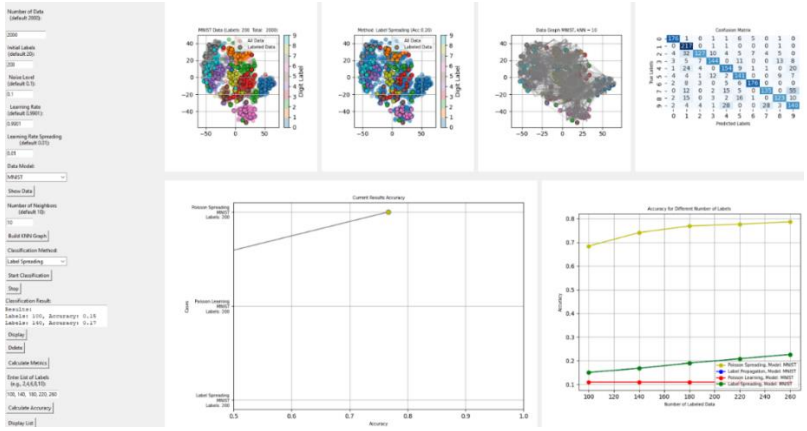

Рис.4.4. Для бази **MNIST** виводиться вихідна вибірка і будується графік, наводяться результати класифікації за чотирма методами

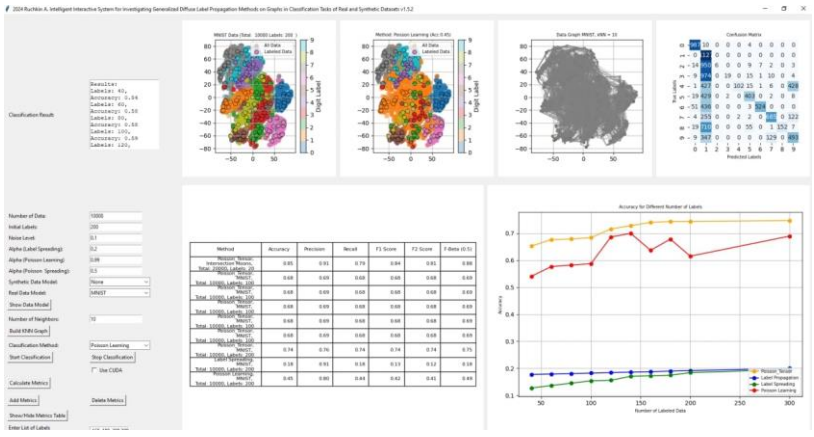

Рис.4.5. Для бази **MNIST** була виведена вихідна вибірка і побудований графік, наведені результати класифікації за чотирма методами

Тестування та приклади роботи з синтетичними даними базами даних:

4.2 Синтетичні дані

4.2.1 Синтетичні дані – «Two moons torch»

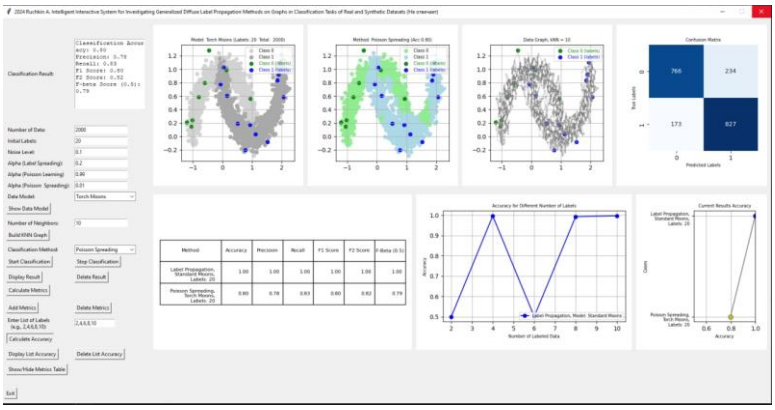

Рис.4.6. Для бази **«Two moons torch»** виводиться вихідний зразок і будується графік, а також наводяться результати класифікації

4.2.2 Синтетичні дані – «Two moons intersection»

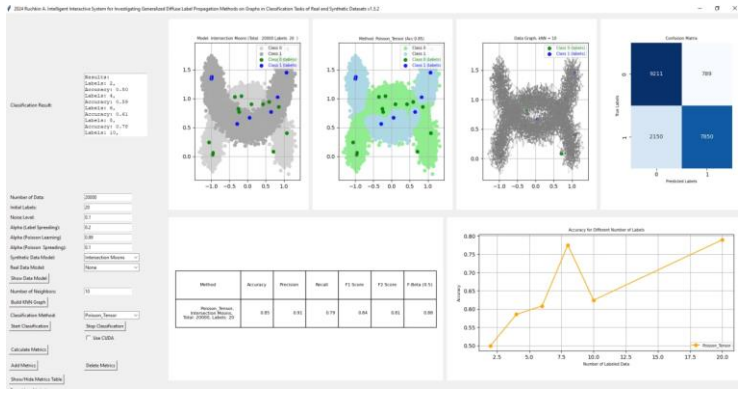

Рис.4.7. Для бази даних «**Two moons intersection»** виводиться вихідна вибірка і будується графік, а також представлені результати класифікації

4.3 Розрахунок та аналіз статистичних даних

Тестування функції розрахунку статистичних даних.

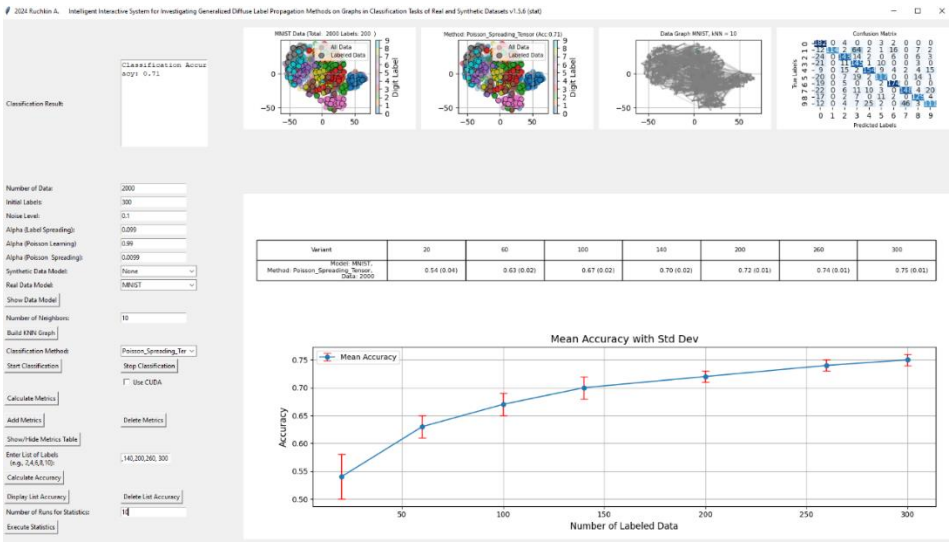

Рис.4.8. База MNIST - Розрахунок середньої точності та стандартного відхилення з 10 варіантів

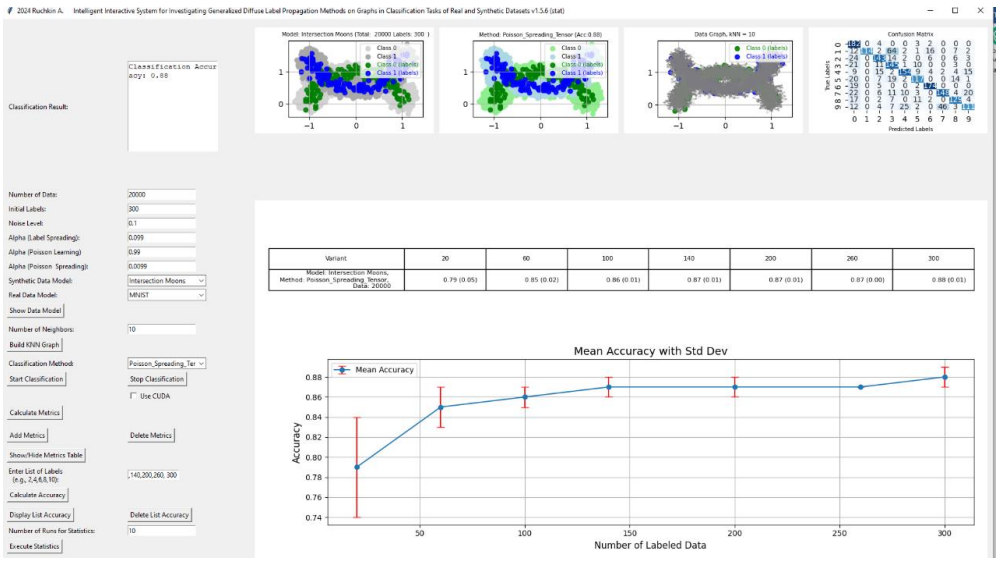

Рис.4.9. Перетин бази - двох місяців - розрахунок середньої точності та стандартного відхилення від 10 варіантів

4.4 Аналіз точності методу Diffuse Label Spreading

Аналіз точності методу Diffuse Label Spreading на реальних (Heart Disease (1000), Cardiovascular Heart Disease (70 000), MNIST (42 000)) та синтетичними даних «Two moons classic», «Two moons torch», «Two moons Intersection», «Two moons embeded».

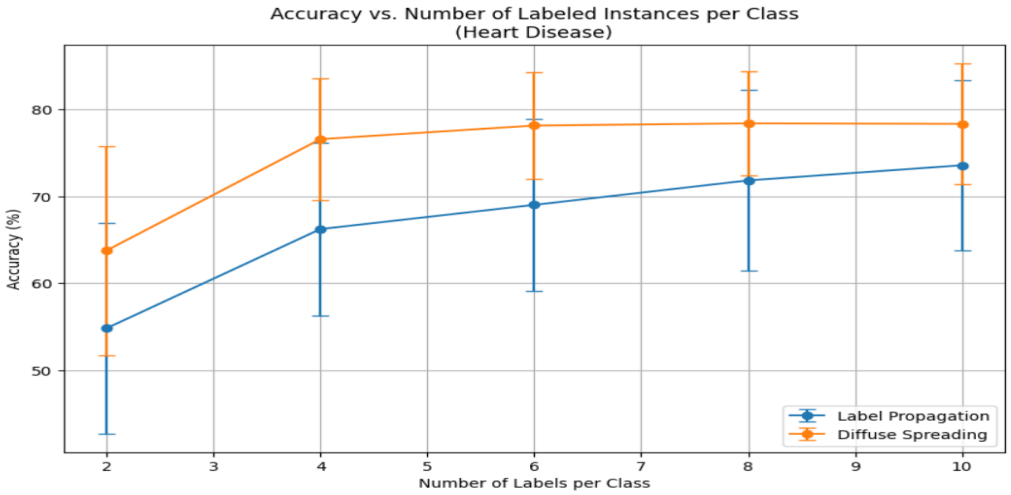

Рис.4.10. Порівняння середньої точності та стандартного відхилення методів для бази серцевих захворювань «Heart disease»(1000)

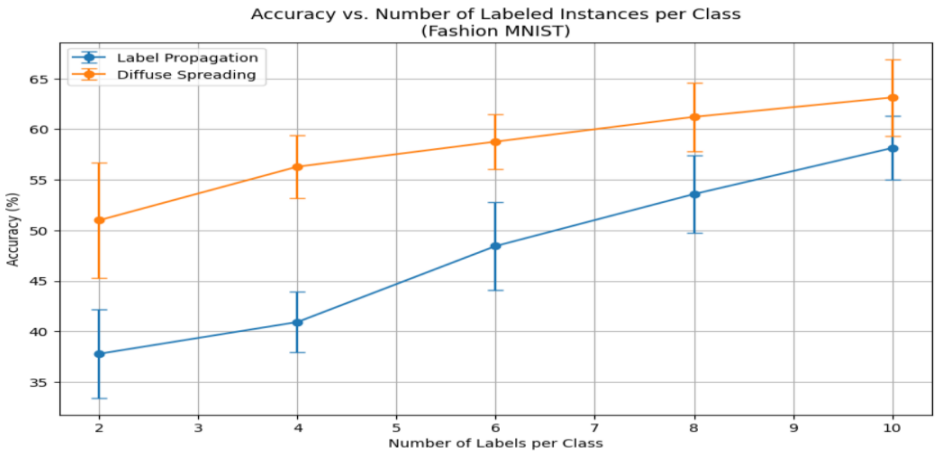

Рис.4.11. Порівняння середньої точності та стандартного відхилення методів для бази даних “Fashion MNIST”(45000)

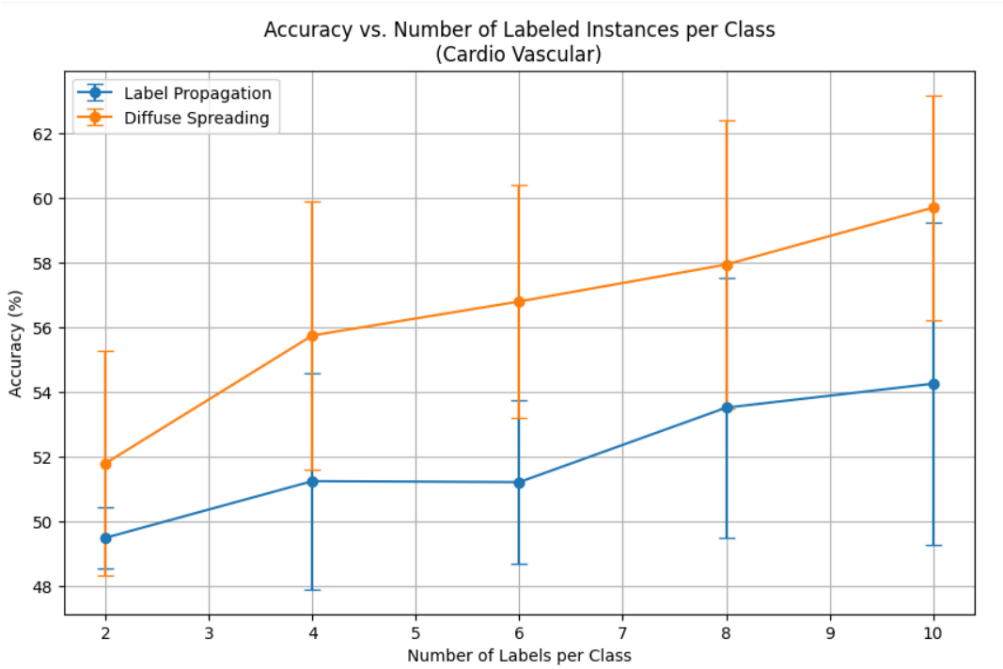

Рис.4.12. Порівняння середньої точності та стандартного відхилення методів для кардіо-судинної бази “Cardio Vascular Disease”(70000)

Проведено порівняльний аналіз методів Label Propagation; Label Spreading; Poisson Learning, Diffuse Label Spreading. Для синтетичних даних отримана точність 92%.

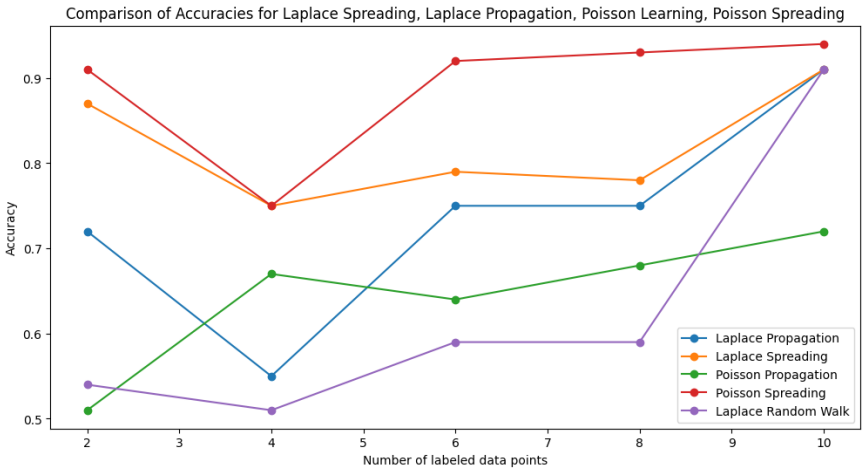

Рис.4.13. Результат порівняння точності Diffuse Label Spreading та інших методів

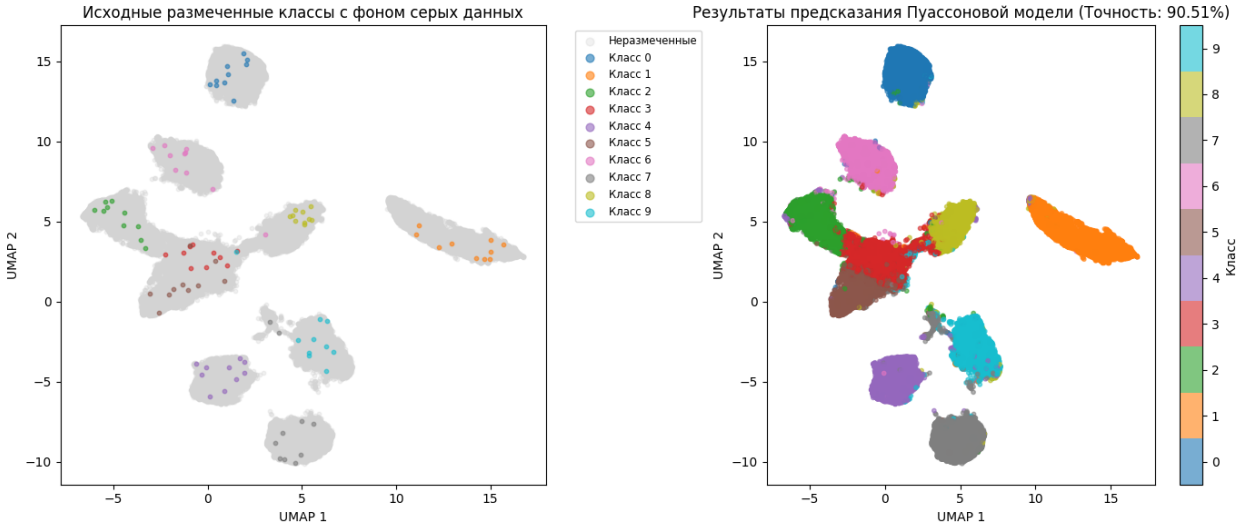

Рис.4.14. Результат класифікації даних бази MNIST

Про аналізуємо, як Diffuse Label Spreading працює з великими даними

Результати великих **даних 1 000 000**

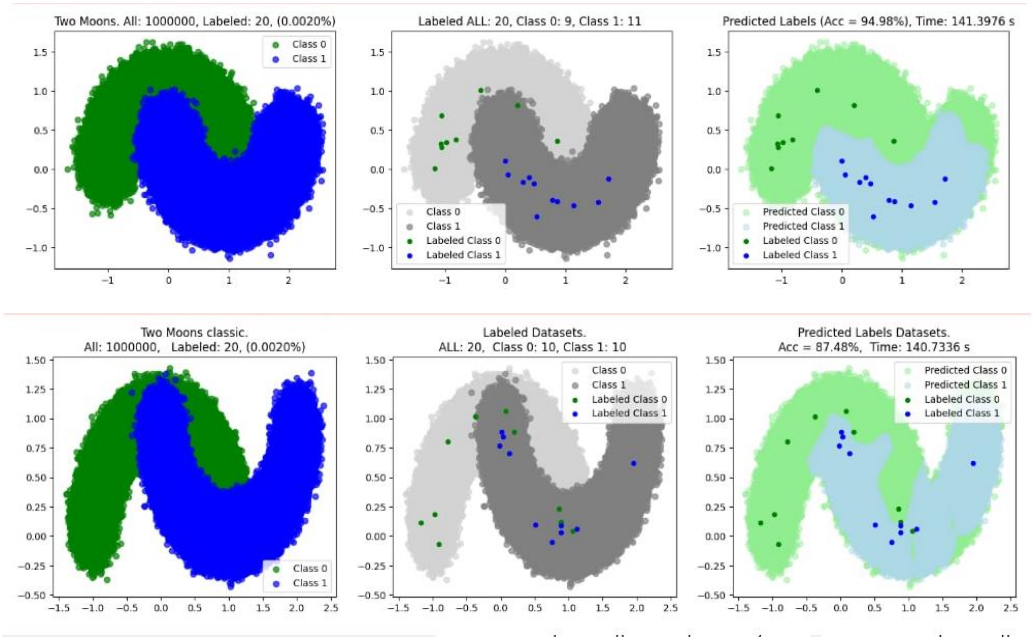

Рис.4.15. Бази даних **«Two moons classic», «Two moons torch»**

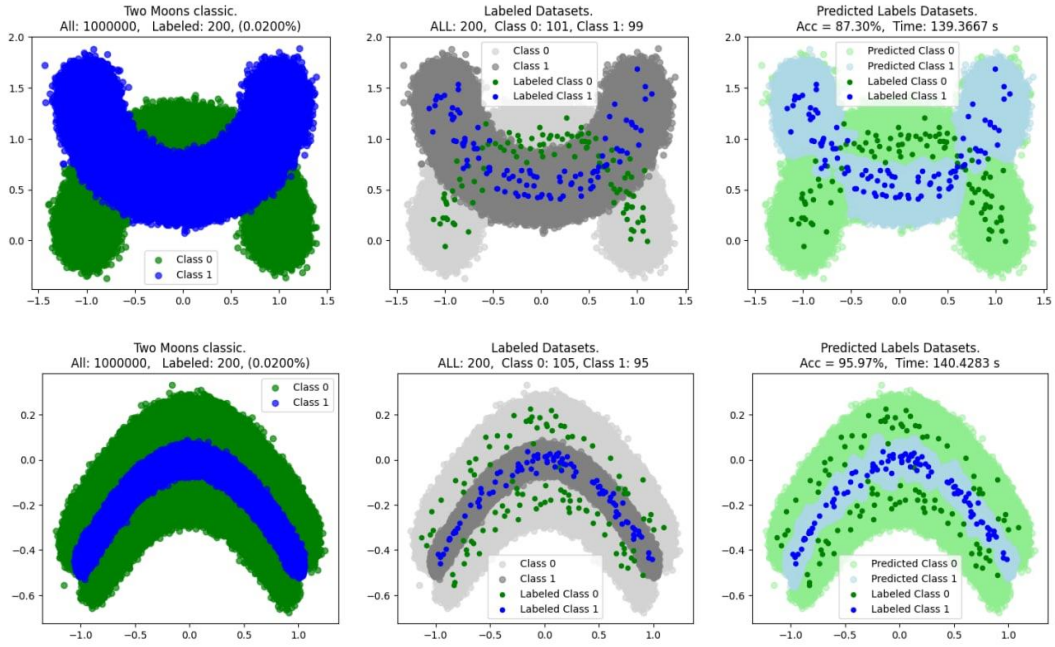

Рис.4.16. Бази даних **«Two moons Intersection», «Two moons embeded»**

На малюнках наведені дані про розрахунок точності і часу виконання

4.5 Висновки до розділу 4

У даному розділі проведено тестування інтелектуальної системи та аналіз результатів на синтетичних та реальних даних.

А саме протестоване роботу інтелектуальної системи на наступних даних:

- «Two moons classic», «Two moons torch», «Two moons Intersection», «Two moons embeded»
- «Heart disease»(1000), “Cardio Vascular Disease”(70000), “MNIST” (40000), “Fashion MNIST”(20000).

Проведено порівняльний аналіз методів Label Propagation; Label Spreading; Poisson Learning, Diffuse Label Spreading. Для синтетичних даних отримана точнысть 92%.

- Проведено порівняння середньої точності та стандартного відхилення методів для баз «Heart disease»(1000), “Cardio Vascular Disease”(70000), “MNIST” (40000), “Fashion MNIST”(20000). Отримані наступні результати точності – 78%, 65%, 60% при дуже малой кількості мічених даних – 10 даних.

Про аналізовано роботу алгоритму Diffuse Label Spreading з великими даними - 1 000 000 немічених.

- Для баз «Two moons classic», «Two moons torch» при 20 мічених (тобто при 0,00002%) отримана наступна точність - 95%, 87%.
- Для баз «Two moons Intersection», «Two moons embeded» при 200 мічених (тобто при 0,00002%) отримана наступна точність - 87%, 95%.

РОЗДІЛ 5 РОЗРОБКА СТАРТАП-ПРОЕКТУ «ІНТЕЛЕКТУАЛЬНА ІНТЕРАКТИВНА СИСТЕМА ДЛЯ ВИРІШЕННЯ ЗАДАЧ КЛАСИФІКАЦІЇ РЕАЛЬНИХ ТА СИНТЕТИЧНИХ ДАНИХ ЗА ДОПОМОГОЮ ГРАФОВИХ МЕТОДІВ»

5.1 Опис ідеї проекту

Старт-ап проект полягає у створенні інтелектуальної системи підтримки прийняття рішень для вирішення задач класифікування реальних та синтетичних даних за допомогою графових методів напівкерованого навчання. Зокрема система повинна мати можливість класифікування медичних даних стосовно захворювань серця. Розроблена система приймає на вході базу даних у форматі Execl. Данні повинні бути частково розмічені. Задаючи різні параметри та вибираючи різні методи класифікування користувач отримає точність класифікування даних залежності від кількості мічених даних на початку у графічному та цифровому форматі. Вихідні дані мають статистичний та рекомендаційний характер який дозволяє отримати загальну картину наявності та розподілу даних по класам. При використанні системи для вирішення задач классифікації медичних даних отримаємо точність з якої можливо поставити діагноз захворювання для групи або для індивідуума. Для групи отримати загальну кількість які захворіли.

Розглянемо загальну концепцію проекту. Інтелектуальна інтерактивна система розробляється для:

- Дослідження та оптимізації методів дифузійної пропагації міток, заснованих на графах.
- Аналізу та вирішення задач класифікації на реальних (наприклад, медичних або промислових) і синтетичних наборах даних.

Ключовими особливостями системи є:

Гнучкість: можливість налаштування параметрів графу (наприклад, ваги ребер, метод регулярізації) для експериментального аналізу.

Візуалізація: інтерактивне відображення результатів, включаючи початковий граф і підсумкову класифікацію.

Дослідження: підтримка широкого спектра методів поширення міток, таких як метод Пуассона, метод Лапласа, та гібридні підходи.

Проект орієнтований на наукову спільноту, студентів, а також інженерів, які займаються машинним навчанням і аналізом графів.

5.2 План розробки стартапу та масштабування його на ринок

Для успішного стартап проекту необхідно привести план розробки та виведення на ринок товару.

1. Виконання конкурентного аналізу для дослідження методів вирішення проблем, що вже використовуються в галузі.

2. Формування ідеї проекту та визначення цільової аудиторії.

3. Розробка стратегії введення товару на ринок, ґрунтуючись на аналізі ринкового середовища.

4. Організація стартапу, включаючи складання плану, розробку таймлайну розробки та запуску продукту.

5. Планування обсягу виробництва та оцінка потрібних ресурсів для виконання плану.

6. Розрахунок витрат, необхідних для реалізації проекту, та запуск.

7. Фінансово-економічний аналіз та оцінка ризиків стартап-проекту, включаючи визначення обсягу інвестиційних втрат.

8. Розрахунок основних фінансово-економічних показників проекту (собівартість, ціна продукту/послуги, податковий збір та чистий прибуток).

9. Визначення показників інвестиційної привабливості проекту, таких як рентабельність продажів та період окупності проекту.

10. Визначення основних ризиків проекту та розробка стратегій їх

11. Розробка заходів з комерціалізації продукту для масштабування та збільшення його розмірів.

12. Залучення інвесторів та знаходження різних шляхів фінансування проекту, включаючи проведення дослідження інтересів потенційних інвесторів.

13. Складання інвестиційної пропозиції, описуючи продукт, його теперішні розміри та можливі шляхи розширення та розвитку.

14. Вибір каналів комунікації з потенційно зацікавленими сторонами.

5.3 Технологічний аудит ідеї проекту

Розглянемо основні аспекти технологічної здійсненність проекту

Існуючі методи: Огляд сучасних підходів до пропагації міток на графах, їх переваги та обмеження. Наприклад, методи Лапласіана можуть бути чутливими до шуму в даних.

Технологічний стек: Вибір основних інструментів для реалізації, таких як Python-бібліотеки (NetworkX, Scikit-learn, PyTorch), для побудови графів, навчання моделей та їхньої візуалізації.

Масштабованість: Оцінка продуктивності системи на великих наборах даних. Наприклад, використання CUDA для прискорення обчислень у випадку великих графів.

Ризики: Ідентифікація можливих викликів, таких як точність при роботі з шумними даними або великими графами.

Цей аналіз дозволяє переконатися, що реалізація проекту є технологічно досяжною.

Таблиця 5.1 – Інформаційна карта стартап-проекту

Автор проєкту	Ручкін Олександр Костянтинович
Коротка анотація	Продукт може використовуватися науковою спільнотою, студентами, а також інженерами в які займаються машинним навчанням і аналізом графів та медичними працівниками.
Термін реалізації	2-3 роки
Необхідні ресурси	Програмне забезпечення для розробки, хмарні обчислювальні ресурси, договір з медичними закладами про постачання розмічених і нерозмічених даних, спеціалісти по сегментації зображень, спеціалісти з машинного навчання . Фінансові кошти на оплату всіх робітників протягом 24 місяці. Фінансові кошти на хмарне сховище баз даних
Головні цілі та завдання проекту	Старт-ап проект полягає у створенні інтелектуальної системи підтримки прийняття рішень для вирішення задач класифікування реальних та синтетичних даних за допомогою графових методів напівкерованого навчання. Зокрема система повинна мати можливість класифікування медичних даних стосовно захворювань сердця.
Очікувані результати	Реалізована система підтримки прийняття рішень та залучення медичних закладів до використання застосунку. Проведення ефективної маркетингової компанії по використанню застосунку як окремими користувачами так і медичними закладами.

5.4 Аналіз ринку та розробка ринкової стратегії

Цільова аудиторія:

Науковці, що займаються графовими алгоритмами.

Викладачі та студенти в університетах, які проводять експерименти з графовим навчанням. Інженери даних, які використовують графові методи для реальних задач класифікації.

Аналіз конкурентів: Огляд наявних інструментів, таких як GraphLab, Gephi, і визначення конкурентних переваг системи. Наприклад, інтерактивність і зручність налаштування параметрів.

Потреби ринку: Визначення потреб користувачів, таких як простота використання, точність і доступність.

Стратегія виходу на ринок: План створення спільноти користувачів через:

- Вебінари та воркшопи.
- Публікації на наукових конференціях.
- Відкрите тестування бета-версії.

5.5 Розроблення маркетингової програми

Загальний план просування системи:

Програмне забезпечення як сервіс: Використання моделі SaaS (Software-as-a-Service) для надання доступу через веб-платформу.

Доступність: Безкоштовна базова версія для освітніх закладів із можливістю преміум-доступу для розширених функцій.

Реклама: Промоція через соціальні мережі та платформи, такі як LinkedIn і GitHub. Участь у тематичних виставках і конференціях.

Зворотний зв'язок: Залучення користувачів до тестування та отримання рекомендацій щодо покращення системи.

5.6 Адаптація старт-ап проекту до впровадження у медичну сферу для діагностики захворювань серця

5.6.1 Додаткова мета проекту

Окрім наукового застосування для дослідження графових методів пропагації міток, проєкт передбачає адаптацію системи для використання у медичній сфері. Основною прикладною задачею є діагностика серцево-судинних захворювань, таких як:

- Ішемічна хвороба серця.
- Серцева недостатність.
- Аритмії та інші серцеві патології.

Система допоможе аналізувати медичні дані пацієнтів, виявляти закономірності та класифікувати пацієнтів за ризиками захворювань.

5.7 Додаткові функціональні можливості для медичної сфери

Розглянемо додаткові функціональні можливості для медичної системи

Підтримка медичних наборів даних:

Інтеграція популярних медичних наборів даних, наприклад, **Heart Disease Dataset** (з відкритих джерел) або локальних клінічних даних.

Визначення параметрів пацієнтів (наприклад, артеріальний тиск, рівень холестерину, частота серцевих скорочень) як атрибутів графа.

Медична класифікація:

Використання графів для зв'язування пацієнтів із подібними медичними показниками та поширення міток "здоровий/ризик/хвороба" на графі.

Автоматична класифікація пацієнтів на основі схожості їхніх даних.

Візуалізація діагностичних результатів:

Графічне представлення класифікації пацієнтів за групами ризику.

Побудова плутанинної матриці для оцінки якості класифікації медичних даних.

Можливості персоналізації:

Можливість налаштування алгоритмів для специфічних медичних даних.

Налаштування параметрів для обліку шуму у вибірці або особливостей конкретної групи пацієнтів.

Оцінка прогресу пацієнтів:

Аналіз змін у класифікації пацієнтів на основі оновлених даних (динамічний граф). Відстеження ефективності лікування та прогнозування ризиків.

5.7.1 Впровадження у медичну сферу

Проблеми, які вирішує система:

Рання діагностика: Автоматичне визначення пацієнтів із групи ризику на основі історичних даних.

Оптимізація роботи лікарів: Система допомагає швидко обробляти великі обсяги медичних даних, що економить час лікарів.

Покращення точності: За допомогою графових методів підвищується точність класифікації пацієнтів.

Інтеграція в клінічну практику:

Пілотне впровадження: Тестування системи на локальних медичних даних однієї клініки або госпіталю.

Навчання персоналу: Організація тренінгів для лікарів і медичних аналітиків щодо використання інтерфейсу.

Адаптація для реальних потреб: Налаштування параметрів системи під конкретні потреби лікарів, таких як:

Автоматична генерація звітів.

Інтеграція з системами управління електронними медичними картками.

5.7.2 Можливості розширення в медичній сфері

Розширення старт-апу в медичній сфері можливо провести за наступними напрямами.

Інтеграція з іншими даними: Аналіз інших медичних показників, таких як рентгенівські знімки, результати електрокардіограм тощо.

Розширення діагностичних можливостей: Застосування системи для аналізу інших захворювань, наприклад, цукрового діабету або легеневих хвороб.

Масштабування: Впровадження системи у великі лікарняні мережі з автоматичною синхронізацією даних.

5.7.3 Конкурентні переваги для медичної сфери

Відзначемо конкурентні переваги для медичної сфери. **Наукова обґрунтованість:** Використання графових методів, підтверджених сучасними дослідженнями.

Простота у використанні: Інтуїтивно зрозумілий інтерфейс, адаптований для лікарів.

Економічна ефективність: Зниження витрат на ручну обробку даних та прискорення процесів діагностики.

Цей проект поєднує передові алгоритми класифікації на графах із практичними потребами медичної галузі. Він сприятиме вдосконаленню діагностики серцево-судинних захворювань, оптимізації роботи лікарів та підвищенню якості медичного обслуговування.

Таблиця 5.2 – Попередня характеристика потенційного ринку стартап проекту

Показники (найменування)	Характеристика
Кількість головних гравців	3
Загальний обсяг продаж конкурентів, грн/ум. од	Підписка 50 ум.од за рік 7 ум.од. за одноразове використання
Динаміка ринку	Зростаюча, позитивна
Наявність обмежень для входу	Точність моделі повинна бути досить високою, щоб вважатися надійною для використання у медичній сфері. Велика вартість на навчання моделі. Наявність договорів з медичними закладами.
Специфічні вимоги до стандартизації та сертифікації	Можливі юридичні особливості надання консультацій для окремих користувачів. Юридичні особливості ведення міжнародної комерційної діяльності у сфері медицини.
Середня норма рентабельності в галузі (або по ринку), %	Залежить від успішності переговорів з медичними закладами та маркетинговій стратегії. 20%

5.8 Висновки до розділу 5

В цьому розділі представлено розробку стартап-проекту – «Інтелектуальна інтерактивна система для задач класифікації реальних та синтетичних даних за допомогою графових методів» з можливістю впровадження у медичну сферу для діагностики захворювань серця.

В розділі розглянуто загальні питання - Унікальність інтерактивної системи для аналізу методів розповсюдження міток. Технологічна та ринкова перспективність реалізації проекту. Чіткий план реалізації, включаючи технічну базу, маркетингові аспекти та стратегію залучення користувачів.

В розділі також обговорено. Опис ідеї проекту. План розробки стартапу та масштабування його на ринок. Технологічний аудит ідеї проекту. Аналіз ринку та розробка ринкової стратегії. Розроблення маркетингової програми. Адаптація старт-ап проекту до впровадження у медичну сферу для діагностики захворювань серця. Додаткові функціональні можливості для медичної сфери. Конкурентні переваги для медичної сфери.

ВИСНОВКИ

В кваліфікаційній роботі магістра вирішувалась задача – розробка нових і поліпшення існуючих методів GSSL та на їх основі розробка інтелектуальної системи вирішення задач класифікації для синтетичних та реальних даних.

В роботі вирішені наступні задачі:

- досліджено сучасний стан напряму та провести огляд методів за напрямом напівкерованого навчання на основі графів;
- розроблено узагальнений метод напівкерованого навчання на основі графів. А саме, розроблено узагальнений метод GSSL – Diffuse Label Spreading (DLS) на основі неоднорідних рівнянь дифузії (рівнянь Пуассону) та додаткової регулярізації;
- розроблено програмне забезпечення інтелектуальної системи классифікації на основі графового підходу та запропонованого узагальненого методу DLS;
- проведено тестування та аналіз результатів роботи інтелектуальної системи
- на основі інтелектуальної системи підготовлено стартап-проект.

Більш детально:

- Запропоновано новий узагальнений метод Diffuse Label Spreading. Новий узагальнений метод застосовує рівняння Пуассону нормалізований Лапласіан та додаткову регулярізацію L2
- Новий узагальнений метод показує кращі результати роботи (точність, час, метрики) в порівнянні з існуючими методами - Label Propagation, Label Spreading, Poisson Learning
- Розроблено програмне забезпечення інтелектуальної системи классифікації даних - «INTELLECTUAL INTERACTIVE SYSTEM FOR INVESTIGATING GENERALIZED DIFFUSE LABEL PROPAGATION METHODS BASED ON GRAPH FOR CLASSIFICATION TASKS OF REAL AND SYNTHETIC DATASETS

Програмне забезпечення дозволяє

- Проводити роботи з синтетичними та реальними даними, особливо з медичними даними по захворюванню серця,
- вибирати базу даних з заданих синтетичних та реальних даних,
- вибирати метод,
- відображати дані,
- будувати граф даних,
- обчислювати метрики методу,
- будувати графік точності для заданої послідовності початкових даних,
- обчислювати та будувати статистичні характеристики середнього та середньої похибки точності методу.

Також у програмі:

- реалізовані методи: Label Propagation, Label Spreading, Poisson Learning, та новий – Diffuse Label Spreading.
- реазізована оптимізація: Gradient Descent, Adam, Nesterov
- реалізовано загрузка синтетичних даних: «Two moons classic», «Two moons torch», «Two moons Intersection», «Two moons embeded».
- реалізовано загрузка реальних даних: «Heart disease» (1000), “Cardio Vascular Disease” (70000), “MNIST” (40000), “Fashion MNIST”(20000)

Проведено порівняння середньої точності та стандартного відхилення методів для баз «Heart disease»(1000), “Cardio Vascular Disease”(70000), “MNIST” (40000), “Fashion MNIST”(20000). Отримані наступні результати точності – 78%, 65%, 60% при невеликої кількості мічених даних – 10 даних.

Проаналізовано роботу алгоритму Diffuse Label Spreading з великими даними - 1 000 000 немічених. Отримано

- Для баз «Two moons classic», «Two moons torch» при 20 мічених (тобто при 0,00002%) отримана наступна точність - 95%, 87%.
- Для баз «Two moons Intersection», «Two moons embeded» при 200 мічених (тобто при 0,00002%) отримана наступна точність - 87%, 95%.

Програма написана на мові Phyton та має більш 2000 строк коду.

Усі поставлені в магістерської дисертації задачі виконано в повному обсязі.

Структура дисертації: магістерська дисертація складається з п’яти суттєвих розділів: вступ; перший розділ - огляд методів за напрямом напівкерованого навчання; другий розділ - розробка узагальненого методу напівкерованого навчання на основі графів; третій розділ - розробка інтелектуальної системи классифікації на основі графового підходу; четвертий розділ - тестування та аналіз результатів роботи інтелектуальної системи; п’ятий розділ - підготовка старт–ап проекту на основі інтелектуальної системи классифікації; висновки

ПЕРЕЛІК ПОСИЛАНЬ

1. Zhu, Xiaojin (Jerry). "Semi-Supervised Learning Literature Survey". Computer Sciences TR 1530 University of Wisconsin -Madison Last modifed on Sep 15, 2005. https://minds.wisconsin.edu/ handle/1793/60444
2. Zhu, Xiaojin (Jerry). "Semi-Supervised Learning Literature Survey". Computer Sciences TR 1530 University of Wisconsin -Madison Last modifed on July 19, 2008. https://www.researchgate.net/ publication/200688680 Semi-Supervised Learning Literature Survey
3. Jesper E. Van Engelen, J.E., Hoos, H.H. "A survey on semi-supervisedlearning". MachLearn 109, 373-440,. https://doi.org/10.1007/s10994-019-05855-6
4. Yunsheng Song, Jing Zhang, Chao Zhang, "A survey of large-scale graph-based semi-supervised classifcation algorithms". International Journal of Cognitive Computing in Engineering, Volume 3, 2022, Pages 188-198, ISSN 2666-3074. https://doi.org/10.1016/j.ijcce.2022.10.002
5. Z. Song, X. Yang, Z. Xu, I. King, "Graph-Based Semi-Supervised Learning: A Comprehensive Review", in IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 11, pp. 81748194, Nov. 2023. doi:10.1109/TNNLS.2022.3155478
6. X. Yang, Z. Song, I. King, Z. Xu, "A Survey on Deep Semi-Supervised Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 9, pp. 8934-8954, 1 Sept. 2023. doi: 10.1109/TKDE.2022.3220219
7. Aromal M. A., Akhtar Rasool. "Semi-Supervised Learning Using Graph Data Structure-A Review". 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV) : 894-899. https://api.semanticscholar.org/CorpusID:233138169
8. Matthieu Toutain, Abderrahim Elmoataz, Olivier Lezoray. "Geometric PDEs on weighted graphs for semi-supervised classifcation". International Conference on Machine Learning and Applications, IEEE, 2012, Detroit, United States. [2014]

hal-01108860) https://www.researchgate.
net/publication/278798624GeometricPDEs\protect\relax$\@@underline{\hbox{
}}\ mathsurround\z@$\relaxon Weighted Graphs for Semi-supervised
Classification

9. Jeff Calder. Brendan Cook. Matthew Thorpe. Dejan Slepcev. "Poisson Learning: Graph-based Semi-Supervised Learning At Very Low Label Rates". Proceedings of the 37th International Conference on Machine Learning. C Proceedings of Machine Learning Research. 2020. Hal Daume III Aarti Singh. Fpmlr-v119-calder20a. PMLR. P 1306-1316. V 119. https://proceedings.mlr.press/v119/ calder20a.html
10. Calder, J.: "The game theoretic p-Laplacian and semi-supervised learning with few labels". Nonlinearity 32(1), 301330. DOI:10.1088/1361-6544/aae949
11. Y. Chong, Y. Ding, Q. Yan, and S. Pan, "Graph-based semi-supervised learning: A review". Neurocomputing, vol. 408, pp. 216-230,2020. https://doi.org/10.1016/j.neucom.2019.12.130
12. Konstantin Avrachenkov, Inria Sophia-Antipolis, Maximilien Dreveton, "Chapter 5. Graph-based semi-supervised learning". France. 2021. http://dx.doi.org/10.1561/9781638280514.ch5
13. C. Chenetal. "Interactive Graph Construction for Graph-Based Semi-Supervised Learning", in IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 9, pp. 3701-3716, Sept. 1, 2021. https://doi.org/10.1109/TVCG.2021.3084694
14. Zhang, Chen Guang, Yan Zhang, Xia Huan Zhang. "Graph Based Semi-Supervised Learning Method for Imbalanced Dataset". Applied Mechanics and Materials. Trans Tech Publications, Ltd., May 2014. https://doi.org/10.4028/www.scientific.net/amm.556-562.4040
15. Zhuang L, Zhou Z, Gao S, Yin J, Lin Z, Ma Y. "Label Information Guided Graph Construction for Semi-Supervised Learning". IEEE Trans Image Process. 2017 Sep; 26(9): 4182-4192. https://doi. org/10.1109/TIP.2017.2703120

16. Liang, J., Cui, J., Wang, J. et al. "Graph-based semi-supervised learning for improving the quality of the graph dynamically". MachLearn 110, 1345-1388. https://doi.org/10.1007/ s10994-021-05975-y
17. Mehdi Sajjadi, Mehran Javanmardi, Tolga Tasdizen. "Regularization With Stochastic Transformations and Perturbations for Deep". NIPS'16: Proceedings of the 30th International Conference on Neural Information Processing Systems. December. 2016. Pages. 11711179. https://proceedings. neurips.cc/paper files/paper/2016/file/30ef30b64204a3088a26bc2e6ecf7602-Paper.pdf
18. Ke Chen, Shihai Wang. "Regularized Boost for Semi-Supervised Learning". Part of Advances in Neural Information Processing Systems 20 (NIPS 2007) https://papers.nips.cc/paper files/ paper/2007/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf
19. S. Sivananthan, "Multi-penalty regularization in learning theory", Journal of Complexity, Volume 36, 2016, Pages 141-165, ISSN 0885-064X, https://doi.org/10.1016/j.jco.2016.05.003
20. Zheng-Chu Guo, Shao-Bo Lin, Lei Shi, "Distributed learning with multi-penalty regularization", Applied and Computational Harmonic Analysis, Volume 46, Issue 3, 2019, Pages 478-499, ISSN 10635203, https://doi.org/10.1016/j.acha.2017.06.001
21. Naumova, V. Pereverzyev, S. V. "Multi-penalty regularization with a component-wise penalization". Inverse Problems, 2013. IOP Publishing -075002 -7 VL -29 SN -0266-5611. https://dx.doi.org/10.1088/0266-5611/29/7/075002
22. Markus Grasmair, Timo Klock, Valeriya Naumova, "Adaptive multi-penalty regularization based on a generalized Lasso path", Applied and Computational Harmonic Analysis, Volume 49, Issue 1, 2020, Pages 30-55, ISSN 1063-5203, https://doi.org/10.1016/j.acha.2018.11.001
23. Mikhail Belkin, Partha Niyogi, Vikas Sindhwani. "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples";

Journal of Machine Learning Research, 7,(85):2399-2434, 2006. http://jmlr.org/papers/v7/belkin06a.html

24. M. Fornasier, V. Naumova, S.V. Pereverzyev. "Parameter choice strategies for multi penalty regularization". SIAM J. Numer. Anal., 52 (4), pp. 1770-1794. https://www.ricam.oeaw.ac. at/files/reports/13/rep13-10.pdf
25. K. Ito, B. Jin, T. Takeuchi. "Multi-parameter Tikhonov regularization -an augmented approach". Chin. Ann. Math. Ser. B, 35 (3), pp. 383-398. https://www.researchgate.net/publication/241689312 Multi-Parameter Tikhonov Regularization -- An Augmented Approach
26. S. Lu, S.V. Pereverzev. "Multi-parameter regularization and its numerical realization". Numer. Math., 118 (1), pp. 1-31. https://www.researchgate.net/publication/225666729Multi-parameter regularization and its numerical realization
27. R. Llugsi, S. E. Yacoubi, A. Fontaine and P. Lupera, "Comparison between Adam, AdaMax and Adam W optimizers to implement a Weather Forecast based on Neural Networks for the Andeancity of Quito", 2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM), Cuenca, Ecuador, 2021, pp. 1-6, https://doi.org/10.1109/ETCM53643.2021.9590681
28. V. Sineglazov, K. Riazanovskiy, A. Klanovets, E. Chumachenko, and N. Linnik, "Intelligent tuberculosis activity assessment system based on an ensemble of neural networks", Comput. Biol. Med., vol. 147, Aug. 2022, Art. no. 105800. https://www.sciencedirect.com/science/article/pii/S0010482522005613
29. Sineglazov, V., Kot, A. "Design of hybrid neural networks of the ensemble structure". Eastern-European Journal of Enterprise Technologies, 2021. 1/4 (109), 3145. https://doi.org/10.15587/1729-4061.2021.225301
30. Zgurovsky, M., Sineglazov, V., Chumachenko, E. "Formation of Hybrid Artifcial Neural Networks Topologies. In: Artifcial Intelligence Systems Based on Hybrid Neural Networks". Studies in Computational Intelligence, 2021. vol 904. Springer, Cham. https://doi.org/10.1007/ 978-3-030-48453-8-3

31. Zgurovsky, M., Sineglazov, V., Chumachenko, E. "Classifcation and Analysis of Multi-criteria Optimization Methods". In: Artifcial Intelligence Systems Based on Hybrid Neural Networks. Studies in Computational Intelligence, vol 904. Springer, Cham. 2021. https://doi.org/10.1007/978-3-030-48453-8-2
32. Zgurovsky, M., Sineglazov, V., Chumachenko, E. "Classifcation and Analysis Topologies Known Artifcial Neurons and Neural Networks". In: Artifcial Intelligence Systems Based on Hybrid Neural Networks. Studies in Computational Intelligence, 2021. vol 904. Springer, Cham. doi.org/10.1007/978-3-030-48453-8-1
33. Smola, Alexander J.; Kondor, Risi. "Kernels and regularization on graphs". (English) Zbl 1274.68351 Sch6lkopf, Bernhard (ed.) et al., Learning theory and kernel machines. 16th annual conference on learning theory and 7th kernel workshop, COLT/Kernel 2003, Washington, DC, USA, August 2427, 2003. Proceedings. Berlin: Springer (ISBN 3-540-40720-0/pbk). Lect. Notes Comput. Sci. 2777, 144-158 (2003). https://people.cs.uchicago.edu/-risi/papers/SmolaKondor.pdf
34. Garcia-Cardona, Cristina; Merkurjev, Ekaterina; Bertozzi, Andrea; Flenner, Arjuna; Percus, Allon. "Multiclass Data Segmentation Using Difuse Interface Methods on Graphs". IEEE Transactions on Pattern Analysis and Machine Intelligence. 36.. doi:10.1109/TPAMI.2014.2300478. https://ieeexplore.ieee.org/document/6714564
35. R. Couillet, Z. Liao and X. Mai, "Classifcation Asymptotics in the Random Matrix Regime", 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 2018, pp. 1875-1879, doi: 10.23919/EUSIPCO.2018.8553034. https://ieeexplore.ieee.org/document/8553034
36. Jung, Alexander. "Sparse Label Propagation".. https://www.researchgate.net/publication/311425813 Sparse Label Propagation

37. Shi, Zuoqiang; Osher, Stanley; Zhu, Wei.. "Weighted Nonlocal Laplacian on Interpolation from Sparse Data". Journal of Scientifc Computing. 73. doi:10.1007/s10915-017-0421-z

38. O. Chapelle, B. Scholkopf and A. Zien, Eds., "Semi-Supervised Learning (Chapelle, O. et al., Eds.; 2006) [Book reviews]," in IEEE Transactions on Neural Networks, vol. 20, no. 3, pp. 542-542, March 2009, doi: 10.1109/TNN.2009.2015974 https://ieeexplore.ieee.org/document/4787647

39. A. Oliver, A. Odena, C.A. Rafel, E.D. Cubuk, I. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms -Advances in neural information processing systems, 2018 https://scholar.google.com/citations?view op=view citation&hl=en&user= Suu45K8AAAAJ&sortby=pubdate&citationforview=Suu45K8AAAAJ:d1gkVwhDpl0C

40. Rie Kubota Ando, Tong Zhang. "Learning on graph with Laplacian regularization". In Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06). MIT Press, Cambridge, MA, USA, 2532. 2006 https://dl.acm.org/doi/10.5555/2976456.2976460

41. Xu, Shuo; An, Xin; Qiao, Xiao-Dong; Zhu, Li-Jun; Li, Lin. "Semi-Supervised Least Squares Support Vector Regression Machines". Journal of Information and Computational Science. 8. 885-892.. https://www.researchgate.net/publication/215913159 Semi-SupervisedLeast Squares Support Vector Regression Machines

42. M. Toutain, A. Elmoataz and O. Lezoray, "Geometric PDEs on Weighted Graphs for Semisupervised Classifcation", 13th International Conference on Machine Learning and Applications, Detroit, MI, USA, 2014, pp. 231-236, 2014 doi:10.1109/ICMLA.2014.43

43. A. Tsitsulin, J. Palowitch, B. Perozzi, E. Mller, "Graph clustering with graph neural networks". Journal of Machine Learning Research 24, 1-21, 2023 https://jmlr.org/papers/volume24/20-998/ 20-998.pdf

ДОДАТОК А
ПЕРЕЛІК ПУБЛІКАЦІЙ

Участь у конференціях.

1. IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI'24), 10 – 13 November, 2024, Houston, TX USA
   Alexander Ruchkin, Victor Sineglazov.
   Application of Poisson Label Spreading for Solving the Classification Dataset of Heart Disease Data in Pandemic Conditions. https://bhi.embs.org/2024/

2. Learning on Graphs Conference 2024. 26 - 29 November
   Alexander Ruchkin, Victor Sineglazov.
   Regularization and Optimization of Poisson Label Propagation Method for Complex Dataset. https://logconference.org/

3. The 13th International Conference on Intelligent Control and Information Processing. February 6 - 11, 2025 Muscat, Oman
   Alexander Ruchkin, Victor Sineglazov.
   Diffuse Label Spreading with Regularisator
   https://conference.cs.cityu.edu.hk/icicip/

4. 1st International Scientific and Practical Conference
   Michael Z. Zgurovsky, Victor Sineglazov, Alexander Ruchkin. "Computational Intelligence and Smart Systems". October 10-12, 2024
   Modificated graph-based semi-supervised learning method based on using the Poissons equation

Роботи прийняті до друку

1. Computational Intelligence Application Workshop. CIAW-2024 October 10-12, 2024
   Michael Z. Zgurovsky, Victor Sineglazov, Alexander Ruchkin.
   Modified Poisson Label Propagation Graph-Based Method with Regularization for Task of Classification Intersection Dataset

Опубліковані роботи

2. Alexander Ruchkin.
   A Multi-Agent Computer Program for Automatic Investigation the Behavior of a Nonlinear Dynamic System in Real-Time.
   Conference paper, First Online: 12 March 2024, pp 511–531
   https://link.springer.com/chapter/10.1007/978-3-031-56496-3_32

ДОДАТОК Б

ЛИСТИНГ ПРОГРАМИ

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons,
fetch_openml
from sklearn.semi_supervised import
LabelPropagation, LabelSpreading
from sklearn.neighbors import kneighbors_graph
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
from sklearn.metrics import accuracy_score,
confusion_matrix, precision_score, recall_score,
f1_score, fbeta_score
from sklearn.utils import check_random_state,
shuffle as util_shuffle
import seaborn as sns
import os
import urllib.request
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import scipy.sparse as sp
from sklearn.preprocessing import OneHotEncoder

# Global variable to handle stopping of
computations
stop_requested = False

# Lists to store accuracies for the accuracy graph
accuracy_data = []
accuracy_labels = []

# Lists to store sequences of accuracies for
different label counts
accuracy_lists_data = []
label_lists_values = []
accuracy_lists_params = []

# List to store metrics entries
metrics_list = []

# Variable to track visibility of the metrics table
metrics_table_visible = True

# Mapping methods to colors
method_colors = {
    "Label Propagation": 'b',
    "Label Spreading": 'g',
    "Poisson Learning": 'r',
    "Poisson Learning Adam": 'c',
    "Poisson Learning Nesterov": 'm',
    "Poisson Spreading": 'y',
    "Poisson_Spreading_Tensor": 'orange'
}

# Definition of all data models (data generation
functions)
## 1
def new_make_moons(n_samples: int = 500, *,
distance: float = 0.5, noise: float = 0.1, seed: int =
42):
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out

    outer_circ_x = np.cos(np.linspace(0, np.pi,
n_samples_out))
    outer_circ_y = np.sin(np.linspace(0, np.pi,
n_samples_out))
    inner_circ_x = 1 - np.cos(np.linspace(0, np.pi,
n_samples_in))
    inner_circ_y = 1 - np.sin(np.linspace(0, np.pi,
n_samples_in)) - distance

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x),
np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [np.zeros(n_samples_out, dtype=np.intp),
np.ones(n_samples_in, dtype=np.intp)]
    )

    generator = np.random.RandomState(seed)
    X += generator.normal(scale=noise,
size=X.shape)

    return X, y
## 2
def make_moons_shuffle(n_samples=100, *,
shuffle=True, noise=None, random_state=None):
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out
    generator = check_random_state(random_state)

    outer_circ_x = np.cos(np.linspace(0, np.pi,
n_samples_out))
    outer_circ_y = np.sin(np.linspace(0, np.pi,
n_samples_out))
    inner_circ_x = 1 - np.cos(np.linspace(0, np.pi,
n_samples_in))
```

```
    inner_circ_y = 1 - np.sin(np.linspace(0, np.pi,
n_samples_in)) - 0.5

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x),
np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [np.zeros(n_samples_out, dtype=np.intp),
np.ones(n_samples_in, dtype=np.intp)]
    )

    if shuffle:
        X, y = util_shuffle(X, y,
random_state=generator)

    if noise is not None:
        X += generator.normal(scale=noise,
size=X.shape)

    return X, y
## 3
def make_moons_torch_shuffle(n_samples=100, *,
shuffle=True, noise=None, random_state=None):
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out
    generator = check_random_state(random_state)

    outer_circ_x = np.cos(np.linspace(0, np.pi,
n_samples_out))
    outer_circ_y = np.sin(np.linspace(0, np.pi,
n_samples_out))
    inner_circ_x = 1 - np.cos(np.linspace(0, np.pi,
n_samples_in))
    inner_circ_y = 1 - np.sin(np.linspace(0, np.pi,
n_samples_in)) - 0.0

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x),
np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [np.zeros(n_samples_out, dtype=np.intp),
np.ones(n_samples_in, dtype=np.intp)]
    )

    if shuffle:
        X, y = util_shuffle(X, y,
random_state=generator)

    if noise is not None:
        X += generator.normal(scale=noise,
size=X.shape)

    return X, y

def
make_moons_intersection_shuffle(n_samples=100
, *, shuffle=True, noise=None,
random_state=None):
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out
    generator = check_random_state(random_state)

    outer_circ_x = np.cos(np.linspace(0, np.pi,
n_samples_out))
    outer_circ_y = np.sin(np.linspace(0, np.pi,
n_samples_out))
    inner_circ_x = 1 - np.cos(np.linspace(0, np.pi,
n_samples_in)) - 1
    inner_circ_y = 1 - np.sin(np.linspace(0, np.pi,
n_samples_in)) + 0.5

    X = np.vstack(
        [np.append(outer_circ_x, inner_circ_x),
np.append(outer_circ_y, inner_circ_y)]
    ).T
    y = np.hstack(
        [np.zeros(n_samples_out, dtype=np.intp),
np.ones(n_samples_in, dtype=np.intp)]
    )

    if shuffle:
        X, y = util_shuffle(X, y,
random_state=generator)

    if noise is not None:
        X += generator.normal(scale=noise,
size=X.shape)

    return X, y

def make_moons_embeded_shuffle(
    n_samples: int = 500,
    *,
    shuffle=True,
    noise: float = 0,
    seed: int = 0,
    stertch_coef_top: float = 1.15,
    stretch_coef_bottom: float = 0.85,
    stretch_horizontal_scalar: float = 0.1,
    random_state=None
):
    n_samples_out = n_samples // 2
    n_samples_in = n_samples - n_samples_out
    generator = check_random_state(random_state)

    outer_circ_x_top = np.linspace(-1 -
stretch_horizontal_scalar, 1 +
stretch_horizontal_scalar, n_samples_out // 2)
    outer_circ_y_top = stertch_coef_top * np.cos((1
- stretch_horizontal_scalar) * outer_circ_x_top) - 1
```

```
    outer_circ_x_bottom = np.linspace(-1 +
stretch_horizontal_scalar, 1 -
stretch_horizontal_scalar, n_samples_out // 2)
    outer_circ_y_bottom = stretch_coef_bottom *
np.cos((1 + stretch_horizontal_scalar) *
outer_circ_x_bottom) - 1

    inner_circ_x = np.linspace(-1, 1,
n_samples_out)
    inner_circ_y = np.cos(inner_circ_x) - 1

    outer_banana_top =
np.vstack([outer_circ_x_top, outer_circ_y_top]).T
    outer_banana_top +=
generator.normal(scale=noise/2.5,
size=outer_banana_top.shape)

    outer_banana_bottom =
np.vstack([outer_circ_x_bottom,
outer_circ_y_bottom]).T
    outer_banana_bottom +=
generator.normal(scale=noise/2.5,
size=outer_banana_bottom.shape)

    inner_banana = np.vstack([inner_circ_x,
inner_circ_y]).T
    inner_banana +=
generator.normal(scale=noise/5.0,
size=inner_banana.shape)

    X = np.concatenate([outer_banana_top,
outer_banana_bottom, inner_banana], axis=0)
    y = np.hstack(
        [np.zeros(n_samples_out, dtype=np.intp),
np.ones(n_samples_in, dtype=np.intp)]
    )

    if shuffle:
        X, y = util_shuffle(X, y,
random_state=generator)

    return X, y

# Download and process Heart Disease data
def download_heart_disease_data():
    #url = 'https://archive.ics.uci.edu/ml/machine-
learning-databases/heart-
disease/processed.cleveland.data'
    #filename = 'processed.cleveland.data'

    filename = 'C:\\temp\\!Intellectual system
diffuse label propagation\\heart.csv'

    if not os.path.exists(filename):
        print("Downloading Heart Disease dataset...")
        urllib.request.urlretrieve(url, filename)
        print("Dataset downloaded successfully.")
    else:
        print("Dataset already exists.")



    return filename

def
load_and_process_heart_disease_data(filename):
    # Define column names
    column_names = [
        'age',       # age
        'sex',       # sex (1 = male; 0 = female)
        'cp',        # chest pain type
        'trestbps',  # resting blood pressure
        'chol',      # serum cholesterol
        'fbs',       # fasting blood sugar > 120 mg/dl (1
= true; 0 = false)
        'restecg',   # resting electrocardiographic
results
        'thalach',   # maximum heart rate achieved
        'exang',     # exercise-induced angina (1 =
yes; 0 = no)
        'oldpeak',   # ST depression induced by
exercise relative to rest
        'slope',     # the slope of the peak exercise ST
segment
        'ca',        # number of major vessels (0-3)
colored by fluoroscopy
        'thal',      # thalassemia
        'target'     # presence of heart disease
    ]

    # Read data
    #df = pd.read_csv('C:\\temp\\!Intellectual
system diffuse label propagation\\heart.csv')
    df = pd.read_csv(filename, header=None,
names=column_names)

    # Replace '?' with NaN
    df.replace('?', np.nan, inplace=True)

    # Convert all columns to numeric types
    for column in column_names:
        df[column] = pd.to_numeric(df[column],
errors='coerce')

    # Drop rows with missing values
    df.dropna(inplace=True)

    # Binary target variable (0: absence, 1: presence)
    df['target'] = df['target'].apply(lambda x: 1 if x >
0 else 0)
```

```
    # Split into features and target variable
    X = df.drop('target', axis=1).values
    y = df['target'].values

    return X, y

def preprocess_heart_disease_data(X):
    # Define categorical and numerical features
    # Categorical: 'sex', 'cp', 'fbs', 'restecg', 'exang',
'slope', 'ca', 'thal'
    # Numerical: 'age', 'trestbps', 'chol', 'thalach',
'oldpeak'

    categorical_features = [1, 2, 5, 6, 8, 10, 11, 12]
# column indices
    numerical_features = [0, 3, 4, 7, 9]

    # Scale numerical features
    scaler = StandardScaler()
    X[:, numerical_features] =
scaler.fit_transform(X[:, numerical_features])

    # Encode categorical features
    X_categorical = X[:,
categorical_features].astype(int)
    X_numerical = X[:, numerical_features]

    # One-Hot Encoding
    df_categorical = pd.DataFrame(X_categorical,
columns=['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',
'ca', 'thal'])
    df_categorical =
pd.get_dummies(df_categorical, columns=['cp',
'restecg', 'slope', 'thal'], drop_first=True)

    # Combine numerical and encoded categorical
features
    X_processed = np.hstack((X_numerical,
df_categorical.values))

    return X_processed

def initialize_heart_labels(y, n_labels,
random_state=42):
    np.random.seed(random_state)
    n = len(y)
    labels = -np.ones(n, dtype=int)  # -1 denotes
unlabeled

    n_classes = len(np.unique(y))
    n_labels_per_class = max(1, n_labels //
n_classes)

    for cls in np.unique(y):
        cls_indices = np.where(y == cls)[0]
        n_labeled_cls = min(n_labels_per_class,
len(cls_indices))
        labeled_cls_indices =
np.random.choice(cls_indices, n_labeled_cls,
replace=False)
        labels[labeled_cls_indices] = cls

    return labels

def initialize_mnist_labels(y, n_labels,
random_state=42):
    np.random.seed(random_state)
    n = len(y)
    labels = -np.ones(n, dtype=int)  # -1 denotes
unlabeled

    n_classes = len(np.unique(y))
    n_labels_per_class = max(1, n_labels //
n_classes)

    for cls in np.unique(y):
        cls_indices = np.where(y == cls)[0]
        n_labeled_cls = min(n_labels_per_class,
len(cls_indices))
        labeled_cls_indices =
np.random.choice(cls_indices, n_labels_per_class,
replace=False)
        labels[labeled_cls_indices] = cls

    return labels

##########################################
##########################################
##########################################
##
# New functions for Cardiovascular Disease
dataset
def load_and_process_cardio_data(filename):
    df = pd.read_csv(filename, sep=';')

    # Create 'age_years' from 'age' in days
    df['age_years'] = (df['age'] / 365).astype(int)
    df = df.drop('age', axis=1)

    # Map 'gender' from {1: 0 (female), 2: 1 (male)}
    df['gender'] = df['gender'].map({1: 0, 2: 1})

    # Convert columns to numeric if necessary
    df = df.apply(pd.to_numeric, errors='coerce')

    # Handle missing values if any
    df.dropna(inplace=True)

    # Target variable
    y = df['cardio'].values
    X = df.drop('cardio', axis=1).values
```

```
    return X, y

def preprocess_cardio_data(X):
    # Define column names for the original data
    df = pd.DataFrame(X, columns=['gender',
'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol',
'gluc', 'smoke', 'alco', 'active', 'age_years'])

    # Define numerical features
    numerical_features = ['height', 'weight', 'ap_hi',
'ap_lo', 'age_years']

    # Scale numerical features
    scaler = StandardScaler()
    df[numerical_features] =
scaler.fit_transform(df[numerical_features])

    # Convert categorical features to int
    categorical_features = ['gender', 'cholesterol',
'gluc', 'smoke', 'alco', 'active']
    df[categorical_features] =
df[categorical_features].astype(int)

    # One-Hot Encode categorical features
    df = pd.get_dummies(df, columns=['cholesterol',
'gluc'], drop_first=True)

    # Convert the processed DataFrame to a numpy
array
    X_processed = df.values

    return X_processed

def initialize_labels(y, n_labels, random_state=42):
    np.random.seed(random_state)
    n = len(y)
    labels = -np.ones(n, dtype=int)  # -1 denotes
unlabeled

    n_classes = len(np.unique(y))
    n_labels_per_class = max(1, n_labels //
n_classes)

    for cls in np.unique(y):
        cls_indices = np.where(y == cls)[0]
        n_labeled_cls = min(n_labels_per_class,
len(cls_indices))
        labeled_cls_indices =
np.random.choice(cls_indices, n_labels_per_class,
replace=False)
        labels[labeled_cls_indices] = cls

    return labels

########################################
########################################
######################
# Define classifiers
def poisson_learning(X, y, W, labeled_indices,
y_labeled, tolerance, max_iter, learning_rate):
    global stop_requested  # Reference to the global
variable
    n = W.shape[0]
    m = len(y_labeled)
    num_classes = len(np.unique(y))

    D = np.diag(W.sum(axis=1))
    L = D - W

    F = -np.ones((n, num_classes))
    for idx, label in zip(labeled_indices, y_labeled):
        F[idx, int(label)] = 1

    c = np.sum(F[F != -1], axis=0) / m
    B = F - c

    U = np.zeros((n, num_classes))
    D_inv = np.linalg.inv(D)

    accuracy_history = []

    for i in range(max_iter):
        if stop_requested:
            print("Computation stopped by user.")
            break

        if np.linalg.norm(U_new - U, ord='fro') <
tolerance:
            print(f"Converged after {i+1} iterations")
            break

        U = U_new

        labels = np.argmax(U, axis=1)
        accuracy_history.append(accuracy_score(y,
labels))

    labels = np.argmax(U, axis=1)
    return labels, accuracy_history

def poisson_spreading(X, y, W, labeled_indices,
y_labeled, tolerance, max_iter,
learning_rate_spreading):
    global stop_requested  # Reference to the global
variable
    n = W.shape[0]
    m = len(y_labeled)
    num_classes = len(np.unique(y))

    D = np.diag(W.sum(axis=1))
```

```
    L = D - W
    L_rw = np.linalg.inv(D) @ L

    F = -np.ones((n, num_classes))
    for idx, label in zip(labeled_indices, y_labeled):
        F[idx, int(label)] = 1

    c = np.sum(F[F != -1], axis=0) / m
    B = F - c

    U = np.zeros((n, num_classes))

    for i in range(max_iter):
        if stop_requested:
            print("Computation stopped by user.")
            break


        if np.linalg.norm(U_new - U, ord='fro') <
tolerance:
            print(f"Converged after {i+1} iterations")
            break

        U = U_new

    labels = np.argmax(U, axis=1)
    return labels

def poisson_learning_adam(X, y, W,
labeled_indices, y_labeled, tolerance, max_iter,
beta, beta1, beta2, epsilon):
    global stop_requested  # Reference to the global
variable
    n = W.shape[0]
    m = len(y_labeled)
    num_classes = len(np.unique(y))

    D = np.diag(W.sum(axis=1))
    L = D - W

    F = -np.ones((n, num_classes))
    for idx, label in zip(labeled_indices, y_labeled):
        F[idx, int(label)] = 1

    c = np.sum(F[F != -1], axis=0) / m
    B = F - c

    U = np.zeros((n, num_classes))
    D_inv = np.linalg.inv(D)

    m_t = np.zeros_like(U)
    v_t = np.zeros_like(U)
    t = 0

    accuracy_history = []

    for i in range(max_iter):
        if stop_requested:
            print("Computation stopped by user.")
            break
        t += 1


        m_t = beta1 * m_t + (1 - beta1) * grad
        v_t = beta2 * v_t + (1 - beta2) * (grad ** 2)

        m_t_hat = m_t / (1 - beta1 ** t)
        v_t_hat = v_t / (1 - beta2 ** t)

        U_new = U + beta * m_t_hat /
(np.sqrt(v_t_hat) + epsilon)

        if np.linalg.norm(U_new - U, ord='fro') <
tolerance:
            print(f"Converged after {i+1} iterations")
            break

        U = U_new

        labels = np.argmax(U, axis=1)
        accuracy_history.append(accuracy_score(y,
labels))

    labels = np.argmax(U, axis=1)
    return labels, accuracy_history

def poisson_learning_nesterov(X, y, W,
labeled_indices, y_labeled, tolerance, max_iter,
beta, momentum):
    global stop_requested  # Reference to the global
variable
    n = W.shape[0]
    m = len(y_labeled)
    num_classes = len(np.unique(y))

    D = np.diag(W.sum(axis=1))
    L = D - W

    F = -np.ones((n, num_classes))
    for idx, label in zip(labeled_indices, y_labeled):
        F[idx, int(label)] = 1

    c = np.sum(F[F != -1], axis=0) / m
    B = F - c

    U = np.zeros((n, num_classes))
    D_inv = np.linalg.inv(D)

    V = np.zeros_like(U)

    accuracy_history = []
```

```
    for i in range(max_iter):
        if stop_requested:
            print("Computation stopped by user.")
            break
        U_lookahead = U + momentum * V


        V = momentum * V + beta * grad
        U_new = U + V

        if np.linalg.norm(U_new - U, ord='fro') <
tolerance:
            print(f"Converged after {i+1} iterations")
            break

        U = U_new

        labels = np.argmax(U, axis=1)
        accuracy_history.append(accuracy_score(y,
labels))

    labels = np.argmax(U, axis=1)
    return labels, accuracy_history

class Poisson_Spreading_Tensor:
    def __init__(self, W=None,
solver='gradient_descent',
                 min_iter=50, max_iter=1000, tol=1e-3,
spectral_cutoff=10, use_cuda=False,
learning_rate_spreading = 0.01):
        """
        Poisson Learning for Semi-Supervised
Classification.
        """
        if solver != 'gradient_descent':
            raise ValueError("Only 'gradient_descent'
solver is implemented in this version.")
        self.W = W
        self.solver = solver
        self.use_cuda = use_cuda
        self.min_iter = min_iter
        self.max_iter = max_iter
        self.tol = tol
        self.spectral_cutoff = spectral_cutoff
        self.learning_rate_spreading =
learning_rate_spreading

    def _fit(self, train_ind, train_labels, n_classes):
        n = self.W.shape[0]
        k = n_classes

        # Zero out diagonal of W
        self.W = self.W - sp.diags(self.W.diagonal())

        # Compute Degree Matrix D
        degrees =
np.array(self.W.sum(axis=1)).flatten()
        D_inv = sp.diags(1 / (degrees + 1e-10))

        # Poisson source term
        onehot_encoder =
OneHotEncoder(sparse_output=False)
        onehot_labels =
onehot_encoder.fit_transform(train_labels.reshape(
-1, 1))
        y_bar = np.mean(onehot_labels, axis=0)
        source = np.zeros((n, k))
        source[train_ind] = onehot_labels - y_bar

        # Gradient Descent Solver
        P = D_inv.dot(self.W.T)
        Db = D_inv.dot(source)

        u = np.zeros((n, k))
        v = np.zeros(n)
        v[train_ind] = 1
        v /= np.sum(v)
        deg = degrees
        vinf = deg / np.sum(deg)
        RW = self.W.T.dot(D_inv)
        T = 0

        if self.use_cuda:
            try:

                # Convert sparse matrices to torch sparse
tensors
                device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')

                # Convert P to torch sparse tensor
                P_coo = P.tocoo()
                indices = np.vstack((P_coo.row,
P_coo.col))
                Pt = torch.sparse_coo_tensor(indices,
P_coo.data, P_coo.shape).float().to(device)

                # Convert u and Db to torch tensors
                ut = torch.zeros((n, k),
dtype=torch.float32, device=device)
                Dbt =
torch.from_numpy(Db).float().to(device)

                # Convert v and vinf to torch tensors
                vt =
torch.from_numpy(v).float().to(device)
                vinft =
torch.from_numpy(vinf).float().to(device)

                # RW matrix
                RW_coo = RW.tocoo()
```

```
            RW_indices = np.vstack((RW_coo.row,
RW_coo.col))
            RWt =
torch.sparse_coo_tensor(RW_indices,
RW_coo.data, RW_coo.shape).float().to(device)

            while (T < self.min_iter or
torch.max(torch.abs(vt - vinft)) > 1 / n) and (T <
self.max_iter):
                ut = Dbt + torch.sparse.mm(Pt, ut)
                vt = torch.sparse.mm(RWt,
vt.unsqueeze(1)).squeeze()
                T += 1

            # Transfer to CPU and convert to numpy
            self.u = ut.cpu().numpy()

        except ImportError:
            print("Torch is not available. Switching
to CPU.")
            self.use_cuda = False

    if not self.use_cuda:  # Use CPU
        while (T < self.min_iter or
np.max(np.abs(v - vinf)) > 1 / n) and (T <
self.max_iter):
            u = self.learning_rate_spreading * Db +
(1 - self.learning_rate_spreading) * P.dot(u)
            v = RW.dot(v)
            T += 1
        self.u = u

  def predict(self):
    pred_labels = np.argmax(self.u, axis=1)
    return pred_labels

########################################
########################################
########################
# Function to stop computations
def stop_computation():
  global stop_requested
  stop_requested = True
  print("Stop requested by user.")

def generate_data():
  global X, y, y_labels, n_labels
  try:
    n_samples = entry_samples.get()
    n_samples = int(n_samples) if n_samples else
200
    entry_samples.delete(0, tk.END)
    entry_samples.insert(0, str(n_samples))

    n_labels = entry_labels.get()
    n_labels = int(n_labels) if n_labels else 20
    entry_labels.delete(0, tk.END)
    entry_labels.insert(0, str(n_labels))

    noise_level = entry_noise.get()
    noise_level = float(noise_level) if noise_level
else 0.1
    entry_noise.delete(0, tk.END)
    entry_noise.insert(0, str(noise_level))

    # Check for validity of label count
    if n_labels < 1 or n_labels >= n_samples:
      raise ValueError("Number of initial labels
must be less than total data points.")

    synthetic_data_model =
synthetic_data_combobox.get()
    real_data_model = real_data_combobox.get()

    if synthetic_data_model != "None":
      data_model = synthetic_data_model
    elif real_data_model != "None":
      data_model = real_data_model
    else:
      raise ValueError("Please select a data
model.")

    if data_model == "Standard Moons":
      X, y =
make_moons(n_samples=n_samples,
noise=noise_level, random_state=42)
      y_labels = np.full_like(y, -1)
      indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
      y_labels[indices] = y[indices]
    elif data_model == "New Moons":
      X, y =
new_make_moons(n_samples=n_samples,
noise=noise_level, seed=42)
      y_labels = np.full_like(y, -1)
      indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
      y_labels[indices] = y[indices]
    elif data_model == "Shuffled Moons":
      X, y =
make_moons_shuffle(n_samples=n_samples,
noise=noise_level, random_state=42)
      y_labels = np.full_like(y, -1)
      indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
      y_labels[indices] = y[indices]
    elif data_model == "Torch Moons":
```

```
            X, y =
make_moons_torch_shuffle(n_samples=n_samples
, noise=noise_level, random_state=42)
            y_labels = np.full_like(y, -1)
            indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
            y_labels[indices] = y[indices]
        elif data_model == "Intersection Moons":
            X, y =
make_moons_intersection_shuffle(n_samples=n_s
amples, noise=noise_level, random_state=42)
            y_labels = np.full_like(y, -1)
            indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
            y_labels[indices] = y[indices]
        elif data_model == "Embedded Moons":
            X, y =
make_moons_embeded_shuffle(n_samples=n_sam
ples, noise=noise_level, random_state=42)
            y_labels = np.full_like(y, -1)
            indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
            y_labels[indices] = y[indices]
        elif data_model == "Heart Disease":
            filename = download_heart_disease_data()
            X_raw, y =
load_and_process_heart_disease_data(filename)
            X = preprocess_heart_disease_data(X_raw)
            n_samples = X.shape[0]
            n_labels = min(n_labels, n_samples)
            y_labels = initialize_heart_labels(y,
n_labels)
        elif data_model == "Cardiovascular Disease":
            filename = 'cardio_train-2000.csv'
            if not os.path.exists(filename):
                messagebox.showerror("Error",
f"Dataset {filename} not found.")
                return
            X_raw, y =
load_and_process_cardio_data(filename)
            X = preprocess_cardio_data(X_raw)
            n_samples = X.shape[0]
            n_labels = min(n_labels, n_samples)
            y_labels = initialize_labels(y, n_labels)
        elif data_model == "MNIST":
            X_all, y_all = fetch_openml('mnist_784',
version=1, return_X_y=True)
            y_all = y_all.astype(int)  # Convert labels
to integers

            n_samples = min(n_samples, len(y_all))
            X = X_all[:n_samples]
            y = y_all[:n_samples]
            n_labels = min(n_labels, n_samples)
            y_labels = initialize_mnist_labels(y,
n_labels)

            # Scale the data
            scaler = StandardScaler()
            X = scaler.fit_transform(X)
        else:
            raise ValueError("Please select a valid data
model.")

        # Display the initial data
        if X.shape[1] > 2:
            # Use t-SNE for visualization
            tsne = TSNE(n_components=2,
random_state=42)
            X_vis = tsne.fit_transform(X)
        else:
            X_vis = X

        fig1, ax1 = plt.subplots(figsize=(2.5, 2.5))
        if data_model == "MNIST":
            scatter = ax1.scatter(X_vis[:, 0], X_vis[:,
1], c=y, cmap='tab10', alpha=0.3, label='All Data')
            labeled_indices = np.where(y_labels != -
1)[0]
            scatter_labels =
ax1.scatter(X_vis[labeled_indices, 0],
X_vis[labeled_indices, 1],

c=y_labels[labeled_indices], cmap='tab10',
edgecolors='k', linewidths=0.5, s=50,
label='Labeled Data')
            cbar = plt.colorbar(scatter, ax=ax1,
ticks=range(10))
            cbar.set_label('Digit Label')
            ax1.set_title(f'MNIST Data (Total:
{n_samples} Labels: {n_labels}  )', fontsize=7)
            ax1.legend(loc='upper right', fontsize=7)
        else:
            ax1.scatter(X_vis[y == 0][:, 0], X_vis[y ==
0][:, 1], c='lightgrey', label='Class 0', marker='o')
            ax1.scatter(X_vis[y == 1][:, 0], X_vis[y ==
1][:, 1], c='darkgrey', label='Class 1', marker='o')
            ax1.scatter(X_vis[y_labels == 0][:, 0],
X_vis[y_labels == 0][:, 1], c='green', label='Class
0 (labels)', marker='o', alpha=0.9)
            ax1.scatter(X_vis[y_labels == 1][:, 0],
X_vis[y_labels == 1][:, 1], c='blue', label='Class 1
(labels)', marker='o', alpha=0.9)
            ax1.set_title(f'Model: {data_model} (Total:
{n_samples} Labels: {n_labels}  )', fontsize=7)
            ax1.legend(loc='upper right', fontsize=7)

            # Adjust legend colors
```

```
            legend = ax1.get_legend()
            for text in legend.get_texts():
                if text.get_text() == "Class 1 (labels)":
                    text.set_color("blue")
                elif text.get_text() == "Class 0 (labels)":
                    text.set_color("green")

        ax1.grid(True)
        plt.tight_layout(rect=[0, 0, 1.0, 1])

        # Clear previous plot
        for widget in
canvas_frame_left.winfo_children():
            widget.destroy()

        # Display the plot on the left side
        canvas1 = FigureCanvasTkAgg(fig1,
master=canvas_frame_left)
        canvas1.draw()
        canvas1.get_tk_widget().grid(row=0,
column=0, sticky="nsew")

        # Clear the result area
        for widget in
canvas_frame_right.winfo_children():
            widget.destroy()
        result_textbox.delete(1.0, tk.END)
    except ValueError as ve:
        messagebox.showerror("Error", str(ve))

# Function to build KNN graph
def build_knn_graph():
    if 'X' not in globals() or X is None:
        messagebox.showerror("Error", "Please
generate data first.")
        return

    n_neighbors_input = entry_neighbors.get()
    n_neighbors = int(n_neighbors_input) if
n_neighbors_input else 10
    entry_neighbors.delete(0, tk.END)
    entry_neighbors.insert(0, str(n_neighbors))

    # Compute W matrix for KNN
    W = kneighbors_graph(X,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

    # Visualize the graph connections
    fig2, ax2 = plt.subplots(figsize=(2.5, 2.5))

    # Display graph connections
    if X.shape[1] > 2:
        # Use t-SNE for visualization
        tsne = TSNE(n_components=2,
random_state=42)
        X_vis = tsne.fit_transform(X)
    else:
        X_vis = X

    for i in range(len(X_vis)):
        neighbors = np.where(W[i] == 1)[0]
        for neighbor in neighbors:
            if neighbor > i:  # Avoid drawing duplicate
lines
                ax2.plot([X_vis[i, 0], X_vis[neighbor,
0]], [X_vis[i, 1], X_vis[neighbor, 1]], color='gray',
alpha=0.2)

    # Display labels
    data_model = synthetic_data_combobox.get() if
synthetic_data_combobox.get() != "None" else
real_data_combobox.get()
    if data_model == "MNIST":
        labeled_indices = np.where(y_labels != -1)[0]
        scatter_labels =
ax2.scatter(X_vis[labeled_indices, 0],
X_vis[labeled_indices, 1],

c=y_labels[labeled_indices], cmap='tab10',
edgecolors='k', linewidths=0.5, s=20,
label='Labeled Data')
        ax2.set_title(f'Data Graph MNIST, kNN =
{n_neighbors}', fontsize=7)
    else:
        ax2.scatter(X_vis[y_labels == 0][:, 0],
X_vis[y_labels == 0][:, 1], c='green', label='Class
0 (labels)', marker='o', alpha=0.9)
        ax2.scatter(X_vis[y_labels == 1][:, 0],
X_vis[y_labels == 1][:, 1], c='blue', label='Class 1
(labels)', marker='o', alpha=0.9)
        ax2.set_title(f'Data Graph, kNN =
{n_neighbors}', fontsize=7)
        ax2.legend(loc='upper right', fontsize=7)

        # Adjust legend colors
        legend = ax2.get_legend()
        for text in legend.get_texts():
            if text.get_text() == "Class 1 (labels)":
                text.set_color("blue")
            elif text.get_text() == "Class 0 (labels)":
                text.set_color("green")

    ax2.grid(True)
    plt.tight_layout(rect=[0, 0, 1.0, 1])

    # Clear KNN graph
    for widget in
canvas_frame_graph.winfo_children():
        widget.destroy()

    # Display the graph in the window
```

```
    canvas2 = FigureCanvasTkAgg(fig2,
master=canvas_frame_graph)
    canvas2.draw()
    canvas2.get_tk_widget().grid(row=0, column=0,
sticky="nsew")

# Function for classification and displaying the
result
def classify_two_moons():
    global X, y, y_labels, y_pred, n_labels,
stop_requested, current_accuracy, current_params
    if 'X' not in globals() or X is None or y is None
or y_labels is None:
        messagebox.showerror("Error", "Please
generate data first.")
        return

    method = method_combobox.get()

    n_neighbors_input = entry_neighbors.get()
    n_neighbors = int(n_neighbors_input) if
n_neighbors_input else 10
    entry_neighbors.delete(0, tk.END)
    entry_neighbors.insert(0, str(n_neighbors))

    # Get learning rates from entries
    learning_rate_input = entry_learning_rate.get()
    learning_rate = float(learning_rate_input) if
learning_rate_input else 0.99
    entry_learning_rate.delete(0, tk.END)
    entry_learning_rate.insert(0, str(learning_rate))

    learning_rate_spreading_input =
entry_learning_rate_spreading.get()
    learning_rate_spreading =
float(learning_rate_spreading_input) if
learning_rate_spreading_input else 0.01
    entry_learning_rate_spreading.delete(0, tk.END)
    entry_learning_rate_spreading.insert(0,
str(learning_rate_spreading))

    # Get alpha parameter for Label Spreading
    alpha_input = entry_alpha.get()
    alpha = float(alpha_input) if alpha_input else 0.2
    entry_alpha.delete(0, tk.END)
    entry_alpha.insert(0, str(alpha))

    # Reset stop_requested before starting
computations
    stop_requested = False

    if method == "Label Propagation":
        model = LabelPropagation()
        model.fit(X, y_labels)
        y_pred = model.transduction_
    elif method == "Label Spreading":
        model = LabelSpreading(alpha=alpha)
        model.fit(X, y_labels)
        y_pred = model.transduction_
    elif method == "Poisson Learning":
        tolerance = 1e-5
        max_iter = 200
        # learning_rate obtained from entry

        W = kneighbors_graph(X,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

        labeled_indices = np.where(y_labels != -1)[0]
        y_labeled = y_labels[labeled_indices]

        y_pred, _ = poisson_learning(X, y, W,
labeled_indices, y_labeled, tolerance, max_iter,
learning_rate)

    elif method == "Poisson Learning Adam":
        tolerance = 1e-5
        max_iter = 200
        beta = 0.01
        beta1 = 0.9
        beta2 = 0.999
        epsilon = 1e-8

        W = kneighbors_graph(X,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

        labeled_indices = np.where(y_labels != -1)[0]
        y_labeled = y_labels[labeled_indices]

        y_pred, _ = poisson_learning_adam(X, y, W,
labeled_indices, y_labeled, tolerance, max_iter,
beta, beta1, beta2, epsilon)

    elif method == "Poisson Learning Nesterov":
        tolerance = 1e-5
        max_iter = 200
        beta = 0.01
        momentum = 0.9

        W = kneighbors_graph(X,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

        labeled_indices = np.where(y_labels != -1)[0]
        y_labeled = y_labels[labeled_indices]

        y_pred, _ = poisson_learning_nesterov(X, y,
W, labeled_indices, y_labeled, tolerance, max_iter,
beta, momentum)

    elif method == "Poisson Spreading":
```

```
        tolerance = 1e-5
        max_iter = 200
        # learning_rate_spreading obtained from entry

        W = kneighbors_graph(X,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

        labeled_indices = np.where(y_labels != -1)[0]
        y_labeled = y_labels[labeled_indices]



    # Save current accuracy and parameters for the
graph
    global current_accuracy, current_params
    data_model = synthetic_data_combobox.get() if
synthetic_data_combobox.get() != "None" else
real_data_combobox.get()
    current_accuracy = accuracy
    n_samples = X.shape[0]  # Get total number of
data
    current_params =
f"\n{method},\n{data_model}, \nTotal:
{n_samples}, Labels: {n_labels}\n"

     # Visualization of classification results
    if X.shape[1] > 2:
        tsne = TSNE(n_components=2,
random_state=42)
        X_vis = tsne.fit_transform(X)
    else:
        X_vis = X

########################################
########################################
    fig3, ax3 = plt.subplots(figsize=(2.5, 2.5))
    if data_model == "MNIST":
        scatter = ax3.scatter(X_vis[:, 0], X_vis[:, 1],
c=y_pred, cmap='tab10', alpha=0.3, label='All
Data')
        labeled_indices = np.where(y_labels != -1)[0]
        scatter_labels =
ax3.scatter(X_vis[labeled_indices, 0],
X_vis[labeled_indices, 1],

c=y_labels[labeled_indices], cmap='tab10',
edgecolors='k', linewidths=0.5, s=50,
label='Labeled Data')
        cbar = plt.colorbar(scatter, ax=ax3,
ticks=range(10))
        cbar.set_label('Digit Label')
        ax3.set_title(f'Method: {method}
(Acc:{accuracy:.2f})', fontsize=7)
        ax3.legend(loc='upper right', fontsize=7)
    else:
        ax3.scatter(X_vis[y_pred == 0][:, 0],
X_vis[y_pred == 0][:, 1], c='lightgreen',
label='Class 0', marker='o')
        ax3.scatter(X_vis[y_pred == 1][:, 0],
X_vis[y_pred == 1][:, 1], c='lightblue',
label='Class 1', marker='o')
        ax3.scatter(X_vis[y_labels == 0][:, 0],
X_vis[y_labels == 0][:, 1], c='green', label='Class
0 (labels)', marker='o', alpha=0.9)
        ax3.scatter(X_vis[y_labels == 1][:, 0],
X_vis[y_labels == 1][:, 1], c='blue', label='Class 1
(labels)', marker='o', alpha=0.9)
        ax3.set_title(f'Method: {method}
(Acc:{accuracy:.2f})', fontsize=7)
        ax3.legend(loc='upper right', fontsize=7)

        # Adjust legend colors
        legend = ax3.get_legend()
        for text in legend.get_texts():
            if text.get_text() == "Class 1 (labels)":
                text.set_color("blue")
            elif text.get_text() == "Class 0 (labels)":
                text.set_color("green")

    ax3.grid(True)
    plt.tight_layout(rect=[0, 0, 1.0, 1])

    # Clear right plot
    for widget in
canvas_frame_right.winfo_children():
        widget.destroy()

    # Display right plot
    canvas3 = FigureCanvasTkAgg(fig3,
master=canvas_frame_right)
    canvas3.draw()
    canvas3.get_tk_widget().grid(row=0, column=0,
sticky="nsew")

########################################
########################################
# Function to display current accuracy on the graph
def display_current_accuracy():
    global current_accuracy, current_params,
accuracy_data, accuracy_labels

    if current_accuracy is None:
        messagebox.showerror("Error", "Please
perform classification first.")
        return

    accuracy_data.append(current_accuracy)
    accuracy_labels.append(current_params)

    update_accuracy_graph()
```

```
# Function to delete the last added accuracy from the graph
def delete_last_accuracy():
    global accuracy_data, accuracy_labels

    if accuracy_data:
        accuracy_data.pop()
        accuracy_labels.pop()
        update_accuracy_graph()
    else:
        messagebox.showinfo("Information", "No data to delete.")

# Function to update the current accuracy graph
def update_accuracy_graph():
    pass  # Remove the function as we are not displaying the current accuracy graph

# Function to calculate and save accuracies for different labels
def calculate_accuracies_for_labels():
    global stop_requested  # Reference to the global variable
    try:
        n_samples = int(entry_samples.get()) if entry_samples.get() else 200
        noise_level = float(entry_noise.get()) if entry_noise.get() else 0.1

        # Get learning rates from entries
        learning_rate_input = entry_learning_rate.get()
        learning_rate = float(learning_rate_input) if learning_rate_input else 0.9901
        entry_learning_rate.delete(0, tk.END)
        entry_learning_rate.insert(0, str(learning_rate))

        learning_rate_spreading_input = entry_learning_rate_spreading.get()
        learning_rate_spreading = float(learning_rate_spreading_input) if learning_rate_spreading_input else 0.01
        entry_learning_rate_spreading.delete(0, tk.END)
        entry_learning_rate_spreading.insert(0, str(learning_rate_spreading))

        # Process input: remove spaces and convert to integers
        label_values = list(map(int, [x.strip() for x in entry_label_list.get().split(',')]))

        if any(value < 1 or value >= n_samples for value in label_values):
            raise ValueError("Number of initial labels must be less than total data points and positive.")

        accuracies = []

        synthetic_data_model = synthetic_data_combobox.get()
        real_data_model = real_data_combobox.get()

        if synthetic_data_model != "None":
            data_model = synthetic_data_model
        elif real_data_model != "None":
            data_model = real_data_model
        else:
            raise ValueError("Please select a data model.")

        method = method_combobox.get()

        # Get alpha parameter for Label Spreading
        alpha_input = entry_alpha.get()
        alpha = float(alpha_input) if alpha_input else 0.2

        # Generate data based on selected model
        if data_model == "Standard Moons":
            X_temp, y_temp = make_moons(n_samples=n_samples, noise=noise_level, random_state=42)
        elif data_model == "New Moons":
            X_temp, y_temp = new_make_moons(n_samples=n_samples, noise=noise_level, seed=42)
        elif data_model == "Shuffled Moons":
            X_temp, y_temp = make_moons_shuffle(n_samples=n_samples, noise=noise_level, random_state=42)
        elif data_model == "Torch Moons":
            X_temp, y_temp = make_moons_torch_shuffle(n_samples=n_samples, noise=noise_level, random_state=42)
        elif data_model == "Intersection Moons":
            X_temp, y_temp = make_moons_intersection_shuffle(n_samples=n_samples, noise=noise_level, random_state=42)
        elif data_model == "Embedded Moons":
            X_temp, y_temp = make_moons_embeded_shuffle(n_samples=n_samples, noise=noise_level, random_state=42)
        elif data_model == "Heart Disease":
            filename = download_heart_disease_data()
            X_raw, y_temp = load_and_process_heart_disease_data(filename)
            X_temp = preprocess_heart_disease_data(X_raw)
            n_samples = X_temp.shape[0]
```

```
        label_values = [lv for lv in label_values if
lv < n_samples]
        if not label_values:
            raise ValueError("Label list is empty or
all values exceed the number of data points.")
    elif data_model == "MNIST":
        X_all, y_temp = fetch_openml('mnist_784',
version=1, return_X_y=True)
        y_temp = y_temp.astype(int)
        n_samples = min(n_samples, len(y_temp))
        X_temp = X_all[:n_samples]
        y_temp = y_temp[:n_samples]

        # Scale the data
        scaler = StandardScaler()
        X_temp = scaler.fit_transform(X_temp)

        label_values = [lv for lv in label_values if
lv < n_samples]
        if not label_values:
            raise ValueError("Label list is empty or
all values exceed the number of data points.")
    else:
        raise ValueError("Please select a valid data
model.")

    n_neighbors = int(entry_neighbors.get()) if
entry_neighbors.get() else 10

    for n_labels in label_values:
        if stop_requested:
            print("Computation stopped by user.")
            break
        if data_model == "Heart Disease":
            y_labels_temp =
initialize_heart_labels(y_temp, n_labels)
        elif data_model == "MNIST":
            y_labels_temp =
initialize_mnist_labels(y_temp, n_labels)
        else:
            y_labels_temp = np.full_like(y_temp, -1)
            indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
            y_labels_temp[indices] =
y_temp[indices]

        if method == "Label Propagation":
            model = LabelPropagation()
            model.fit(X_temp, y_labels_temp)
            y_pred_temp = model.transduction_
        elif method == "Label Spreading":
            model = LabelSpreading(alpha=alpha)
            model.fit(X_temp, y_labels_temp)
            y_pred_temp = model.transduction_
        elif method == "Poisson Learning":
            tolerance = 1e-5
            max_iter = 100
            # learning_rate obtained from entry

            W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

            labeled_indices =
np.where(y_labels_temp != -1)[0]
            y_labeled =
y_labels_temp[labeled_indices]

            y_pred_temp, _ =
poisson_learning(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance, max_iter,
learning_rate)
        elif method == "Poisson Learning Adam":
            tolerance = 1e-5
            max_iter = 200
            beta = 0.01
            beta1 = 0.9
            beta2 = 0.999
            epsilon = 1e-8

            W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

            labeled_indices =
np.where(y_labels_temp != -1)[0]
            y_labeled =
y_labels_temp[labeled_indices]

            y_pred_temp, _ =
poisson_learning_adam(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance, max_iter,
beta, beta1, beta2, epsilon)
        elif method == "Poisson Learning
Nesterov":
            tolerance = 1e-5
            max_iter = 200
            beta = 0.01
            momentum = 0.9

            W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

            labeled_indices =
np.where(y_labels_temp != -1)[0]
            y_labeled =
y_labels_temp[labeled_indices]

            y_pred_temp, _ =
poisson_learning_nesterov(X_temp, y_temp, W,
```

```
labeled_indices, y_labeled, tolerance, max_iter,
beta, momentum)
        elif method == "Poisson Spreading":
            tolerance = 1e-5
            max_iter = 200
            # learning_rate_spreading obtained from
entry

            W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

            labeled_indices =
np.where(y_labels_temp != -1)[0]
            y_labeled =
y_labels_temp[labeled_indices]

            y_pred_temp =
poisson_spreading(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance, max_iter,
learning_rate_spreading)
        elif method ==
"Poisson_Spreading_Tensor":
            use_cuda = use_cuda_var.get()
            min_iter = 50
            max_iter = 1000
            tol = 1e-3

            # Compute W as a sparse matrix
            W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True)

            # Prepare labels
            labeled_indices =
np.where(y_labels_temp != -1)[0]
            y_labeled =
y_labels_temp[labeled_indices]

            # Number of classes
            num_classes = len(np.unique(y_temp))

            # Create and fit the model
            model =
Poisson_Spreading_Tensor(W=W,
use_cuda=use_cuda, min_iter=min_iter,
max_iter=max_iter, tol=tol)
            model._fit(labeled_indices, y_labeled,
num_classes)
            y_pred_temp = model.predict()
        else:
            raise ValueError("Please select a
classification method.")

        accuracy = accuracy_score(y_temp,
y_pred_temp)
            accuracies.append(accuracy)

        # Save results
        accuracy_lists_data.append(accuracies)
        label_lists_values.append(label_values)
        #accuracy_lists_params.append(f"{method},
Model: {data_model}")

        # Используйте это
        accuracy_lists_params.append(method)

        # Display results in the text box
        table_text = "\n".join([f"Labels: {n},
\nAccuracy: {a:.2f}" for n, a in
zip(label_values[:len(accuracies)], accuracies)])
        result_textbox.delete(1.0, tk.END)
        result_textbox.insert(tk.END,
f"Results:\n{table_text}")

    except ValueError as ve:
        messagebox.showerror("Error", str(ve))


# Function to display saved sequences on the graph
def display_accuracy_list():
    update_accuracy_labels_graph()

# Function to delete the last added sequence and
update the graph
def delete_last_accuracy_list():
    global accuracy_lists_data, label_lists_values,
accuracy_lists_params
    if accuracy_lists_data:
        accuracy_lists_data.pop()
        label_lists_values.pop()
        accuracy_lists_params.pop()
        update_accuracy_labels_graph()
    else:
        messagebox.showinfo("Information", "No
data to delete.")

# Function to update the accuracy graph for
different labels
def update_accuracy_labels_graph():
    # Clear old accuracy graph
    for widget in
canvas_frame_accuracy_labels.winfo_children():
        widget.destroy()

    if not accuracy_lists_data:
        return

    # Build the accuracy graph
    fig4, ax4 = plt.subplots(figsize=(6.5, 3.0))  #
Make the figure wider
```

```
    # Use colors for different methods
    for idx, (labels, accuracies, method) in
enumerate(zip(label_lists_values,
accuracy_lists_data, accuracy_lists_params)):
        color = method_colors.get(method, 'k')
        mean_accuracy = np.mean(accuracies)
        std_accuracy = np.std(accuracies)
        ax4.errorbar(labels[:len(accuracies)],
[mean_accuracy]*len(accuracies),
yerr=[std_accuracy]*len(accuracies),
                     fmt='-o', color=color, label=method,
capsize=5)

    ax4.set_xlabel('Number of Labeled Data',
fontsize=7)
    ax4.set_ylabel('Accuracy', fontsize=7)
    ax4.set_title('Accuracy for Different Number of
Labels', fontsize=7)
    ax4.grid(True)
    ax4.legend(loc='lower right', fontsize=7)

    plt.tight_layout(rect=[0, 0, 1.0, 1])

    canvas_accuracy_labels =
FigureCanvasTkAgg(fig4,
master=canvas_frame_accuracy_labels)
    canvas_accuracy_labels.draw()

canvas_accuracy_labels.get_tk_widget().grid(row=
0, column=0, sticky="nsew")

# Function to show/hide the metrics table
def toggle_metrics_table():
    global metrics_table_visible
    if metrics_table_visible:
        # Hide the metrics table
        canvas_frame_metrics_table.grid_remove()
        metrics_table_visible = False
    else:
        # Show the metrics table with grid parameters
        canvas_frame_metrics_table.grid(row=0,
column=0, padx=5, pady=5, sticky="nsew")
        metrics_table_visible = True

#

# Function to calculate and display confusion
matrix and metrics 2
def calculate_metrics():
    global y_pred, y, current_precision,
current_recall, current_f1, current_f2,
current_fbeta
    if 'y_pred' not in globals() or y_pred is None:
        messagebox.showerror("Error", "Please
perform classification first.")
        return

    # Calculate confusion matrix
    conf_matrix = confusion_matrix(y, y_pred)

########################################
########################################
    # Visualize confusion matrix
    fig5, ax5 = plt.subplots(figsize=(2.5, 2.5))
    sns.heatmap(conf_matrix, annot=True, fmt="d",
cmap="Blues",
                xticklabels=np.unique(y),
yticklabels=np.unique(y),
                ax=ax5, cbar=False)
    ax5.set_title("Confusion Matrix", fontsize=7)
    ax5.set_xlabel("Predicted Labels", fontsize=7)
    ax5.set_ylabel("True Labels", fontsize=7)

    plt.tight_layout(rect=[0, 0, 1.0, 1])

    # Clear previous confusion matrix plot
    for widget in
canvas_frame_confusion_side.winfo_children():
        widget.destroy()

    # Display confusion matrix in the upper row
    canvas5 = FigureCanvasTkAgg(fig5,
master=canvas_frame_confusion_side)
    canvas5.draw()
    canvas5.get_tk_widget().grid(row=0, column=0,
sticky="nsew")

    # Compute metrics
    data_model = synthetic_data_combobox.get() if
synthetic_data_combobox.get() != "None" else
real_data_combobox.get()
    if data_model == "MNIST":
        average_method = 'macro'
    else:
        average_method = 'binary'

    precision = precision_score(y, y_pred,
average=average_method, zero_division=0)
    recall = recall_score(y, y_pred,
average=average_method, zero_division=0)
    f1 = f1_score(y, y_pred,
average=average_method, zero_division=0)
    f2 = fbeta_score(y, y_pred, beta=2,
average=average_method, zero_division=0)
    fbeta_val = fbeta_score(y, y_pred, beta=0.5,
average=average_method, zero_division=0)

    # Store metrics in global variables
    global current_precision, current_recall,
current_f1, current_f2, current_fbeta
    current_precision = precision
    current_recall = recall
```

```
    current_f1 = f1
    current_f2 = f2
    current_fbeta = fbeta_val

    # Display metrics in the text box
    result_textbox.insert(tk.END, f"\nPrecision:
{precision:.2f}\nRecall: {recall:.2f}\nF1 Score:
{f1:.2f}\nF2 Score: {f2:.2f}\nF-beta Score (0.5):
{fbeta_val:.2f}")

# Function to add metrics to the table
def add_metrics_to_table():
    global metrics_list, current_params,
current_accuracy, current_precision,
current_recall, current_f1, current_f2,
current_fbeta

    if current_accuracy is None or current_precision
is None:
        messagebox.showerror("Error", "Please
perform classification and calculate metrics first.")
        return

    metrics_entry = {
        'method': current_params,
        'accuracy': current_accuracy,
        'precision': current_precision,
        'recall': current_recall,
        'f1_score': current_f1,
        'f2_score': current_f2,
        'f_beta': current_fbeta
    }
    metrics_list.append(metrics_entry)
    update_metrics_table()

# Function to delete last metrics from the table 2
def delete_last_metrics():
    global metrics_list
    if metrics_list:
        metrics_list.pop()
        update_metrics_table()
    else:
        messagebox.showinfo("Information", "No
metrics to delete.")

# Function to update the metrics table
def update_metrics_table():
    # Clear old table
    for widget in
canvas_frame_metrics_table.winfo_children():
        widget.destroy()

    if not metrics_list:
        return

    # Prepare table data
    columns = ['Method', 'Accuracy', 'Precision',
'Recall', 'F1 Score', 'F2 Score', 'F-Beta (0.5)']
    table_data = []
    for entry in metrics_list:
        row = [
            entry['method'],
            f"{entry['accuracy']:.2f}",
            f"{entry['precision']:.2f}",
            f"{entry['recall']:.2f}",
            f"{entry['f1_score']:.2f}",
            f"{entry['f2_score']:.2f}",
            f"{entry['f_beta']:.2f}"
        ]
        table_data.append(row)

    # Create a figure and add table
    fig, ax = plt.subplots(figsize=(6,
len(table_data)*0.5 + 1))
    ax.axis('off')

    table = ax.table(cellText=table_data,
colLabels=columns, loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(8)
    table.scale(1, 1.5)

    # Adjust column widths
    col_widths = [0.25] + [0.75/6]*6  # First column
25%, others share remaining 75%
    fig_width = fig.get_figwidth() / 6
    col_widths_in_inches = [ w * fig_width for w in
col_widths]

    for (row, col), cell in table.get_celld().items():
        cell.set_width(col_widths_in_inches[col])

    plt.tight_layout()

    # Display the figure in the canvas
    canvas_metrics = FigureCanvasTkAgg(fig,
master=canvas_frame_metrics_table)
    canvas_metrics.draw()
    canvas_metrics.get_tk_widget().grid(row=0,
column=0, sticky="nsew")


# ---------------------------------------- New Additions
Start ---------------------------------------- #


##########################################
##########################################
###############

def execute_statistics():
    global stop_requested
```

```
    try:
        n_runs = int(entry_num_runs_stat.get())
        if n_runs < 1:
            raise ValueError("Number of runs must be
at least 1.")
    except ValueError as ve:
        messagebox.showerror("Error", f"Invalid
number of runs: {ve}")
        return

    try:
        n_samples = int(entry_samples.get()) if
entry_samples.get() else 200
        noise_level = float(entry_noise.get()) if
entry_noise.get() else 0.1

        # Получение параметров обучения из
полей ввода
        learning_rate_input =
entry_learning_rate.get()
        learning_rate = float(learning_rate_input) if
learning_rate_input else 0.9901
        entry_learning_rate.delete(0, tk.END)
        entry_learning_rate.insert(0,
str(learning_rate))

        learning_rate_spreading_input =
entry_learning_rate_spreading.get()
        learning_rate_spreading =
float(learning_rate_spreading_input) if
learning_rate_spreading_input else 0.01
        entry_learning_rate_spreading.delete(0,
tk.END)
        entry_learning_rate_spreading.insert(0,
str(learning_rate_spreading))

        # Обработка списка меток: удаление
пробелов и преобразование в целые числа
        label_values = list(map(int, [x.strip() for x in
entry_label_list.get().split(',')]))

        if any(value < 1 or value >= n_samples for
value in label_values):
            raise ValueError("Number of initial labels
must be less than total data points and positive.")

        synthetic_data_model =
synthetic_data_combobox.get()
        real_data_model = real_data_combobox.get()

        if synthetic_data_model != "None":
            data_model = synthetic_data_model
        elif real_data_model != "None":
            data_model = real_data_model
        else:
            raise ValueError("Please select a data
model.")

        method = method_combobox.get()

        # Получение параметра alpha для Label
Spreading
        alpha_input = entry_alpha.get()
        alpha = float(alpha_input) if alpha_input else
0.2

        # Подготовка информации о варианте
        variant_info = f"Model: {data_model},
\nMethod: {method}, \nData: {n_samples}"

        # Подготовка списков для DataFrame
        variant_info_list = []
        num_labeled_data_list = []
        accuracy_str_list = []

        # Генерация данных на основе выбранной
модели
        if data_model == "Standard Moons":
            X_temp, y_temp =
make_moons(n_samples=n_samples,
noise=noise_level, random_state=42)
        elif data_model == "New Moons":
            X_temp, y_temp =
new_make_moons(n_samples=n_samples,
noise=noise_level, seed=42)
        elif data_model == "Shuffled Moons":
            X_temp, y_temp =
make_moons_shuffle(n_samples=n_samples,
noise=noise_level, random_state=42)
        elif data_model == "Torch Moons":
            X_temp, y_temp =
make_moons_torch_shuffle(n_samples=n_samples
, noise=noise_level, random_state=42)
        elif data_model == "Intersection Moons":
            X_temp, y_temp =
make_moons_intersection_shuffle(n_samples=n_s
amples, noise=noise_level, random_state=42)
        elif data_model == "Embedded Moons":
            X_temp, y_temp =
make_moons_embeded_shuffle(n_samples=n_sam
ples, noise=noise_level, random_state=42)
        elif data_model == "Heart Disease":
            filename = download_heart_disease_data()
            X_raw, y_temp =
load_and_process_heart_disease_data(filename)
            X_temp =
preprocess_heart_disease_data(X_raw)
            n_samples = X_temp.shape[0]
            label_values = [lv for lv in label_values if
lv < n_samples]
            if not label_values:
```

```
            raise ValueError("Label list is empty or
all values exceed the number of data points.")
        elif data_model == "MNIST":
            X_all, y_temp = fetch_openml('mnist_784',
version=1, return_X_y=True)
            y_temp = y_temp.astype(int)
            n_samples = min(n_samples, len(y_temp))
            X_temp = X_all[:n_samples]
            y_temp = y_temp[:n_samples]

            # Масштабирование данных
            scaler = StandardScaler()
            X_temp = scaler.fit_transform(X_temp)

            label_values = [lv for lv in label_values if
lv < n_samples]
            if not label_values:
                raise ValueError("Label list is empty or
all values exceed the number of data points.")
        else:
            raise ValueError("Please select a valid data
model.")

        n_neighbors = int(entry_neighbors.get()) if
entry_neighbors.get() else 10

        for n_labels in label_values:
            if stop_requested:
                print("Computation stopped by user.")
                break

            accuracies = []

            for run in range(n_runs):
                if stop_requested:
                    print("Computation stopped by user.")
                    break

                if data_model == "Heart Disease":
                    y_labels_temp =
initialize_heart_labels(y_temp, n_labels,
random_state=run)
                elif data_model == "MNIST":
                    y_labels_temp =
initialize_mnist_labels(y_temp, n_labels,
random_state=run)
                else:
                    y_labels_temp = np.full_like(y_temp,
-1)
                    indices =
np.random.choice(range(n_samples), n_labels,
replace=False)
                    y_labels_temp[indices] =
y_temp[indices]

                if method == "Label Propagation":
                    model = LabelPropagation()
                    model.fit(X_temp, y_labels_temp)
                    y_pred_temp = model.transduction_
                elif method == "Label Spreading":
                    model = LabelSpreading(alpha=alpha)
                    model.fit(X_temp, y_labels_temp)
                    y_pred_temp = model.transduction_
                elif method == "Poisson Learning":
                    tolerance_clf = 1e-5
                    max_iter_clf = 100
                    # learning_rate получен из поля
ввода

                    W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

                    labeled_indices =
np.where(y_labels_temp != -1)[0]
                    y_labeled =
y_labels_temp[labeled_indices]

                    y_pred_temp, _ =
poisson_learning(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance_clf,
max_iter_clf, learning_rate)
                elif method == "Poisson Learning
Adam":
                    tolerance_clf = 1e-5
                    max_iter_clf = 200
                    beta_clf = 0.01
                    beta1_clf = 0.9
                    beta2_clf = 0.999
                    epsilon_clf = 1e-8

                    W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

                    labeled_indices =
np.where(y_labels_temp != -1)[0]
                    y_labeled =
y_labels_temp[labeled_indices]

                    y_pred_temp, _ =
poisson_learning_adam(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance_clf,
max_iter_clf, beta_clf, beta1_clf, beta2_clf,
epsilon_clf)
                elif method == "Poisson Learning
Nesterov":
                    tolerance_clf = 1e-5
                    max_iter_clf = 200
                    beta_clf = 0.01
                    momentum_clf = 0.9
```

```
                W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

                labeled_indices =
np.where(y_labels_temp != -1)[0]
                y_labeled =
y_labels_temp[labeled_indices]

                y_pred_temp, _ =
poisson_learning_nesterov(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance_clf,
max_iter_clf, beta_clf, momentum_clf)
            elif method == "Poisson Spreading":
                tolerance_clf = 1e-5
                max_iter_clf = 200
                # learning_rate_spreading получен из
поля ввода

                W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True).toarray()

                labeled_indices =
np.where(y_labels_temp != -1)[0]
                y_labeled =
y_labels_temp[labeled_indices]

                y_pred_temp =
poisson_spreading(X_temp, y_temp, W,
labeled_indices, y_labeled, tolerance_clf,
max_iter_clf, learning_rate_spreading)
            elif method ==
"Poisson_Spreading_Tensor":
                use_cuda = use_cuda_var.get()
                min_iter = 50
                max_iter_clf = 1000
                tol_clf = 1e-3

                # Compute W as a sparse matrix
                W = kneighbors_graph(X_temp,
n_neighbors=n_neighbors, mode='connectivity',
include_self=True)

                # Prepare labels
                labeled_indices =
np.where(y_labels_temp != -1)[0]
                y_labeled =
y_labels_temp[labeled_indices]

                # Number of classes
                num_classes =
len(np.unique(y_temp))

                # Create and fit the model
                model =
Poisson_Spreading_Tensor(W=W,
use_cuda=use_cuda, min_iter=min_iter,
max_iter=max_iter_clf, tol=tol_clf)
                model._fit(labeled_indices, y_labeled,
num_classes)
                y_pred_temp = model.predict()
            else:
                raise ValueError("Please select a
classification method.")

            accuracy = accuracy_score(y_temp,
y_pred_temp)
            accuracies.append(accuracy)

        if not accuracies:
            continue  # Если не было выполнено
ни одной итерации

        # Вычисление среднего и стандартного
отклонения точности
        mean_accuracy = np.mean(accuracies)
        std_accuracy = np.std(accuracies)

        # Округление значений до двух знаков
после запятой
        mean_accuracy_rounded =
round(mean_accuracy, 2)
        std_accuracy_rounded =
round(std_accuracy, 2)

        # Форматирование строки точности
        accuracy_str =
f"{mean_accuracy_rounded:.2f}
({std_accuracy_rounded:.2f})"

        # Добавление данных в списки для
DataFrame
        variant_info_list.append(variant_info)
        num_labeled_data_list.append(n_labels)
        accuracy_str_list.append(accuracy_str)

    if not num_labeled_data_list:
        messagebox.showinfo("Information", "No
data to display.")
        return

    # Создание DataFrame для статистики
    stats_df = pd.DataFrame({
        'Variant': variant_info_list,
        'Labeled Data': num_labeled_data_list,
        'Accuracy (Std)': accuracy_str_list
    })

    # Транспонирование DataFrame с
использованием pivot
```

```
        pivot_df = stats_df.pivot(index='Variant',
columns='Labeled Data', values='Accuracy
(Std)').reset_index()

        # Сортировка столбцов по числовому
порядку
        labeled_data_sorted = sorted([col for col in
pivot_df.columns if isinstance(col, int)])
        pivot_df = pivot_df[['Variant'] +
labeled_data_sorted]

        # Вычисление относительных ширин
столбцов
        num_columns = len(pivot_df.columns)
        col_widths = []
        for i in range(num_columns):
            if i == 0:
                col_widths.append(2.5)  # Первый
столбец в 2 раза шире
            else:
                col_widths.append(1.5)  # Остальные
столбцы в 1.5 раза шире

        # Нормализация ширин, чтобы сумма была
равна 1
        total_width = sum(col_widths)
        col_widths_normalized = [w / total_width for
w in col_widths]

        # Определение количества строк
(вариантов)
        num_rows = len(pivot_df)

        # Вычисление необходимой высоты
фигуры
        # Предположим, что каждая строка
требует 0.5 дюйма в высоту,
        # и добавляем дополнительное
пространство для заголовков и границ
        fig_height = max(2, num_rows * 0.5 + 2)

        # Отображение таблицы статистики
        fig_table, ax_table = plt.subplots(figsize=(8,
fig_height))  # Увеличен размер для лучшего
отображения
        ax_table.axis('off')

        # Создание таблицы с
транспонированными данными и заданными
ширинами столбцов
        table =
ax_table.table(cellText=pivot_df.values,
                               colLabels=pivot_df.columns,
                               loc='center',

colWidths=col_widths_normalized)

        # Настройка параметров таблицы
        table.auto_set_font_size(False)
        table.set_fontsize(8)  # Уменьшить размер
шрифта при необходимости
        table.scale(1, 2)  # Увеличить вертикальный
масштаб для увеличения высоты строк

        plt.tight_layout()

        # Очистка предыдущей таблицы и графика
        for widget in
canvas_frame_accuracy_labels.winfo_children():
            widget.destroy()

        # Отображение таблицы в холсте
        canvas_table =
FigureCanvasTkAgg(fig_table,
master=canvas_frame_accuracy_labels)
        canvas_table.draw()
        canvas_table.get_tk_widget().grid(row=0,
column=0, sticky="nsew")

        # Построение графика со средними
значениями и стандартным отклонением
        fig_graph, ax_graph = plt.subplots(figsize=(8,
4))  # Увеличен размер графика

        # Извлечение числовых значений из строк
'Accuracy (Std)'
        means = []
        stds = []
        for acc_str in accuracy_str_list:
            mean, std = acc_str.split('(')
            mean = float(mean)
            std = float(std.strip(')'))
            means.append(mean)
            stds.append(std)

        ax_graph.errorbar(num_labeled_data_list,
means, yerr=stds,
                          fmt='-o', ecolor='r', capsize=5,
label='Mean Accuracy')

        ax_graph.set_xlabel('Number of Labeled
Data', fontsize=12)
        ax_graph.set_ylabel('Accuracy', fontsize=12)
        ax_graph.set_title('Mean Accuracy with Std
Dev', fontsize=14)
        ax_graph.grid(True)
        ax_graph.legend()

        plt.tight_layout()

        # Отображение графика ниже таблицы
```

```
        canvas_graph =
FigureCanvasTkAgg(fig_graph,
master=canvas_frame_accuracy_labels)
        canvas_graph.draw()
        canvas_graph.get_tk_widget().grid(row=1,
column=0, sticky="nsew")


    except ValueError as ve:
        messagebox.showerror("Error", f"Invalid
number of runs: {ve}")
        return

# ------------------------------------- New Additions
End ------------------------------------- #

# Create the main application window
window = tk.Tk()
window.title("2024 Ruchkin A.      Intelligent
Interactive System for Investigating Generalized
Diffuse Label Propagation Methods on Graphs in
Classification Tasks of Real and Synthetic Datasets
v 1.5.6.1 (stat)")
window.geometry("1800x900")  # Window size

# Configure the main window grid
window.grid_rowconfigure(0, weight=1)
window.grid_rowconfigure(1, weight=0)
window.grid_columnconfigure(0, weight=1)

# Main frame
main_frame = tk.Frame(window)
main_frame.grid(row=0, column=0,
sticky="nsew")
main_frame.grid_rowconfigure(0, weight=1)
main_frame.grid_columnconfigure(0, weight=1)
main_frame.grid_columnconfigure(1, weight=3)

# Left frame for data input and text information
left_frame = tk.Frame(main_frame, padx=5,
pady=5)
left_frame.grid(row=0, column=0, sticky="nsew")
left_frame.grid_rowconfigure(0, weight=1)
left_frame.grid_columnconfigure(0, weight=1)

# Right frame for plots
right_frame = tk.Frame(main_frame, padx=5,
pady=5)
right_frame.grid(row=0, column=1,
sticky="nsew")
right_frame.grid_rowconfigure(0, weight=1)  #
Upper container with plots
right_frame.grid_rowconfigure(1, weight=1)
right_frame.grid_columnconfigure(0, weight=1)

# Left frame grid
# Upper part of the window: input fields and
buttons (inside left_frame)
top_frame = tk.Frame(left_frame)
top_frame.grid(row=0, column=0, sticky="nsew",
pady=10)
top_frame.grid_rowconfigure(0, weight=1)
top_frame.grid_columnconfigure(0, weight=1)
top_frame.grid_columnconfigure(1, weight=1)  #
Add a second column for entries


# 1. Classification Result
result_label = tk.Label(top_frame,
text="Classification Result:")
result_label.grid(row=0, column=0, sticky="w",
padx=5, pady=(0,5))
result_textbox = tk.Text(top_frame, height=10,
width=20)
result_textbox.grid(row=0, column=1, sticky="w",
padx=2, pady=(0,5))


# 2. Number of Data
label_samples = tk.Label(top_frame,
text="Number of Data:")
label_samples.grid(row=1, column=0, sticky="w",
padx=5, pady=(0,5))
entry_samples = tk.Entry(top_frame, width=20)
entry_samples.grid(row=1, column=1, sticky="w",
padx=2, pady=(0,5))
entry_samples.insert(0, "2000")  # Set default
value

# 3. Initial Labels
label_labels = tk.Label(top_frame, text="Initial
Labels:")
label_labels.grid(row=2, column=0, sticky="w",
padx=5, pady=(0,5))
entry_labels = tk.Entry(top_frame, width=20)
entry_labels.grid(row=2, column=1, sticky="w",
padx=2, pady=(0,5))
entry_labels.insert(0, "20")  # Set default value

# 4. Noise Level
label_noise = tk.Label(top_frame, text="Noise
Level:")
label_noise.grid(row=3, column=0, sticky="w",
padx=5, pady=(0,5))
entry_noise = tk.Entry(top_frame, width=20)
entry_noise.grid(row=3, column=1, sticky="w",
padx=2, pady=(0,5))
entry_noise.insert(0, "0.1")  # Set default value

# 5. Alpha for Label Spreading
label_alpha = tk.Label(top_frame, text="Alpha
(Label Spreading):")
```

```
label_alpha.grid(row=4, column=0, sticky="w",
padx=5, pady=(0,5))
entry_alpha = tk.Entry(top_frame, width=20)
entry_alpha.grid(row=4, column=1, sticky="w",
padx=2, pady=(0,5))
entry_alpha.insert(0, "0.2")  # Set default value

# 6. Learning Rate
label_learning_rate = tk.Label(top_frame,
text="Alpha (Poisson Learning)")
label_learning_rate.grid(row=5, column=0,
sticky="w", padx=5, pady=(0,5))
entry_learning_rate = tk.Entry(top_frame,
width=20)
entry_learning_rate.grid(row=5, column=1,
sticky="w", padx=2, pady=(0,5))
entry_learning_rate.insert(0, "0.99")  # Set default
value

# 7. Learning Rate Spreading
label_learning_rate_spreading =
tk.Label(top_frame, text="Alpha (Poisson
Spreading):")
label_learning_rate_spreading.grid(row=6,
column=0, sticky="w", padx=5, pady=(0,5))
entry_learning_rate_spreading =
tk.Entry(top_frame, width=20)
entry_learning_rate_spreading.grid(row=6,
column=1, sticky="w", padx=2, pady=(0,5))
entry_learning_rate_spreading.insert(0, "0.01")  #
Set default value


# 8. Synthetic Data Model
label_synthetic_data_model = tk.Label(top_frame,
text="Synthetic Data Model:")
label_synthetic_data_model.grid(row=7,
column=0, sticky="w", padx=5, pady=(0,5))

synthetic_data_combobox = ttk.Combobox(
    top_frame,
    values=[
        "None",
        "Standard Moons",
        "New Moons",
        "Shuffled Moons",
        "Torch Moons",
        "Intersection Moons",
        "Embedded Moons"
    ],
    state="readonly"
)
synthetic_data_combobox.set("Standard Moons")
synthetic_data_combobox.grid(row=7, column=1,
sticky="w", padx=2, pady=(0,5))

# 9. Real Data Model
label_real_data_model = tk.Label(top_frame,
text="Real Data Model:")
label_real_data_model.grid(row=8, column=0,
sticky="w", padx=5, pady=(0,5))

real_data_combobox = ttk.Combobox(
    top_frame,
    values=[
        "None",
        "Heart Disease",
        "Cardiovascular Disease",
        "MNIST"
    ],
    state="readonly"
)
real_data_combobox.set("None")
real_data_combobox.grid(row=8, column=1,
sticky="w", padx=2, pady=(0,5))

# 10. Show Data Button
generate_button = tk.Button(top_frame,
text="Show Data Model",
command=generate_data)
generate_button.grid(row=9, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,10))

# 11. Number of Neighbors
label_neighbors = tk.Label(top_frame,
text="Number of Neighbors:")
label_neighbors.grid(row=10, column=0,
sticky="w", padx=5, pady=(0,5))
entry_neighbors = tk.Entry(top_frame, width=20)
entry_neighbors.grid(row=10, column=1,
sticky="w", padx=2, pady=(0,5))
entry_neighbors.insert(0, "10")  # Set default value

# 12. Build KNN Graph Button
knn_button = tk.Button(top_frame, text="Build
KNN Graph", command=build_knn_graph)
knn_button.grid(row=11, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,10))

# 13. Classification Method
label_method = tk.Label(top_frame,
text="Classification Method:")
label_method.grid(row=12, column=0, sticky="w",
padx=5, pady=(0,5))
method_combobox = ttk.Combobox(
    top_frame,
    values=[
        "Label Propagation",
        "Label Spreading",
        "Poisson Learning",
        "Poisson Spreading",
        "Poisson Learning Adam",
```

```
        "Poisson Learning Nesterov",
        "Poisson_Spreading_Tensor"
    ],
    state="readonly"
)
method_combobox.set("Label Propagation")
method_combobox.grid(row=12, column=1,
sticky="w", padx=2, pady=(0,5))

# 14. Start Classification Button
start_button = tk.Button(top_frame, text="Start
Classification", command=classify_two_moons)
start_button.grid(row=13, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,5))

# 15. Stop Button
stop_button = tk.Button(top_frame, text="Stop
Classification", command=stop_computation)
stop_button.grid(row=13, column=1,
columnspan=2, sticky="w", padx=2, pady=(0,5))

# Add Use CUDA Checkbox
use_cuda_var = tk.BooleanVar(value=True)
use_cuda_checkbox = tk.Checkbutton(top_frame,
text="Use CUDA", variable=use_cuda_var)
use_cuda_checkbox.grid(row=14, column=1,
sticky="w", padx=2, pady=(0,5))

# 16. Calculate Metrics Button
metrics_button = tk.Button(top_frame,
text="Calculate Metrics",
command=calculate_metrics)
metrics_button.grid(row=15, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,10))

# 17. Add Metrics Button
add_metrics_button = tk.Button(top_frame,
text="Add Metrics",
command=add_metrics_to_table)
add_metrics_button.grid(row=16, column=0,
sticky="w", padx=5, pady=(5,5))

# 18. Delete Metrics Button
delete_metrics_button = tk.Button(top_frame,
text="Delete Metrics",
command=delete_last_metrics)
delete_metrics_button.grid(row=16, column=1,
sticky="w", padx=2, pady=(5,5))

# Add the Show/Hide Metrics Table button
toggle_metrics_button = tk.Button(top_frame,
text="Show/Hide Metrics Table",
command=toggle_metrics_table)
toggle_metrics_button.grid(row=17, column=0,
columnspan=2, sticky="w", padx=5, pady=(5,5))

# 19. Enter List of Labels
label_label_list = tk.Label(top_frame, text="Enter
List of Labels\n(e.g., 2,4,6,8,10):")
label_label_list.grid(row=18, column=0,
sticky="w", padx=5, pady=(0,5))
entry_label_list = tk.Entry(top_frame, width=15)
entry_label_list.grid(row=18, column=1,
sticky="w", padx=2, pady=(0,5))
entry_label_list.insert(0, "2,4,6,8,10")

# 20. Calculate Accuracy for the List of Labels
Button
accuracy_button = tk.Button(top_frame,
text="Calculate Accuracy",
command=calculate_accuracies_for_labels)
accuracy_button.grid(row=19, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,5))

# 21. Display List and Delete List Buttons
display_list_button = tk.Button(top_frame,
text="Display List Accuracy",
command=display_accuracy_list)
display_list_button.grid(row=20, column=0,
sticky="w", padx=5, pady=(5,5))
delete_list_button = tk.Button(top_frame,
text="Delete List Accuracy",
command=delete_last_accuracy_list)
delete_list_button.grid(row=20, column=1,
sticky="w", padx=2, pady=(5,5))

# ---------------------------------------- New Additions
Start ---------------------------------------- #

# 22. Number of Runs for Statistics
label_num_runs_stat = tk.Label(top_frame,
text="Number of Runs for Statistics:")
label_num_runs_stat.grid(row=21, column=0,
sticky="w", padx=5, pady=(0,5))
entry_num_runs_stat = tk.Entry(top_frame,
width=20)
entry_num_runs_stat.grid(row=21, column=1,
sticky="w", padx=2, pady=(0,5))
entry_num_runs_stat.insert(0, "5")  # Set default
value

# 23. Execute Statistics Button
execute_statistics_button = tk.Button(top_frame,
text="Execute Statistics",
command=execute_statistics)
execute_statistics_button.grid(row=22, column=0,
columnspan=2, sticky="w", padx=5, pady=(0,10))

# ---------------------------------------- New Additions
End ---------------------------------------- #

# Frames for plots (inside right_frame)
```

```
# Create container frame for upper plots (4 plots side by side)
upper_frame = tk.Frame(right_frame)
upper_frame.grid(row=0, column=0, sticky="nsew", pady=5)
upper_frame.grid_rowconfigure(0, weight=1)
upper_frame.grid_columnconfigure(0, weight=1)
upper_frame.grid_columnconfigure(1, weight=1)
upper_frame.grid_columnconfigure(2, weight=1)
upper_frame.grid_columnconfigure(3, weight=1)
# Added fourth column

# Frame for initial data (left)
canvas_frame_left = tk.Frame(upper_frame)
canvas_frame_left.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")
canvas_frame_left.grid_rowconfigure(0, weight=1)
canvas_frame_left.grid_columnconfigure(0, weight=1)

# Frame for classification results (center)
canvas_frame_right = tk.Frame(upper_frame)
canvas_frame_right.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")
canvas_frame_right.grid_rowconfigure(0, weight=1)
canvas_frame_right.grid_columnconfigure(0, weight=1)

# Frame for KNN graph (right)
canvas_frame_graph = tk.Frame(upper_frame)
canvas_frame_graph.grid(row=0, column=2, padx=5, pady=5, sticky="nsew")
canvas_frame_graph.grid_rowconfigure(0, weight=1)
canvas_frame_graph.grid_columnconfigure(0, weight=1)

# Frame for confusion matrix (fourth plot)
canvas_frame_confusion_side = tk.Frame(upper_frame)
canvas_frame_confusion_side.grid(row=0, column=3, padx=5, pady=5, sticky="nsew")
canvas_frame_confusion_side.grid_rowconfigure(0, weight=1)
canvas_frame_confusion_side.grid_columnconfigure(0, weight=1)

# Create container frame for lower plots (2 plots side by side)
lower_frame = tk.Frame(right_frame)
lower_frame.grid(row=1, column=0, sticky="nsew", pady=5)
lower_frame.grid_rowconfigure(0, weight=1)
lower_frame.grid_columnconfigure(0, weight=1)

# Frame for metrics table (row=0)
canvas_frame_metrics_table = tk.Frame(lower_frame)
canvas_frame_metrics_table.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")
canvas_frame_metrics_table.grid_rowconfigure(0, weight=1)
canvas_frame_metrics_table.grid_columnconfigure(0, weight=1)

# Frame for accuracy graph for different labels (row=1)
canvas_frame_accuracy_labels = tk.Frame(lower_frame)
canvas_frame_accuracy_labels.grid(row=1, column=0, padx=5, pady=5, sticky="nsew")
canvas_frame_accuracy_labels.grid_rowconfigure(0, weight=1)
canvas_frame_accuracy_labels.grid_columnconfigure(0, weight=1)

# Add exit button at the bottom of the window
exit_button = tk.Button(window, text="Exit", command=window.quit)
exit_button.grid(row=1, column=0, sticky="sw", padx=5, pady=10)

# Initialize variables for current accuracy and parameters
current_accuracy = None
current_params = None
current_precision = None
current_recall = None
current_f1 = None
current_f2 = None
current_fbeta = None


# Run the application
window.mainloop()
```